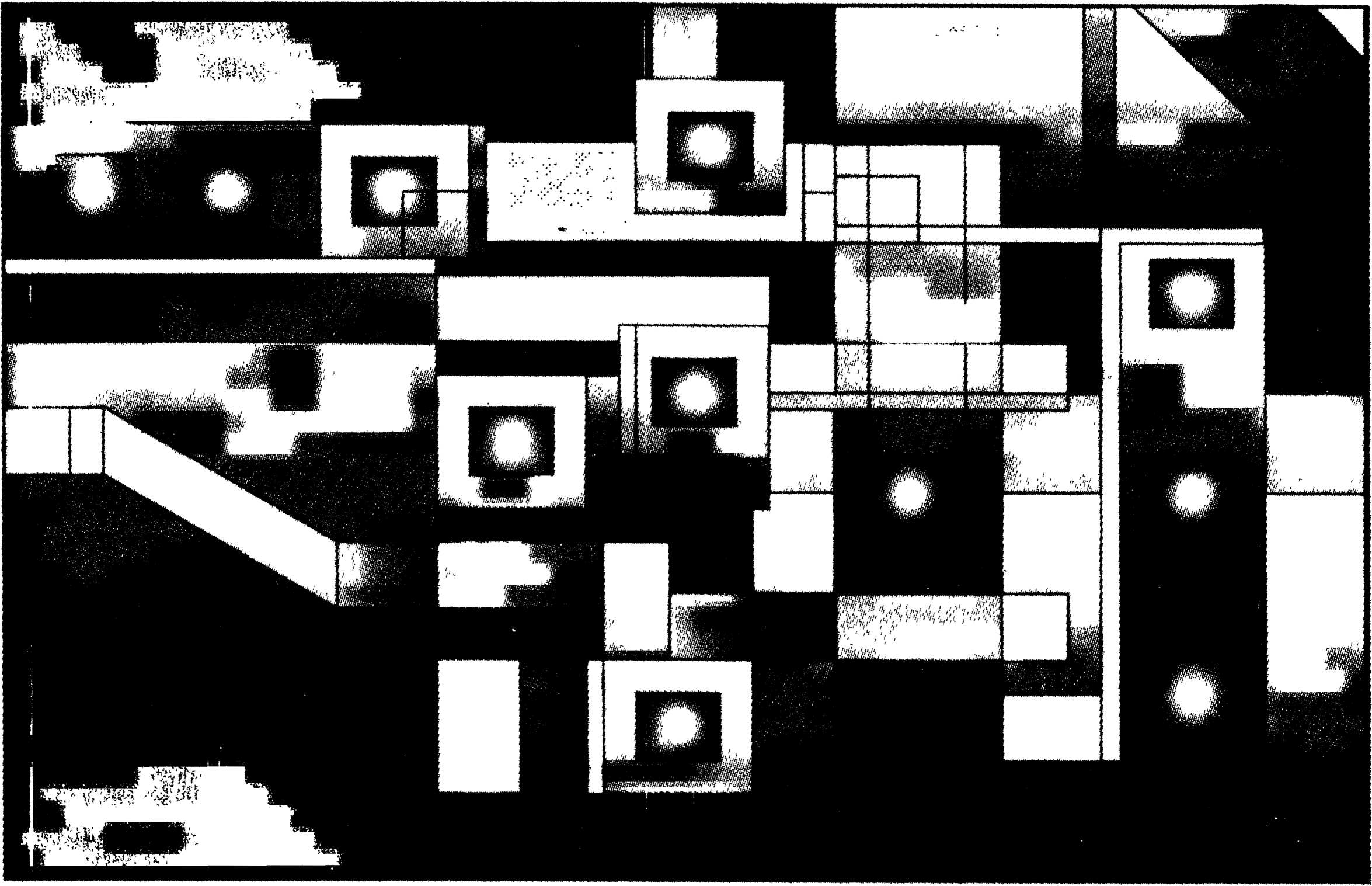




*Preliminary Technical Manual*



**Z280**

MPU Microprocessor Unit

rwi neumuljIr

**j И ELEKTRONIK-BAUTEILE**

EscnenstraBe 2 • Postfach 1252 • 8028 Taufkirchen bei Munchen

Telefon (0 89) 612 08-0 • Telex 5 22106 • Telefax (0 89) 612 08-2^L

Table of Contents

Chapter 1. Z280 Architectural Overview

* 1. [Introduction 1-1](#bookmark43)
  2. [MPU Architectural Features 1-2](#bookmark3)

t t •

\* • • \* »

* + 1. [System and User Modes 1-2](#bookmark5)
    2. [Address Spaces 1-2](#bookmark39)
    3. [Data Types 1-2](#bookmark10)
    4. Addressing Modes 1-3
    5. [Instruction Set 1-3](#bookmark467)
    6. Exception Conditions 1-3
    7. Memory Management 1-3
    8. [Cache Memory V4](#bookmark410)

[• 1.2.9 Refresh 1-4](#bookmark17)

* + 1. [On-Chip Peripherals 1-4 .](#bookmark19)
    2. Multiprocessor Mode 1-4

[. 1.2.12 Extended Instruction Facility 1-4](#bookmark21)

• . •

[1.3 Benefits of the Architecture 1-5](#bookmark24)

***• • I •***

* + 1. [High Throughput .1-5](#bookmark26)
    2. [Integration of System Functions 1-5](#bookmark28)
    3. [Operating System Support .......... 1-5](#bookmark23)
    4. [Code Density 1-5](#bookmark32)
    5. [Compiler Efficiency ...... 1-5](#bookmark34)

[1.4 Summary ....1-6](#bookmark36)

• / ■

1. [**Address Spaces**](#bookmark42)
   1. [Introduction 2-1](#bookmark59)
   2. CPU Register File 2-1
   3. [CPU Control Registers 2-2](#bookmark55)
   4. Memory Address Spaces ............ 2-3
   5. [I/O Address Space 2-4](#bookmark50)
2. [**CPU Control Registers**](#bookmark58)
   1. [Introduction 3-1](#bookmark364)
   2. [System Configuration Registers 3-1](#bookmark62)
      1. [Bus Timing and Initialization Register 3-1](#bookmark64)
      2. Bus Timing and Control Register 3-2
      3. Local Address Register 3-3
      4. Cache Control Register 3-3

**Table of Contents** (Continued)

* 1. [System Status Registers ..... 3-4](#bookmark73)
     1. [Master Status Register 3-4](#bookmark75)
     2. [interrupt Status Register 3-4](#bookmark77)
     3. Interrupt/Trap Vector Table Pointer 3-5
     4. [1/0 Page Register 3-5](#bookmark82)
     5. Trap Control Register 3-5
     6. [System Stack Limit Register 3-6](#bookmark87)

1. **Addressing Modes and Data Types**
   1. [Introduction . 4-1](#bookmark408)
   2. [Addressing Mode Descriptions . . . . ; 4-1](#bookmark96)

[" 4.2.1 Register (R, RX) . . ". 4-1](#bookmark104)

* + 1. [, Immediate (IM) 4-1](#bookmark98)
    2. [Indirect Register (IR) 4-2](#bookmark100)
    3. [Direct Address (DA) 4-2](#bookmark102)
    4. [Indexed (X) 4-3](#bookmark106)
    5. [Short Index (SX) 4-3](#bookmark108)
    6. Relative Address (RA) 1 ' 4-4
    7. [Stack Pointer Relative (SR) 4-5](#bookmark114)
    8. [Base Index (BX) ; 4-5](#bookmark117)

Data Types

1. **Instruction Set**
   1. [Introduction 5-1](#bookmark487)
   2. Processor Flags 5-1
      1. [Carry Flag (C) 5-1](#bookmark124)
      2. Add/Subtract Flag (N) 5-1
      3. [Parity/Overflow Flag (P/V) 5-2](#bookmark122)
      4. Half-Carry Flag (H) . 5-2

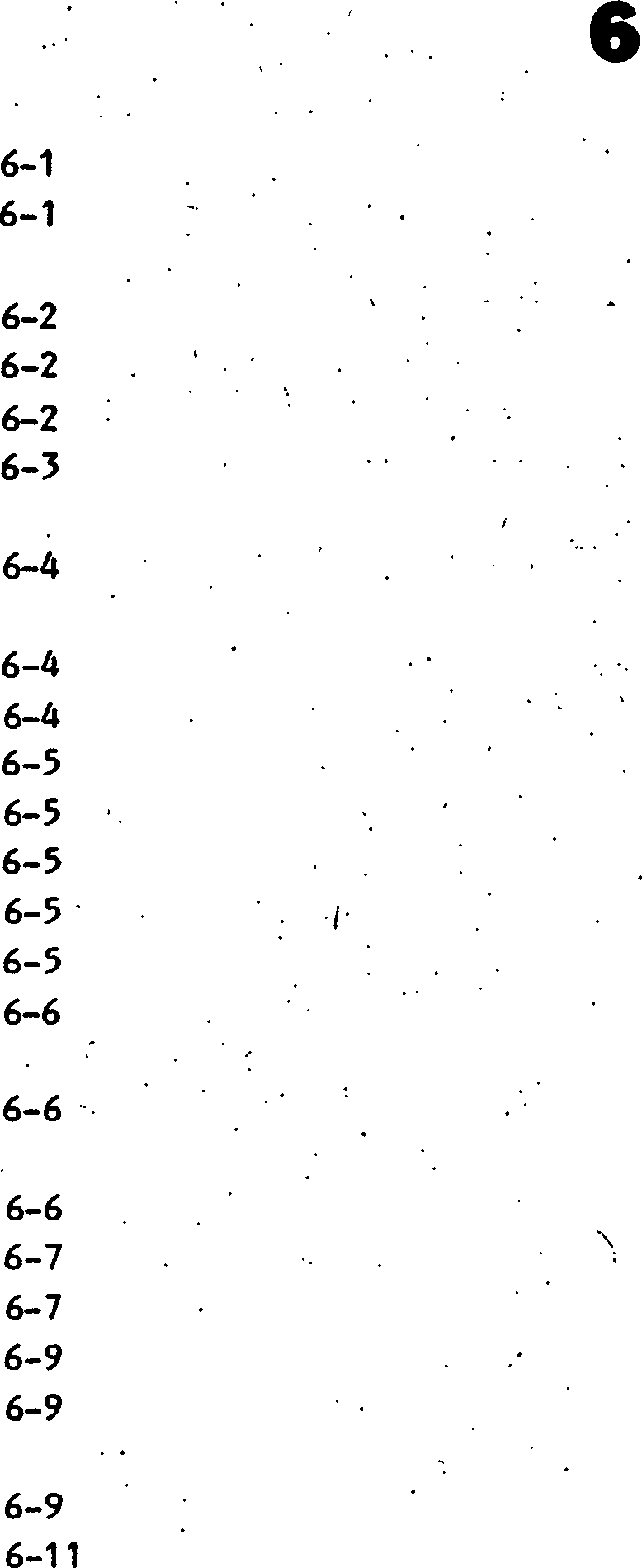
[, 5.2.5 Zero Flag (Z) 5-2](#bookmark126)

* + 1. [Sign Flag (S) . 5-2](#bookmark128)
    2. [Condition Codes 5-2](#bookmark130)

4 w 5.3 Instruction Execution and Exceptions . 5-3

X • \

* + 1. Instruction Execution and Interrupts 5-3
    2. Instruction Execution and Traps 5-3



Instruction Set Functional Groups

* + 1. [8-bit Load Group 5-4](#bookmark134)
    2. 16-bit Load and Exchange Group 5-5
    3. [Block Transfer and Search Group 5-5](#bookmark138)
    4. [8-bit Arithmetic and Logic Group . 5-6](#bookmark140)
    5. [16-bit Arithmetic Group 5-6](#bookmark142)
    6. [Bit Manipulation, Rotate and Shift Group ......... 5-7](#bookmark145)
    7. [Program Control Group 5-7](#bookmark148)
    8. [Input/Output Instruction Group 5-9](#bookmark151)
    9. [CPU Control Group 5-9](#bookmark153)
    10. [Extended Instruction Group 5-10](#bookmark462)

• [[1]](#footnote-1) [[2]](#footnote-2) [[3]](#footnote-3) [[4]](#footnote-4) [[5]](#footnote-5) [[6]](#footnote-6)

* 1. Notation and Binary Encoding 5-10
  2. Instruction Set 5-13

***!***

**Table of Contents** (Continued)

1. Memory Management Uhit
   1. [Introduction 7-1](#bookmark93)
   2. [MMU Architecture 7-1](#bookmark396)
   3. Page Description Registers 7-2
   4. [Address Translation 7-3](#bookmark398)
      1. Address Translation without Program/Data Separation .... 7-3
      2. [Address Translation with Program/Data Separation 7-4](#bookmark400)
   5. [MMU Control Registers 7-5](#bookmark48)
   6. Accessing Page Descriptor Registers 7-6
      1. Descriptor Select Port 7-6
      2. Block Move Port 7-6
      3. Invalidation Port . . ........... 7-6
   7. Instruction Aborts 7-7
2. **On-Chip Memory**
   1. Introduction ' 8-1
   2. [Cache Memory Mode ....' 8-1](#bookmark15)
   3. [Fixed-Address Mode 8-4](#bookmark413)
3. **On-Chip Peripherals**
   1. Introduction 9-1
   2. [Clock Oscillator 9-1](#bookmark421)
   3. [Refresh Controller 9-1](#bookmark419)
   4. Counter/Timers 9-2

**z**

* + 1. [Counter/Timer Operating Modes \* 9-3](#bookmark427)
    2. [Gates and Triggers 9-3](#bookmark424)
    3. Terminal Count Condition 9-4
    4. Counter/Timer Registers 9-4
    5. Linking Counter/Timers 9-7
    6. [Counter/Timer Sequence of Events . 9-7](#bookmark432)
  1. [DMA Channels 9-9](#bookmark434)
     1. Types of DMA Operations 9-10
     2. DMA Transfer Modes 9-10
     3. [End-о f-Process 9-11](#bookmark437)
     4. Priority Resolution 9-12
     5. DMA Linking 9-12
     6. DMA Registers 9-13
     7. DMA Sequence of Events 9-15
     8. DMA Programming: Linked DMAs 9-16
     9. DMA Programming: DMAs Linked to UART 9-17
  2. DART
     1. Transmitter Operation
     2. Receiver Operation
     3. DART Registers
     4. [UART Operation](#bookmark457)
  3. [UART Bootstrapping Option](#bookmark457)

1. Multiprocessor Configurations
   1. Introduction 10-1 \

9-17

9-17 9-18 9-18 9-21

9-21

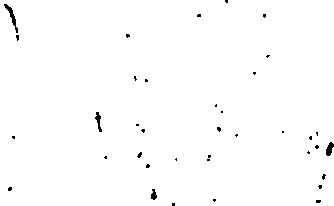
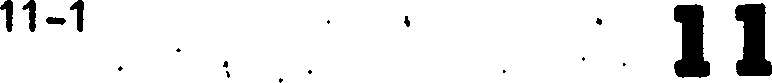
**10**

* 1. Slave Processors 10-1 .
  2. Tightly Coupled Multiple Processors .... 10-2
     1. The Local Address Register 10-2
     2. Bus Request Protocols 10-2
     3. Examples of the Use of the Global Bus 10-4

t

\* \* \* \* .

* 1. [Loosely Coupled Multiple CPUs 10-6](#bookmark458)
  2. Coprocessors and the Extended Processing Architecture ...... 10-6
     1. Extended Instructions 10-6 ..
     2. Extended Instruction Execution Sequence ......... 10-7

1. **Reset**
2. [**Z80 Bus External Interface**](#bookmark469)
   1. Introduction 12-1
   2. Bus Operations 12-2
   3. Pin Descriptions 12-3
   4. Bus Configuration and Timing 12-4
   5. [Transactions 12-4](#bookmark470)

*, B \* f \**

* + 1. [Memory Transactions 12-5](#bookmark474)
    2. RETI Transactions .. ..... 12-9
    3. [Halt and Refresh Transactions 12-9](#bookmark497)
    4. [1/0 Transactions 12-10](#bookmark499)
    5. Interrupt Acknowledge Transactions 12-12
    6. DMA Flyby Transactions 12-13
  1. [Requests 12-14](#bookmark509)
     1. Interrupt Requests 12-14.
     2. [Local Bus Requests ' 12-15](#bookmark479)
     3. [Global Bus Requests 12-15](#bookmark481)

**Table of Contents** (Continued)

1. Z-BUS External Interface '
   1. Introduction ....... 13-1
   2. Bus Operations 13-2
   3. Pin Descriptions . . . 13-3
   4. Bus Configuration and Timing 13-4
   5. Transactions ..... . 13-4

» / • .

* + 1. Memory Transactions ...... 13-5
    2. Halt and Refresh Transactions 13-10
    3. [I/O Transactions 13-11](#bookmark490)
    4. Interrupt Acknowledge Transactions 13-13

• 13.5.5 Extended Processing Unit (EPU) Transactions 13-14

13.5.6 DMA Flyby Transactions 13-17

. « \* .

13.6 Requests ......... ' 13-18

* + 1. [Interrupt Requests 13-19](#bookmark477)
    2. [Local Bus Requests 13-19](#bookmark513)
    3. Global Bus Requests 13-19

[**Appendix A. Z80/Z280 Compatibility ’** A-1](#bookmark517)

• • \* z

**Appendix B. Z280 MPU Instruction Formats** B-1

**. Appendix C. Instructions in Alphabetic Order** C-1

‘ .

**Appendix D. Instructions in Numeric Order** D-1

*:\* • \* • ... • \** \* / • \* \ • Appendix E. Instruction Timing . . E-1

[**. Appendix F. Compatible Peripheral Families** F-1](#bookmark522)

. • I

**Glossary** G-1

**Index** 1-1

LIST OF ILLUSTRATIONS AND TABLES

Number

Page Number

3-10.

CPU Control Registers

Numbering of Bits Within a Byte.....

Formats, Multiple-Byte Data Elements in Memory

Bus Timing and Control Register Local Address Register Cache Control Register.... Master Status Register Interrupt Status Register

I/O Page Register

Instruction Execution Sequence Format of Saved Status on System Stack Due to a Mode 3 Interrupt

Address Translation Without Program/Data Separation Address Translation With Program/Data Separation... MMU Master Control Register..

Refresh Rate Register

Counter Operation With Gate Only Counter Operation With Trigger Only Counter Operation With Gate and Trigger Counter/Timer Configuration Register Counter/Timer Command/Stat us Register Modes of Operation DMA Master Control Register Transaction Descriptor Register Source & Destination Address Registers Format General Format, Asynchronous Transmission....

9-10.

9-14. UART Configuration Register 9-18

[9-15. Transmitter Control/Status Register 9-19](#bookmark451)

9-16. Receiver Control/Status Register... 9-20

10-1. Multiprocessor Configurations 10-1

10-2. Local Address Register 10-2

10-3. State Diagram for CPU Bus Request Protocol 10-3

10-4. Tightly Coupled Processors With Shared Global Memory 10-4

10-5. Tightly Coupled Processors Without Global Memory 10-5

10-6. Z280 MPU as an 1/0 Processor 10-5

**Table of Contents** (Continued)

10-7. EPU Connection in Z280 MPU System

10-8. CPU-EPU Instruction Execution Sequence ."......

12-1. Z80 Bus Configuration (Input OPT tied to GND)

1. Pin Functions
2. Pin Assignments.....

12-2. Memory Read Timing..

12-3. Memory Write Timing

12-4. Memory Read Timing W/One External Wait State

12-5. Memory Write Timing W/One External Wait State

12-6. Memory Read Timing W/One Internal Wait State

12-7. RETI Read Timing '

12-8. Halt Timing

12-9. Memory Refresh Timing

12-10. I/O Read Timing

12-11. I/O Write Timing.

12-12. Interrupt Acknowledge Sequence

12-13. On-Chip DMA Channel Flyby Memory Read Transaction.....

12-14. On-Chip DMA Channel Flyby Memory Write Transaction

12-15. Multiprocessor Mode Timing

13-1. Z-BUS Configuration (Input OPT tied to +5V or not connected) . “ • a) Pin Functions.

1. Pin Assignments

13-2. Memory Read Timing

13-3. Memory Write Timing

13-4. Memory Read Timing With External Wait Cycle

13-5. Memory Write Timing With External Wait Cycle

13-6. Memory Read Timing With Internal Wait, Cycle

13-7. Burst Memory Read Timing

13-8. Halt Timing...

13-9. Memory Refresh Timing

13-10. I/O Read Timing

13-11. I/O Write Timing

13-12. Interrupt Acknowledge Timing

13-13. Memory to EPU Timing

13-14. EPU Write To Memory

13-15. EPU To CPU Timing

13-16. PAUSE Timing .•

13-17. On-Chip DMA Channel Flyby Memory Read Transaction

13-18. On-Chip DMA Channel Flyby Memory Write Transaction....

13-19. Multiprocessor Mode Timing

• \* » ■ . . . . . ■ . ■ . • . • • • ■ • .«

10-6 .

10-7 ;

■ '

12-1 .

12-1 12-5 ...

12-6

12-6 ...

12-7

12-7

12-8

12-9

12-10 ;

12-11 • .

12-11

12-12 ;

12-13

12-14 12-15 ’

13-1 ■

13-1 ;

13-6

13-7 . ■

13-7 4 • • :

13-8 '

13-8 .

13-9

13-10

13-11

13-12 . • •

13-12 . ‘

13-13 .

13-14 '

13-15

13-16 , . '

13-16

13-17 •

13-18 . ' •

13-19 .

Page

Table

Number

3-1 .

3-2.

3-3.

3-4.

3-5.

5-1 .

5-2.

5-3.

5-4.

5-5.

5-6.

5-7.

5-8.

5-9.

5-10. 5-11. 5-12.

6-1 .

6-2.

6-3.

6-4.

6-5.

7-1 .

7-2.

7-3.

8-1.

8-2.

8-3.

9-1 .

9-2.

9-3.

9-4.

9-5.

9-6 .

9-7.

9-8.

9-9.

9-10.

9-11 . 9-12 .

Number

CS Field, Bus Timing & Initialization Register 3-1

LM Field, Bus Timing & Initialization Register 3-1

I/O Field of Bus Timing and Control Register.... 3-2

HM Field of Bus Timing and Control Register 3-2

DC Field of Bus Timing and Control Register 3-2

Condition Codes 5-3

8-Bit Load Group Instructions 5-4

16-Bit Load and Exchange Group Instructions 5-5

Block Transfer and Search Group 5-5

8-Bit Arithmetic and Logic Group 5-6

16-Bit Arithmetic Operation Instructions... 5-7

[Bit Manipulation, Rotate and Shift Group 5-8](#bookmark146)

Program Control Group Instructions 5-8

Input/Output Instruction Group Instructions «.5-9

CPU Control Group 5-10

Extended Instructions 5-10

Encoding of 8-Bit Registers in Instruction Opcodes 5-11

Grouping of Maskable Interrupt Requests 6-1

Interrupt Modes 6-4

Trap Types 6-7

Interrupt Acknowledge Encoding for Z80 Bus Parts 6-7

Interrupt/Trap Vector Table Format 6-10

Page Descriptor Register Addresses 7-5

MMU Invalidation Port 7-6

1/0 Port Addresses for MMU Control Registers 7-6

CPU Accesses to On-Chip Memory as Cache.....' 8-2

On-Chip DMA Accesses (Both Flowthrough and Flyby) Effect on On-Chip Memory as Cache 8-3

DMA/CPU Accesses to On-Chip Memory as Fixed Memory Location...8-4

Encoding, IPA Field in C/T Configuration Register 9-5

1/0 Addresses of Counter/Timer Registers 9-7

Configuration and Command/Status Registers for Linked Counter/Timers ...9-8

Encoding of DAD & SAD Fields in DMA Transaction Descriptor Register 9-13

Encoding of Type Field in Transaction Descriptor Register 9-14 /

Encoding of BRP Field in Transaction Descriptor Register 9-14

Encoding of ST Field in Transaction Descriptor Register 9-14

1/0 Addresses of DMA Registers 9-15

CR Field of UART Configuration Register 9-19

BC Field of UART. Control Register .9-19

1/0 Addresses of UART Registers 9-20

Reset Value of UART and DMA Registers When Bootstrap Mode Is Selected ....'. 9-21

**Table of Contents** (Continued)

10-1. Bus Transactions Involved in Fetch of Extended Instruction Template 10-8

10-2. Sequence of Transactions for Data Transfers Between an EPl) and Memory 10-9

11-1. Effect of a Reset on Z280 CPU & MMU Registers 11-2

11-2. Effect of a Reset on Z280 On-Chip Peripheral Registers 11-3

13-1. ST Status Line Decode 13-4

B —1. Format 1 Instruction Encodings B-2

B-2. Format 2 Instruction Encodings B-2

B-3. Format 3 Instruction Encodings B-2

B-4. Format 4 Instruction Encodings B-2

E-1. Instruction Execution Times E-2 ,

E-2. Extended Instruction Execution Times E-11

E-3. Interrupt, Trap, and Special Condition Execution Times E-12

E-4. Instruction Fetch and Decode Timing E-13

E-5. Data Read Timing E-14

E-6. Data Write Timing E-14 '

E-7. I/O Read and Write Timing E —15

E-8. EPU Read and Write Timing E-15

E-9. Interrupt Acknowledge Timing E-15

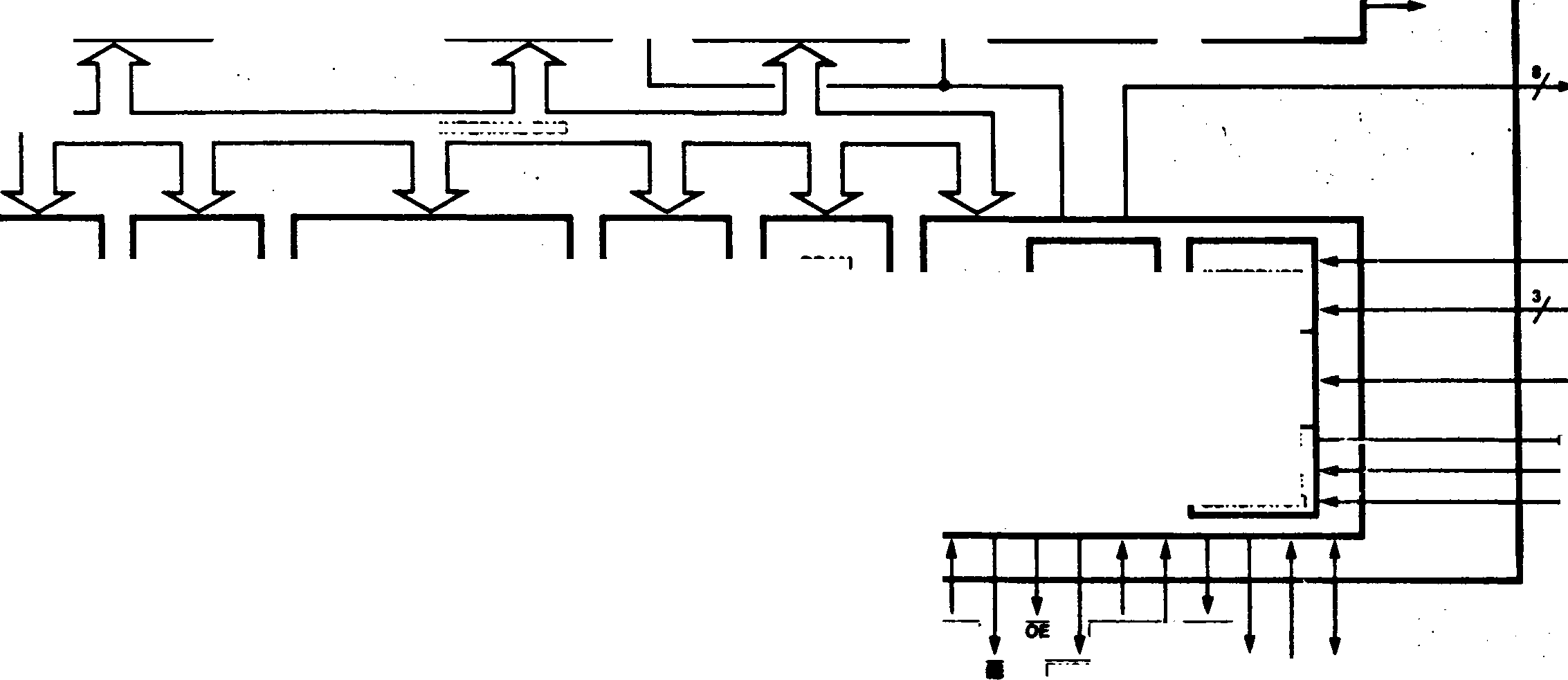
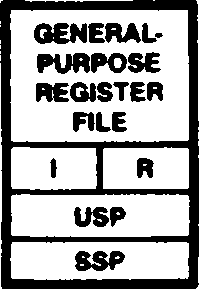
E-10. Miscellaneous Transaction Timing E-16

F-1. Z8400 Peripheral Family F-1

F-2. Z8000/Z8500 Peripheral Family....... .....F-1

**Chapter!**

**Z280 Architectural Overview**



XTALI

XTALO

3-STAGE PIPELINE

Z\*0 COMPATIBLE

EXECUTION UNIT

15-BIT ALU

PROGRAM COUNTER

MEMORY  
MANAGEMENT  
UNIT

INSTRUCTION/  
DATA CACHE  
OR MEMORY

ADDRESS  
TAGS

256  
BYTES

INSTRUCTION  
DECODER

EXECUTION  
SEQUENCER

INTERNAL CONTROL SIGNALS

LRU

INTERNAL BUS

INT

UART

OPT

24-BIT SOURCE

24-BIT DESTINATION

► CLK

15-BIT COUNTER

WAIT

CONTROL

PAUSE

3

I^SET

BUSREQ

GREQ

BUSACK GACK А1ГАгз

CT IN

DMASTB

RxD

1X0

ADo-AOt

BURST MEMORY CONTROL

DRAM 10-BIT REFRESH ADDRESS GENERATOR

FOUR 15-BIT DMA CHANNELS

ZSOBUS (•-BIT) OR Z-BUS (1S-BIT)

THREE 15-BIT COUNTER/ TIMERS

CLOCK OSCILLATOR

BUS SCALE ANO WAIT STATE GENERATOR

EXTERNAL BUS INTERFACE

'RO/OS •HALT/B/W ‘WS/R/W •fiFSH/STo •tORQ/STj

•MI/ST2 •MREQ/ST3

RfiTl

ac£ad1s

• Signal definition depends on OPT.

♦ SOP shares W/ШТд

♦ 6XCR shares W/CTINO

: \* GREQ shares W/CT Юо.

INTERRUPT  
CONTROL

**Figure 1-1. Block Diagram**

* 1. INTRODUCTION

Микропроцессорный блок Z280\* (MPU) оснащен усовершенствованным 16-разрядным процессором, который совместим с объектным кодом процессора Z80®. Микропроцессорный блок Z280 включает в себя управление памятью, периферийные устройства, логику обновления памяти, кэш-память, генераторы состояний ожидания и тактовый генератор на той же интегральной схеме, что и центральный процессор. Встроенные периферийные устройства включают в себя 4 канала DMA (прямого доступа к памяти), 3 счетчика/таймера и UART (универсальный асинхронный приемник/передатчик)• Структурная схема MPU Z280 показана на рисунке 1-1. В этой главе представлены некоторые особенности семейства MPU Z280, а подробные описания различных аспектов работы процессора приведены в последующих главах.

MPU Z280 имеет мультиплексированную шину адреса/данных для связи с внешней памятью и периферийными устройствами. Z280 поддерживает две различные структуры шин: 8-разрядную шину данных, использующую сигналы управления шиной Z80, и 16-разрядную шину данных, использующую сигналы управления шиной Z-BUS®. Периферийные устройства семейств Zilog Z80 и Z8500 легко подключаются к шине Z80; Периферийные устройства семейства Zilog Z8000® легко подключаются к Z-шине.

1.2 АРХИТЕКТУРНЫЕ ОСОБЕННОСТИ MPU

Центральный процессор Z280 MPU является двоично-совместимым расширением архитектуры процессора Z80. Высокая пропускная способность процессора Z280 достигается за счет высокой тактовой частоты, конвейерной обработки команд и использования встроенной кэш-памяти. Тактовая частота внутреннего процессора может быть уменьшена, чтобы обеспечить более медленную синхронизацию транзакций по шине. На кристалле предусмотрен программируемый механизм обновления для динамических ОЗУ и генератор тактовых импульсов.

* + 1. Systee and User Modes

Для облегчения проектирования операционной системы предусмотрены два режима работы процессора, системный и пользовательский. В системном режиме могут выполняться все инструкции и доступен доступ ко всем регистрам процессора. Этот режим предназначен для использования программами, выполняющими функции операционной системы. В пользовательском режиме определенные инструкции, влияющие на состояние машины, не могут быть выполнены, и управляющие регистры в центральном процессоре недоступны. В общем, пользовательский режим предназначен для использования прикладными программами. Такое разделение ресурсов ЦП способствует целостности системы, поскольку программы, выполняющиеся в пользовательском режиме, не могут получить доступ к тем аспектам ЦП, которые имеют дело с зависящими от времени событиями или событиями системного интерфейса.

Структура регистров была расширена, чтобы включать отдельные регистры указателей стека, один для стека системного режима и один для стека пользовательского режима. Стек системного режима используется для сохранения состояния программы при возникновении состояния прерывания или ловушки, тем самым гарантируя, что пользовательский стек свободен от системной информации. Изоляция системного стека от программ пользовательского режима дополнительно повышает целостность системы. • ■ • \* \* 1 \*

* + 1. Address Spaces

Адресные пространства в процессоре Z280 включают пространство регистров процессора, пространство регистров управления процессором, адресное пространство памяти и адресное пространство ввода-вывода. Файл регистров процессора идентичен набору регистров Z80, за исключением отдельных указателей стека системного и пользовательского режимов. Регистр A действует как 8-разрядный накопитель; регистр HL является 16-разрядным накопителем. Они дополняются четырьмя другими 8-разрядными регистрами (V, C, 0, E) и двумя другими 16-разрядными регистрами (IX, IY); 8-разрядные регистры могут быть сопряжены для 16-разрядной работы, и каждый 16-разрядный регистр может рассматриваться как два 8-разрядных регистры. Регистр флагов (F) содержит информацию о результате последней операции. Регистры A, F, V, C, D, E, H и L реплицируются во вспомогательном банке регистров. Этими вспомогательными регистрами можно обмениваться с основным банком регистров для быстрого переключения контекста.

Несколько регистров управления процессором определяют работу MPU Z280. Например, содержимое регистров управления определяет режим работы процессора, какие прерывания включены, и время выполнения транзакций по шине. Регистры управления доступны только при работе в системном режиме.

Адресное пространство логической памяти процессора Z280 такое же, как и у процессора Z80: 16-разрядные адреса используются для обращения к памяти объемом до 64 тыс. байт. Однако встроенный модуль управления памятью (MMU) расширяет 16-разрядный логический адрес памяти до 24-разрядного адреса физической памяти. Два отдельных логических адресных пространства, одно для системного режима и одно для пользовательского режима, поддерживаются центральным процессором и MMU. Опционально MMU может быть запрограммирован на различение выборки команд и доступа к данным; таким образом, центральный процессор Z280 может иметь до четырех адресных пространств памяти: программа системного режима,

данные системного режима, программа пользовательского режима и данные пользовательского режима. Логическое адресное пространство разделено на страницы для облегчения контролируемого обмена программой или данными между отдельными процессами. .

Архитектура процессора Z280 также проводит различие между адресными пространствами памяти и ввода-вывода и, следовательно, требует специальных инструкций ввода-вывода. Адреса ввода-вывода в процессоре Z280 имеют длину 24 бита, причем верхние 8 бит предоставляются регистром страницы ввода-вывода в процессоре. • , • « ■ ’ • • . .

* + 1. Data Types '

Процессор Z280 поддерживает многие типы данных. архитектура. Основным типом данных является 8-разрядный байт, который также является основным адресуемым элементом памяти. Архитектура также поддерживает операции с битами, цифрами BCD, 2-байтовыми словами и байтовыми строками.

* + 1. Addressing Modes

Режим адресации операндов - это метод, с помощью которого определяется местоположение операндов данных. Центральный процессор Z280 поддерживает девять режимов адресации, включая пять режимов, доступных на центральном процессоре Z80. Режимами адресации центрального процессора Z280 являются;

• Регистр • Немедленный.

• Косвенный регистр

• Прямой адрес

• Индексированный (с 16-разрядным смещением)

• Короткий индекс (с 8-разрядным смещением)

• Относительный счетчик программ (PC)

• Относительный указатель стека (SP)

• Базовый индекс + . 4 . . z .. . \*

Все режимы адресации доступны для 8-разрядных команд загрузки, арифметических и логических команд; 8-разрядные команды сдвига, поворота и манипулирования битами ограничены режимами адресации регистра, косвенного регистра и короткого индекса. 16-разрядные нагрузки на регистры адресации поддерживают все режимы адресации, кроме короткого индекса, в то время как другие 16-разрядные операции ограничены режимами регистра, непосредственного, косвенного регистра, индекса, прямого адреса и относительной адресации ПК.

\*\* \* '

* + 1. Instruction Set

Набор команд процессора Z280 является расширением набора команд Z80; улучшения включают поддержку дополнительных режимов адресации для инструкций Z80, а также добавление новых инструкций. Набор команд процессора Z280 обеспечивает полный набор 8- и 16-разрядных арифметических операций, включая умножение и деление со знаком и без знака. Дополнительные 8-разрядные вычислительные инструкции поддерживают логические и десятичные операции. Инструкции по обработке битов, повороту и сдвигу дополняют возможности процессора Z280 по обработке данных. Инструкции перехода, вызова и возврата имеют как условную, так и безусловную версии; для команд перехода и вызова предусмотрена относительная адресация для поддержки программ, не зависящих от местоположения. Инструкции перемещения блоков, поиска и ввода-вывода предоставляют мощные возможности перемещения данных. Кроме того, были включены специальные инструкции для облегчения многозадачности, настройки нескольких процессоров и типичных функций языка высокого уровня и операционной системы.

* + 1. Exception Conditions

MPU Z280 поддерживает три типа исключений (условий, которые изменяют нормальный ход выполнения программы); прерывания, ловушки и сброс.

Прерывания - это асинхронные события, обычно запускаемые периферийными устройствами, требующими внимания. Структура прерываний MPU Z280 была значительно улучшена за счет увеличения количества строк запроса на прерывание и добавления эффективного

• средства для обработки вложенных прерываний. Существует четыре режима обработки прерываний:

• совместимый с 8080, в котором прерывающее устройство выдает первую команду процедуры прерывания.

• Выделенные прерывания, при которых центральный процессор переходит на выделенный адрес при возникновении прерывания.

• Режим векторизованного прерывания, при котором периферийное устройство, выполняющее прерывание, предоставляет вектор в таблицу адресов перехода.

• Улучшенный режим векторизованного прерывания, в котором центральный процессор обрабатывает ловушки и множественные источники прерываний, сохраняя управляющую информацию, а также программный счетчик при возникновении прерывания. ;

Первые три режима совместимы с режимами прерываний процессора Z80; четвертый режим обеспечивает большую гибкость благодаря поддержке вложенных прерываний и сложной схеме векторизации. .

Ловушки - это синхронные события, которые запускают специальный отклик процессора при возникновении определенных условий во время выполнения команды. Процессор Z280 CPU 4 поддерживает сложный набор ловушек, включая исключение разделения, системный вызов, привилегированную инструкцию, расширенную инструкцию, одноступенчатую, точку останова при остановке, нарушение доступа к памяти и предупреждения о переполнении системного стека.

Аппаратный сброс происходит, когда активирована строка RESET, и отменяет все остальные условия. Сброс приводит к инициализации определенных регистров управления процессором. \*•

* + 1. Memory Management , ,

Memory management consists primarily of dynamic relocation, protection, and sharing of memory.

Proper memory management can provide a logical structure to the memory space that is independent of the actual physical location of data, protect the user from inadvertent mistakes (such as trying to execute data), prevent unauthorized accesses to memory, and protect the operating system from disruption by users. .

The 16-bit addresses manipulated by the pro­grammer, used by instructions, and output by the CPU are called logical addresses. The on-chip Memory Management Unit (MMU) transforms the logical addresses into the corresponding 24-bit physical addresses required for accessing memory. This address transformation process is called relocation, and makes user software independent of physical memory. Thus, the user is freed from specifying where information is actually located in physical memory.

Status information generated by the CPU allows the MMU to monitor the intended use of each memory access. Illegal types of accesses, such as writes to read-only memory, can be suppressed; thus, areas of memory can be protected from unintended or unwanted modes of use. Also, the MMU records which memory areas have been modified and can inhibit copies of data from being retained in the on-chip cache. « ■

When a memory access violation is detected by the MMU, a trap condition is generated in the CPU and execution of the current instruction is auto­matically aborted. This mechanism facilitates the easy implementation of virtual memory systems based on the Z280 MPU. ’ • • • \* '\* ' ' . \* • •

* + 1. Cache Memory , .

Cache memories are small high-speed buffers situated between the processor and main memory. For each memory access, control logic checks to see if the data at that memory location is currently stored in the cache. If so, the access is made to the high-speed cache; if not, the access is made to main memory, and the cache itself might be updated. Thus, use of a cache leads to increased performance with fewer memory transactions on the system bus. .

The Z280 MPU includes on-chip memory that can be used as a cache for programs, data, or both. Cache operations, including updating, are performed automatically and are completely trans­parent to the user. Optionally, this on-chip memory can be dedicated to a set of memory locations that are specified under program control, instead of being used as a cache.

* + 1. Refresh •

MPU Z280 имеет внутренний механизм обновления динамической памяти. Этот механизм может быть включен или отключен под управлением программы. Если он включен, операции обновления памяти выполняются периодически со скоростью, определяемой содержимым регистра частоты обновления. Для каждой операции обновления генерируется 10-разрядный адрес обновления.

«

* + 1. On-Chip Peripherals ♦ ,

В MPU Z280 встроено несколько программируемых периферийных устройств: четыре канала DMA, три 16-разрядных счетчика/таймера и UART. Опционально один из каналов DMA может использоваться с UART в качестве загрузчика начальной загрузки для памяти Z280 MPU после сброса.

**1.2.11 Multiprocessor Mode .**

Специальный режим работы позволяет Z280 MPU работать в средах, имеющих глобальную шину, при этом Z280 MPU не является ведущим устройством глобальной шины. Набор адресов памяти (определенный под программным управлением) выделяется локальной шине, которая управляется MPU Z280, а другой набор адресов используется для глобальной шины. MPU Z280 требуется для отправки запроса на шину и получения подтверждения, прежде чем осуществлять доступ к памяти по адресу на глобальной шине. Этот режим работы облегчает использование MPU Z280 в многопроцессорных конфигурациях. Например, MPU Z280 может использоваться в качестве процессора ввода-вывода в системах на базе Z80000, Z8000- или Z280.

1.2.12 Extended Instruction Facility

• . • • ♦

The Z280 MPU architecture has a mechanism for extending the basic instruction set through the - use of external devices called Extended Processing Units (EPUs). Special opcodes have been set aside to implement this feature. When the Z280 MPU encounters an instruction with one of these opcodes, it performs any indicated address calcu­lations and data transfers; otherwise, it treats the "extended instruction" as if it were executed by the EPU. , •

If an EPU is not present, the Z280 MPU can be programmed to trap when an extended instruction is encountered so that system software can emulate the EPU’s activity. •

* 1. BENEFITS OF THE ARCHITECTURE

The features of the Z280 MPU architecture provide several significant benefits, including increased program throughput, increased integration of system functions, support for operating systems, and improvements in compiler efficiency and code density. .

* + 1. High Throughput ,

> ■ \*

Very high throughput rates can be achieved with the Z280 MPU, due to the cache memory, instruction pipelining, and high clock rates achievable with this processor. The CPU clock rate can be scaled down to provide the bus clock rate, allowing the designer to use slower, less-expensive memory and I/O devices. Use of the on-chip cache memory further increases throughput by minimizing the number of accesses to the slower, off-chip memory devices. The high code density achievable with the Z280 CPU's expanded instruction set also contributes to program throughput, since fewer instructions are needed to accomplish a given task.

* + 1. Integration of System Functions / ‘ •

Besides a powerful CPU, the Z280 MPU includes many on-chip devices that previously had to be implemented in logic external to the micro­processor chip. These devices include a clock oscillator, memory refresh logic, wait state generators, the MMU, cache memory, DMA channels, counter/timers, and a UART. Integration of all these functions onto a single chip results in a reduced parts count in a system design, accom­panied by a resulting reduction in design and debug time, power requirements, and printed circuit board space. This increased level of integration also contributes to system throughput, since the on-chip devices can be accessed quickly without the need of an external bus transaction.

* + 1. Operating System Support , \* t • ‘

Several of the Z280 MPU's architectural features facilitate the implementation of multitasking operating systems for Z280-based systems.

The inclusion of user and system operating modes improves operating system organization. User-mode programs are automatically inhibited from per­forming operating-system type functions. System­mode memory can be separated from user-mode memory and separate stacks can be maintained for system­mode and user-mode operations. The System Call instruction and the trap mechanism provide a controlled means of accessing operating system functions during user-mode execution. • #

The interrupt- and trap-handling mechanisms are well suited for operating system implementations. Several levels of interrupts are provided, allowing for separate control of various peripher­al devices (both on and off the chip). A new interrupt mode is provided, wherein status infor­mation about the currently executing task is saved on the stack and new program status information for the service routine is automatically loaded from a special memory area. Traps result in the same type of program status saving. In both cases, status is always saved on the system stack, leaving the user stack undisturbed. .

. \ \* ■

Allocation of resources within the operating system can be accomplished using a special Test and Set instruction. Other instructions, such as the Purge Cache instruction, are provided to aid in task switching and other operating system chores. •

The on-chip MMU supports a multitasking environ­ment by providing both a means of quickly allocating physical memory to tasks as they are executed on the system and protection mechanisms to enforce proper memory usage. 4 ***. • t • • : ' .***

' ’ - • ‘ '

* + 1. Code Density .

. Я ' • ***4 ' • • e \*\* \****

Code density affects both processor speed and memory utilization. Code compaction s a vies memory space and improves processor speed by reducing the number of instructions that must be fetched and decoded. The largest reduction in program size results from the powerful instruction set, where instructions such as Multiply and Divide help substantially reduce the number of instructions required to complete a task. . ' '

The efficiency of the instruction set is enhanced by the addition of new addressing modes. For example, all nine addressing modes are available for all the 8-bit load, arithmetic, and logical instructions. •

’ • I • \*

* + 1. Compiler Efficiency *. 4 ' •*

For microprocessor users, the transition from assembly language to high-level languages allows greater freedom from architectural dependency and improves ease of programming. For the Z280 MPUs, high-level language support is provided through the inclusion of features designed to minimize typical compilation and code-generation problems.

Among these features is the variety and the power of the Z280 instruction set, allowing the Z280 CPU ' to easily handle a large amount and variety of data types. The Z280 CPU's ability to manipulate many different data types aids in compiler efficiency; since data structures are high-level constructs frequently used in programming, processing performance is enhanced by providing efficient mechanisms for manipulating them.

Examples of commonly used data structures include arrays, strings, and stacks. Arrays are supported in the Z280 CPU by the Indirect Register, Index, and Base Index addressing modes. Strings are supported by those same addressing modes and the Block Move and Compare instructions; since compilers and assemblers often must manipulate character strings, the Block Move and Block Compare instructions can result in dramatic speed improvements over software simulations of those tasks. Nuneric strings of BCD data can be manipulated using the Decimal Adjust and Rotate Digit instructions. Stacks are supported by the Push and Pop instructions and the Stack Pointer Relative, Index, and Base Index addressing modes; the Stack Pointer Relative addressing mode is

especially useful for accessing parameters and local variables stored on the stack. '

* 1. SUMMARY

The Z280 MPU is a high-performance 16-bit micro­processor, available with 8- and 16-bit external bus interfaces. Code-compatible with the Z80 CPU, the Z280 MPU architecture has been expanded to include features such as multiple memory address spaces, efficient handling of nested interrupts, system and user operating modes, and support for multiprocessor configurations. Additional functions such as memory management, clock generation, wait state generation, and cache memory are included on-chip, as well as a number of peripheral devices. The benefits of this architecture—including high throughput rates, a high level of system integration, operating system support, code density, and compiler efficiency— greatly enhance the power and versatility of the Z280 MPU. Thus, the Z280 MPU provides both a growth path for existing Z80-based designs and a high-performance processor for future applications.

Chapter 2.

Address Spaces

| **A ACCUMULATOR** | **F FLAG REGISTER** | **A' ACCUMULATOR**  **• 1** | **F' FLAG REGISTER** |
| --- | --- | --- | --- |
| **В GENERAL PURPOSE** | **C GENERAL PURPOSE** | **B’ GENERAL PURPOSE** | **C' GENERAL PURPOSE** |
| **D GENERAL PURPOSE** | **E GENERAL PURPOSE** | **D\* GENERAL PURPOSE** | **E' GENERAL PURPOSE** |
| **H GENERAL PURPOSE** | **L GENERAL PURPOSE t .** | **Hr GENERAL PURPOSE** | **L’ GENERAL PURPOSE** |

AUXILIARY FILE

8 BITS

NOTE: A is the 8-bit accumulator.

HL is the 16-bit accumulator.

* 1. INTRODUCTION

The Z280 MPU supports four address spaces corre­sponding to the different types of locations that can be addressed, the method by which the logical addresses are formed, and the translation mecha­nisms used to map the logical address into physical locations. These four address spaces ares .

. • • • •

* **CPU register space.** This consists of the addresses of all registers in the CPU register file.
* **CPU control register space.** This consists of the addresses of all registers in the CPU control register file.
* **Meaory address space.** This consists of the addresses of all locations in the main memory. ■ t

. • \* 4 ■

* **I/O address space.** This consists of the addresses of all I/O ports through which peripheral devices are accessed, including on-chip peripherals and MMU registers. •

***. I***

PRIMARY FILE

* 1. CPU REGISTER SPACE . » » •

• •

The Z280 CPU register file is illustrated in Figure 2-1. The primary register file, consisting of the A, F, В, C, D, E, H, and L registers, is augmented by an auxiliary file containing duplicates of those registers. Only one set (either the primary or auxiliary file) can be used at any one time. Special exchange instructions are provided for switching between the primary and auxiliary registers.

The CPU register file is divided into five groups of registers (an apostrophe indicates a register in the auxiliary file): • '

’ \* \* \*’ \* ’

* Flag and accunulator registers (F, A, F’, A’)
* Byte/word registers (В, C, D, E, H, L, В’, C’, D', E', H’, L’)
* Index registers (IX, IY) .
* Stack Pointers (SSP, USP) /.
* Program Counter, Interrupt register, and . Refresh register (PC, I, R) ;

■ ./ •. •• • '

| **1 INTERRUPT VECTOR** | **A '** |
| --- | --- |
| **IX INDEX REGISTER |** | |
| **1 IY INDEX REGISTER 1** | |
| **PC PROGRAM COUNTER** | |
| **SP STACK POINTER**  **USER (USP)** | |

SYSTEM (SSP)

16 BITS

**Figure 2\*1. Register File Organization**

Register addresses are either specified explicitly in the instruction or are implied by the semantics of the instruction.

The flag registers (F, F') contain eight status flags. Four can be individually used for control of program branching, two are used to support decimal arithmetic, and two are reserved (see section 5-2). The accumulator (A) is the implied destination (i.e., where the result is stored) for the 8-bit arithmetic and logical instructions. Two sets of flag and accumulator registers exist in the Z280 CPU, with only one set accessible as the flag register and the accumulator at any one time. An exchange instruction allows switching to the alternate flag register and accumulator. ***J***

The byte/word registers can be accessed either as 8-bit byte registers or 16-bit word registers. Bits within these registers can also be accessed individually. For 16-bit accesses, the registers are paired В with C, D with E, and H with L. Two sets of byte/word registers exist in the Z280 CPU, although only one set is used as the current byte/word registers; the other set is accessible as the alternate group of byte/word registers via an exchange instruction.

The index registers IX and IY can be accessed as 16-bit registers or their upper and lower bytes (IXH, IXL, IYH, and IYL) can be individually accessed.

The Z280 CPU has two hardware Stack Pointers, one dedicated to system mode operation and one to user mode operation. The System Stack Pointer (SSP) is used for saving information when an interrupt or trap occurs and for supporting subroutine calls and returns in system mode. The User Stack Pointer (USP) is used for supporting subroutine calls and returns in user mode.

The Program Counter is used to sequence through instructions in the currently executing program and for generating relative addresses. The Inter­rupt register is used in interrupt mode 2 to generate a 16-bit logical address from an 8-bit vector returned by a peripheral during an inter­rupt acknowledge. The Refresh register is used by the Z80 CPU to indicate the current refresh address, but does not perform this function in the Z280 CPU; instead, it is another 8-bit register available for the programmer.

The explicit or implicit register specified by an instruction is mapped into the CPU register file based on the state of three control bits. One of the three control bits is used to map the flag and accumulator registers, selecting either F, A or F’, A’ whenever the instruction specifies the flag register or the accumulator. Another control bit is used to map the byte/word registers, selecting the В, C, D, E, H, L registers or the В’, C’, D’, E’, H’, L’ registers. These two control bits are changed by the Exchange Flag and Accumulator and the Exchange Byte/Word Registers instructions, respectively. At any time the program can sense the state of these control bits by special jump instructions. The third control bit, the User/System control bit in the Master Status register, specifies whether the System Stack Pointer register or the User Stack Pointer register is selected whenever an instruction specifies the Stack Pointer register. In addition, the User Stack Pointer register also has an address in the CPU control register space via a special Load Control instruction.

* 1. CPU CONTROL REGISTER SPACE

The Z280 CPU status and control registers govern the operation of the CPU. They are accessible only by the privileged Load Control (LDCTL) instruction. : ■ t

Control register addresses are specified by the contents of the C register. No translation is performed in mapping this 8-bit logical address into the control register file location.

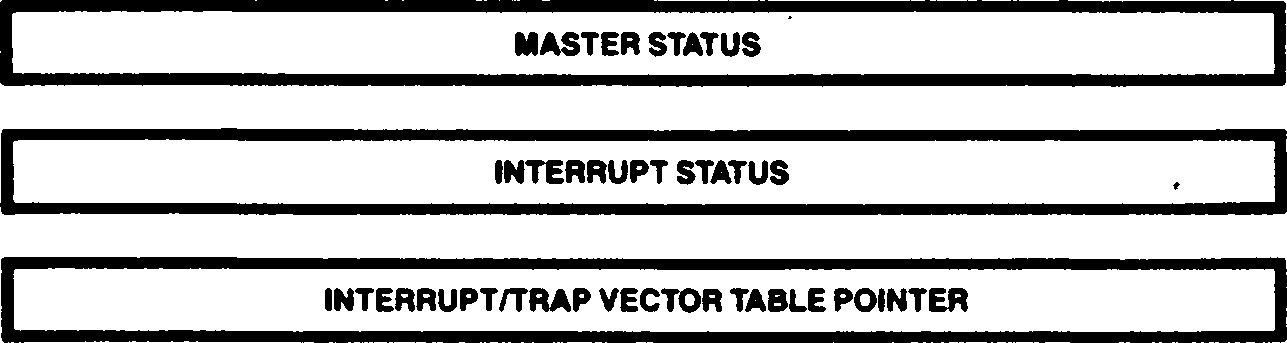
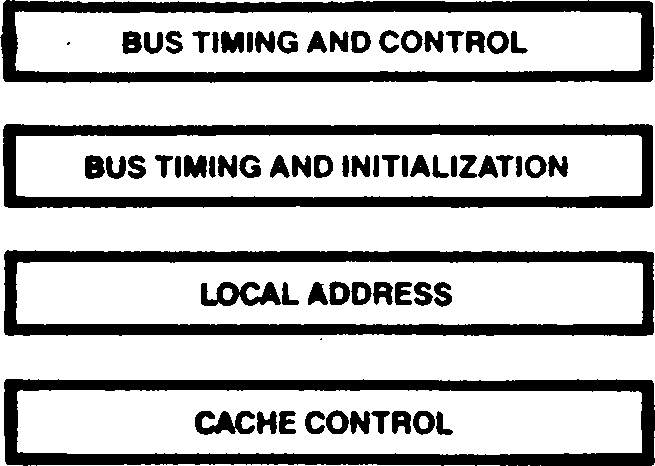
The Z280 CPU control registers are the Bus Timing and Initialization register, the Bus Timing and Control register, the Master Status register, the Interrupt/Trap Vector Table Pointer, the I/O Page register, the System Stack Limit register, the Trap Control register, the Interrupt Status register, the Cache Control register, and the Local Address register (Figure 2-2). The CPU control registers are described in detail in Chapter 3.

CONTROL  
REGISTERS \

SYSTEM STATUS 7 REGISTERS \

I/O PAGE

TRAP CONTROL



SYSTEM STACK LIMIT

Figure 2-2. CPU Control Registers

* 1. MEMORY ADDRESS SPACES

Two memory address spaces, one for system and one for user mode operation, are supported by the Z280 MPU. They are selected by the User/System mode control bit in the Master Status register, which governs the selection of page descriptor registers in the MMU during address translation.

Each address space can be viewed as a string of 64K bytes numbered consecutively in ascending order. The 8-bit byte is the basic addressable element in the Z280 MPU memory address spaces. However, there are other addressable data ele­ments: bits, 2-byte words, byte strings, and multiple-byte EPU operands.

• • 4 ’

The size of the data element being addressed depends on the instruction being executed. A bit can be addressed by specifying a byte and a bit within that byte. Bits are numbered from right to left, with the least significant bit being bit 0, as illustrated in Figure 2-3. .

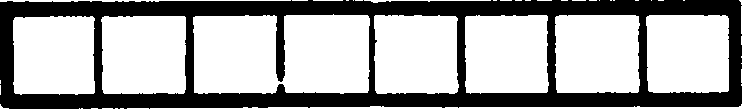


Figure 2-3. Numbering of Bits within a Byte

The address of a multiple-byte entity is the same as the address of the byte with the lowest memory address within the entity. Multiple-byte entities can be stored beginning with either even or odd memory addresses. A word (2-byte entity) is aligned if its address is even; otherwise it is unaligned. Multiple bus transactions, which may be required to access multiple-byte entities, can be minimized if alignment is maintained.

The formats of multiple byte data types in memory are given in Figure 2-4. . .

Note that when a word is stored in memory, the least significant byte precedes the most significant byte of the word, as in the Z80 CPU architecture. • • • . • • '• . \* ’ \* • • . \* . ‘ ’

The 16-bit logical addresses generated by a program can be translated into 24-bit physical addresses by the on-chip MMU. When the translation mechanism is disabled, the 24-bit physical address consists of the logical address for bits Aq-A-|5 and zeros for A-jg-Azp

60-bit floating-point (EPU instruction only) at address n:

16-bit word at address n:

sign,E10-4

E3-0, F51-48

F47-40

F39-32

F31-24

F23-16

F15-8

F7-0

<--1 byte -->

address n

address n +1 address n + 2 address n + 3 address n + 4 address n + 5 address n + 6

address n + 7

least significant byte most significant byte < 1 byte >

address n

address n +1

32-bit integer (EPU instruction only) at address n:

80-bit floating-point (EPU instructions only) at address n:

B31-24 (most significant byte)

B23-16

B15-8

B7-0 (least significant byte) < 1 byte >

address n

address n +1 address n + 2

address n + 3

sign,E14-8

E7-0

F63-56

F55-48

F47-40

F39-32

F31-24

F23-16

F15-8 -

F7-0

address n

address n +1 address n + 2 address n + 3 address n + 4 address n+5 address n+6 address n + 7 address n + 8

address n + 9

64-bit integer (EPU instruction only) at address n:

B63-56 (most significant byte)

B55-48 .

B47-40 .

B39-32

B31-24 .

B23-16

B15-8 '

B7-0 (least significant byte) < 1 byte >

address n

address n +1 address n+ 2

address n + 3 address n + 4

address n + 5 address n + 6

address n + 7

BCD digit strings (EPU instruction only) at address n: (up to 10 bytes in length; the illustration is for the maximum length string)

32-bit floating-point (EPU instruction only) at address n:

sign,D18 D17.D16 D15.D14

D13.D12 D11.D10 D9.D8 D7.D6

D5.D4 D3,D2 D1.D0

address n

address n + 1

address n + 2 address n+3

address n+4 address n + 5 address n+6 address n + 7 address n + 8

address n + 9

sign,E7-1 E0.F22-16 F15-8

address n

address n +1 address n + 2

address n + 3

**Figure 2-4. Formats of Multiple-Byte Data Elements in Memory**

2.5 I/O ADDRESS SPACE *I .•*

I/O addresses are generated only by I/O instructions. The 8-bit logical port address specified in the instruction appears on ADg-ADy; this is concatenated with the contents of the A register on lines Ag-А^з for Direct addressing, mode, or by the contents of the В register for Indirect Register addressing mode or block I/O instructions. The contents of the I/O Page register are appended to this address on lines A16“A23\* Thus, the 24-bit I/O port address consists of the 8-bit address specified in the instruction, the contents of the A or В register, and the contents of the I/O Page register.

An I/O read or write is always one transaction, regardless of the bus size and the type of I/O instruction. On-chip peripherals with word registers are always accessed with word instructions, regardless of the size of the external bus. . '

Chapters.

CPU Control Registers

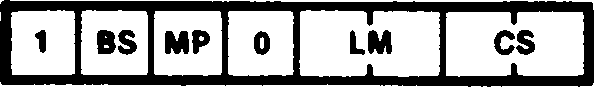


Figure 3-1. Bus Timing and Initialization Register

Multiprocessor Configuration Enable (И\*) Bit.

This 1-bit field enables the multiprocessor mode of operation, wherein the Z280 MPU is connected to both a local and a global bus. Transactions to

* 1. INTRODUCTION

Several CPU control and status registers specify the operating mode of the Z280 MPU. There are two types of CPU control registers: system

configuration registers and system status regis­ters. The system configuration registers contain information about the physical configuration of the Z280-based system, such as bus timing infor­mation. Typically, the system configuration registers are loaded once during system initial­ization and are not altered during subsequent operations. The system status registers contain information that may change during system operation, such as the current I/O page. Access to the CPU control registers is restricted to system mode operation only, using the privileged Load Control (LDCTL) instruction. Resets ini­tialize the control registers so that a Z80 object program will execute successfully on the Z280 MPU. (Z80 programs do not affect these registers, since the Load Control instruction is not part of the Z80 CPU's instruction set.) Unused bits in these registers should always be loaded with zeros.

* 1. SYSTEM CONFIGURATION REGISTERS • # 1 •

There are four 8-bit system configuration regis­ters: the Bus Timing and Initialization register, the Bus Timing and Control register, the Local Address register, and the Cache Control register.

***. I***

3.2.1 Bus Timing and Initialization Register

The Bus Timing and Initialization register controls the scaling of the processor clock for bus timing, the duration of bus transactions to the lower half of physical memory, and the enabling of the multiprocessor and bootstrap modes. Figure 3-1 illustrates the bit fields in this register.

**Clock Scaling (CS) Field.** This 2-bit field governs the scaling of the CPU clock for generation of bus timing cycles. The state of the CS field determines the bus clock frequency for all bus transactions, as per Table 3-1. This field is initialized during a reset operation, as described below, and cannot be modified via software. . .

• • . •

**Table 3-1. CS Field of Bus Timing and Initialization Register**

**CS Field Bus Clock Frequency**

00 . Bus clock frequency equals 1fe CPU clock frequency (one bus clock cycle for every two CPU clock cycles)

01 Bus clock frequency equals CPU clock frequency (one bus clock cycle for every one CPU clock cycle)

1. Bus clock frequency equals V» CPU clock frequency

(one bus clock cycle for every four CPU clock . cycles) 1 ..

1. Reserved >

» X

**Low Memory Wait Insertion (LN) Field.** This 2-bit field specifies the number of automatic wait states to insert in memory transactions to the lower 8 megabytes of physical memory (that is, all memory locations where bit 23 of the physical address is a 0), as per Table 3-2. Additional wait states can still be added to any given memory transaction via control of the WAIT input.

**Table 3-2. LM Field of Bus Timing and Initialization Register**

**Number of Walt States for LM Field Lower 8M Bytes of Memory**

00 **0**

. 01 : **1**

1. . ’ 2
2. : ■ ■ **3**

addresses on the global bus require a special bus request and acknowledgement before the bus trans­action can occur. (See Chapter 10 for details concerning this mode of operation.) Setting this bit to 1 enables the multiprocessor mode, and clearing this bit to 0 disables this mode.

**Bootstrap Mode Enable (BS) Bit.** This 1-bit field enables the bootstrap mode of operation. If the bootstrap mode is selected during a reset oper­ation, memory is automatically initialized via the UART after the reset; the UART receiver and DMA channel 0 are used to transfer 256 bytes of data into the first 256 memory locations; execution then begins from memory location 0. (See Chapter 9 for further details.) Setting this bit to 1 enables the bootstrap mode and clearing this bit to 0 disables this mode. The BS bit can be set to 1 only during a reset operation, as described below. Writing to this bit via a software command has no effect. This bit is always a 1 when this register is read.

Bits 4 and 7 of the Bus Timing and Initialization register are reserved for special use by Zilog and should always be loaded with a zero when writing to this register. When this register is read, bits 4 and 7 may return a 1.

• • . ■ . ■.

The Bus Timing and Initialization register can be initialized with either of two methods during a reset operation. If the MPU's WAIT input is not asserted during reset, this register is auto­matically initialized to all zeros, thereby specifying a bus clock frequency of one-half the internal CPU clock, no automatic wait states during transactions to the lower 8M bytes of memory, and disabling of the multiprocessor and bootstrap modes. If the WAIT input is asserted during reset, the Bus Timing and Initialization register is set to the contents of the ADQ-AD7 bus lines, as read during the reset operation (see Chapter 12); this form of initialization is the only way to specify the bootstrap mode. Once the CS field has been loaded during reset, it cannot be modified via software; however, the LM and MP fields can be written using the LDCTL instruction.

3.2.2 Bus Timing and Control Register

The 8-bit Bus Timing and Control register deter­mines the timing of bus transactions to the upper 8M bytes of memory and to all I/O devices, and the timing of interrupt acknowledge transactions. Figure 3-2 indicates the format of this register.

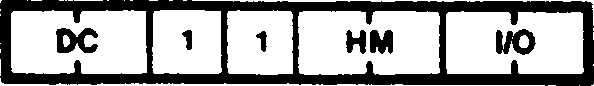


Figure 3-2. Bus Timing and Control Register

**I/O Wait Insertion (I/O) Field.** This 2-bit field specifies the number of automatic wait states (in addition to the one wait state always present during I/O transactions) to be inserted during each I/O read or write transaction, as per Table 3-3. The specified number of wait states is also added to the vector read portion of an interrupt acknowledge cycle. :

**Table 3-3. I/O Field of Bus Timing and Control Register**

**Number of Wait States I/O Field for I/O**

00 . **0**

• • 01 **i**

1. 2
2. **3 ;**

**High Memory Wait Insertion (HM) Field.** This 2-bit field specifies the number of automatic wait ' states to be inserted during memory transactions to the upper 8M bytes of physical memory (locations where address bit 23 of the physical address is a 1), as per Table 3-4.

***\* I .***

**Table 3-4. HM Reid of Bus Timing and Control Register**

**Number of Walt States for HM Field Upper 8M Bytes of Memory**

00 0

01 ’ ' • ■ ■ 1

1. .. 2
2. 3

**Daisy Chain Timing (DC).** This 2-bit field determines the number of automatic wait states to be inserted during interrupt acknowledge transactions while the interrupt acknowledge daisy chain is settling, as per Table 3-5. Normally, 2.5 bus clock cycles elapse between the assertion of Address Strobe and the assertion of Data Strobe during an interrupt acknowledge (for the Z-BUS) or between the assertion of ЙТ and the assertion of IORQ (for the Z80 Bus). The value of the DC field determines if any additional clocks are to be added between the Address Strobe and Data Strobe (or ЯТ and IORQ) assertions.

**Table 3-5. DC Field of Bus Timing and Control Register**

**Number of Walt States for**

**DC Field Interrupt Acknowledge**

00 0

01 1 .

10 2

. 11 3

The contents of the Bus Timing and Control register govern the number of automatic wait states to be inserted during various bus trans­actions. Additional wait states can be added to any bus transaction via control of the WAIT input.

The Bus Timing and Control register is set to ЗОН by a reset. Bits 4 and 5 should always be written with 0.

When this register is read, bits 4 and 5 may return a

**Match Enable bit (MEn):** If MEn is set to 1, then the corresponding physical address bit An is compared to base bit Bn to determine if the address requires the use of the global bus. If MEn is a zero, then any values for An and Bn produce a match, signifying a local bus access. If every MEn is cleared to 0, then all memory transactions are performed on the local bus.

\* •

The Local Address register is cleared to all zeros by a reset.

* + 1. Local Address Register



ME2o

В23

B22

B21

The 8-bit Local Address register is used while in multiprocessor mode to determine which memory addresses are accessed via the local bus and which memory addresses are accessed via the global bus. If the multiprocessor mode is disabled (that is, if there is a 0 in bit 5 of the Bus Timing and Initialization register), the contents of the Local Address register have no effect on MPU operation.

If multiprocessor mode is enabled, the MPU auto­matically uses the Local Address register during each memory access to determine if the global bus is required. The Local Address register consists of a 4-bit match field and a 4-bit base field that are compared to the upper four bits of the physical memory address during memory trans­actions. The 4-bit match field specifies which bits of the physical memory address are of interest; for those bit positions specified in the match field, if all the corresponding address bits match the Local Address register's base field bits, then the bus transaction can proceed on the local bus. If there is a mismatch in at least one of the specified bit positions, then the global bus is requested, and the transaction cannot proceed until the global bus acknowledge signal is asserted. (See Chapter 10 for further discussion of the Multiprocessor mode.) ***t***

The format of the Local Address register is illustrated in Figure 3-3.

**Figure 3-3. Local Address Register**

**Base bit (Bn):** For each MEn that is set to 1, the corresponding value of Bn must match the value of address bit An in order for the local bus to be used; otherwise, the transaction requires the use of the global bus.

* + 1. Cache Control Register

The 8-bit Cache Control register controls the operation of the on-chip memory. The contents of the Cache Control register determine if the on-chip memory is to be used as a cache or as fixed memory locations; if used as a cache, the cache can be enabled for instruction fetches only, for data fetches only, or for both instruction and data fetches. This register is also used to determine if burst-mode memory transactions are supported. (See Chapter 8 for further discussion of the on-chip memory and Chapter 13 for a description of the burst mode memory transaction.)

The Cache Control register contains five control bits, as described below. The format for this register is shown in Figure 3-4. . .

' r I

. I

7 ° .

■ |m/c| i 0 IlmbIhmbI 0 I 0 I 0 I '

**Figure 3-4. Cache Control Register**

**Neeory/Cache (М/С) Bit.** While this bit is set to 1, the on-chip memory is accessed as physical memory with fixed memory addresses; the user can programmably select the ranges of memory addresses for which the on-chip memory will respond. While this bit is cleared to 0, the on-chip memory is accessed associatively as a cache.

**Cache Instruction Disable (I) Bit.** While this bit and the М/С bit are cleared to 0, the on-chip memory is used as a cache during instruction fetches. While this bit is set to 1, instruction fetches do not use the cache. If the М/С bit is a 1, the state of this bit is ignored.

**Cache Data Disable (D) Bit.** While this bit and the М/С bit are cleared to 0, the on-chip memory is used as a cache during data fetches. While this bit is set to 1, data fetches do not use the cache. (The cache can be enabled for both

instruction and data fetches by clearing both the I and D bits.) If the М/С bit is a 1, the state of this bit is ignored. ■ ■ .

**Low Memory Burst Capability (LMB) Bit.** This 1-bit field specifies whether burst-mode memory transactions will occur during memory transactions to the lower 8M bytes of physical memory . (locations where address bit 23 of the physical address is a 0). Setting this bit to 1 enables burst-mode transactions; clearing this bit to 0 disables burst mode transactions.

**High Memory Burst Capability (HMB) Bit.** This . 1-bit field specifies whether burst-mode memory transactions will occur during memory transactions to the upper 8M bytes of physical memory (locations where address bit 23 of the physical address is a 1). Setting this bit to 1 enables burst-mode transactions; clearing this bit to 0 disables burst-mode transactions.

The Cache Control register is set to а 20ц (hexadecimal) by a reset, enabling the on-chip memory for use as a cache for instruction fetches only and disabling burst mode transactions. Bits 0, 1, and 2 of this register are not used.

• ’ • • • ■ ■ •' ' - ■■ . • • ' ' •

3.3 SYSTEM STATUS REGISTERS

There are six system status registers in the Z280 CPU: the Master Status register, Interrupt Status register, Interrupt/Trap Vector Table Pointer, I/O Page register, Trap Control register, and System Stack Limit register.

* + 1. Master Status Register ,

The 16-bit Master Status register (MSR) contains status information about the currently executing program. Typically, the MSR changes when a new programming task is dispatched; it changes automatically when an interrupt or trap occurs. For all traps and for interrupts processed using interrupt mode 3, the old value of the MSR is saved on the system stack and a new MSR is loaded along with the Program Counter to define the service routine. (See Chapter 6 for a detailed discussion of interrupt and trap processing).

The format of the Master Status register is shown in Figure 3-5. .

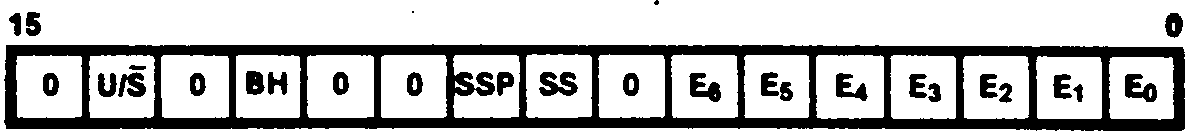


Figure 3\*5. Master Status Register

**User/System (U/5) Bit.** While this bit is cleared to 0, the Z280 MPU is in the system mode of operation; while set to 1, the MPU is in the user mode of operation. The current operating mode determines which Stack Pointer is in use and which instructions can be executed; privileged instructions can be executed only while in system mode. • . ,

**Breakpoint-on-Halt Enable (BH) Bit.** While this bit is set to 1, the CPU generates a breakpoint trap whenever a Halt instruction is encountered; while cleared to 0, the Halt instruction is executed normally.

**Single-Step Pending (SSP) Bit.** The CPU checks this bit prior to the start of an instruction execution and generates a Single-Step trap if this bit is set to 1. The Single-Step bit is automatically copied into this field at the completion of an instruction. This bit is automatically cleared when a Single-Step, Division Exception, Access Violation, Privileged Instruction, or Breakpoint-on-Halt trap is executed, so that the saved MSR has a 0 in this bit position. (For these traps, the PC address of the trapped instruction is saved for possible re-execution.)

»\* . ■ • Ш • • • » \* •

**Single-Step (SS) Bit.** This bit is the enable for the single-step operating mode. While this bit is set to 1, the CPU is in a single-step mode wherein a Single-Step trap is generated for each instruction; if cleared to 0, single-step mode is disabled. \* ’

**Interrupt Request Enable (En) Bit.** There are seven interrupt enable bits in the MSR, one for each type of maskable interrupt source. The Z280 MPU1 s interrupt sources, including both the external interrupt requests and the on-chip peripherals, are grouped into seven levels of interrupt requests. While bit En is set to 1, interrupt requests from sources at level n are accepted by the CPU; while En is cleared to 0, interrupt requests from sources at level n are not accepted. ,

The Master Status register is loaded with all zeros by a reset. Bits 7, 10, 11, 13, and 15 of the MSR always should be written with zeros.

•

3.3.2 Interrupt Status Register

• I. ■ ■ I

The 16-bit Interrupt Status register indicates which interrupt mode is in effect, which interrupt requests are pending, and which interrupt requests are to be vectored. Only the interrupt vector

enable bits are writeable; all other bits in this register are read-only status bits. The fields in the Interrupt Status register are shown in Figure 3-6. • t

The contents of the Interrupt/Trap Vector Table Pointer are unaffected by a reset and are undefined after power-up. When this register is read, bits 3,2,1 and 0 may return a 1.

15

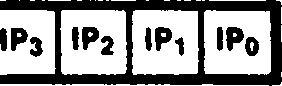
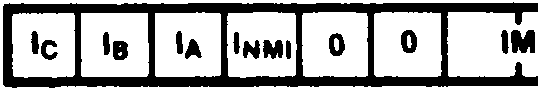


Figure 3-6. Interrupt Status Register *t* Interrupt Vector Enable (In) Bits. These four bits indicate which of the four external interrupt inputs are to be vectored. While In is set to 1, interrupts on the Interrupt n line are vectored when the CPU is in interrupt mode 3; while In is cleared to 0, that interrupt is not vectored. These bits are ignored when not in interrupt mode 3. . ....

**Interrupt Mode (IH) Field.** This 2-bit field indicates the current interrupt mode in effect, with a value n in this field\* denoting interrupt mode n. This field can be changed by executing the IM instruction.

**Interrupt Request Rending. (IPn) Bits.** When bit IPn is a 1, an interrupt request from a source at level n is pending.

On reset, the Interrupt Vector Enable bits are cleared to all zeros, interrupt mode 0 is in effect, and the Interrupt Pending bits reflect the state of the interrupt requests. Bits 7, 10, and 11 of this register are not used.

3.3.3 Interrupt/Trap Vector Table Pointer

***I***

The 16-bit Interrupt/Trap Vector Table Pointer contains the twelve most significant bits of the physical memory address of the start of the Interrupt/Trap Vector Table. The Interrupt/Trap Vector Table is a memory area that holds the values that are loaded into the Master Status . register and Program Counter during trap and interrupt processing under interrupt mode 3, as described in Chapter 6. The twelve low-order bits of the 24-bit physical address are assumed to be all zeros: thus, the Interrupt/Trap Vector Table must start on a 4K byte boundary in physical memory. The low-order four bits of the

Interrupt/Trap Vector Table Pointer must be all zeros (Figure 3-7).

15 о

I A23IA221 A211A201 Aig I Aib I Ai71 Aie IA151 Auj A\*i31 A12I 0 I 0 I 0 I 0 I

**Figure 3-7. Interrupt/Trap Vector Table Pointer**

3.3.4 I/O Page Register . ' .

\* V .

The 8-bit I/O Page register determines the upper eight bits of the 24-bit peripheral address output during execution of an I/O transaction (Figure 3-8). I/O pages FEH and FFH are reserved for on-chip peripheral addresses.

I \* ’ “ ‘ . • • •

**7 0**

**IA23IA22IA21 Am I Alt IA1SIA171А» I '**

**Figure 3-8. I/O Page Register**

The contents of the I/O Page register are cleared to all zeros by a reset. . . .

**3.3.5 Trap Control Register .**

. •

The 8-bit Trap Control register contains the enables for the maskable traps. Figure 3-9 illustrates the format of this register.

7 0

|o|o|o|o|o|||e|s| .

4 **Figure 3-9. Trap Control Register ,**

**Inhibit User I/O (I) Bit.** This bit determines whether or not I/O instructions are privileged instructions. While this bit is set to 1, all I/O instructions are treated as privileged

instructions, and an attempt to execute an I/O instruction while in user mode results in a Privileged Instruction trap. While this bit is cleared to 0, I/O instructions can be successfully executed in user mode. I/O instructions can always be executed in system mode, regardless of the state of this bit. ***t ‘ ' \* •***

**EPU Enable (E) Bit.** This bit indicates whether or not an Extended Processor Unit (EPU) is available in the system for execution of extended in­structions. If this bit is cleared to 0, indicating that no EPUs are present, the CPU generates an Extended Instruction trap whenever an extended instruction is encountered. If this bit is set to 1, the CPU performs whatever data transfers are indicated by the extended in­struction opcode, and assumes that the EPU is present to execute the instruction.

**System Stack Overflow Warning (S) Bit.** This is the enable bit for the System Stack Overflow Warning trap. While it is set to 1, Stack

Overflow Warning traps can occur during a stack access while in system mode, as determined by the contents of the Stack Limit register. While this bit is cleared to 0, Stack Overflow Warning traps are disabled. This bit is automatically cleared when a System Stack Overflow Warning trap is generated. • \* •

The Trap Control register is cleared to all zeros» by a reset, indicating that I/O instructions are not privileged, EPUs are not present in the system, and Stack Overflow Warning traps are disabled. Bits 3 through 7 of this register are not used.

3.3.6 System Stack Limit Register

The 16-bit System Stack Limit register determines when a System Stack Overflow Warning trap is to be generated. Pushes onto the system-mode stack cause the 12 most significant bits of the logical address of the System Stack Pointer to be compared to the 12 most significant bits of this register; a System Stack Overflow Warning trap is generated if they match. The low-order four bits of this register must be zeros (Figure 3-10). This register has no effect on MPU operation if the System Stack Overflow Warning enable bit in the Trap Control register is cleared to 0.

15

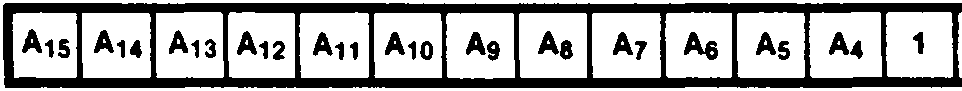


Figure 3-10. System Stack Limit Register

The contents of the System Stack Limit register are cleared to zeros by a reset.

Chapter 4.

Addressing Modes and Data Types

BC:

HL:

* 1. INTRODUCTION

An instruction is a consecutive list of one or more bytes in memory. Most instructions act upon some data; the term operand refers to the data to be operated upon. For Z280 CPU instructions, operands can reside in CPU registers, memory locations, or I/O ports. The methods used to designate the location of the operands for an instruction are called addressing modes. The Z280 CPU supports nine addressing modes: Register, Immediate, Indirect Register, Direct Address, Indexed, Short Index, Program Counter Relative Address, Stack Pointer Relative, and Base Index. A wide variety of data types can be accessed using these addressing modes.

* 1. ADDRESSING MODE DESCRIPTIONS .

The following pages contain descriptions of the addressing modes for the Z280 CPU. Each description explains how the operand's location is calculated, indicates which address spaces can be accessed with that particular addressing mode, and gives an example of an instruction using that mode, illustrating the assembly language format for the addressing mode. The examples using memory addresses use logical memory addresses; if the MMU is enabled, these logical addresses can be translated to physical addresses before the physical memory is accessed, but this process is not discussed or illustrated here.

INSTRUCTION REGISTER

| OPERATION | REGISTER |—»-| OPERAND~|

THE OPERAND VALUE IS THE CONTENTS OF THE REGISTER.

The operand is always in the register address space. The register length (byte or word) is specified by the instruction opcode.

Example of R mode:

LD BC,HL ;load the contents of HL into BC

*Before instruction execution: After instruction execution:*

BC:

HL:

4.2.2 Immediate (IM)

When the Immediate addressing mode is used, the data processed is in the instruction. . ,

The Immediate addressing mode is the only mode that does not indicate a register or memory address as the source operand. , • .

INSTRUCTION

OPERATION  
:OPERAND

THE OPERAND VALUE IS IN THE INSTRUCTION.

A: 6 7

4.2.1 Register (R, RX) \* •

When this addressing mode is used, the instruction processes data taken from one of the 8-bit registers А, В, C, D, E, H, L, IXH, IXL, IYH, IYL, or one of the 16-bit registers BC, DE, HL, IX, IY, SP, or one of the special byte registers I or R.

Storing data in a register allows shorter instructions and faster execution than occur with instructions that access memory.

Because an immediate operand is part of the instruction, it is always located in the program memory address space. Immediate mode is often used to initialize registers.

Example of IM mode:

LD A,55H ;load hex 55 into the accimulator

*Before instruction execution: After instruction execution:*

In the Indirect Register addressing mode, the register specified in the instruction holds the address of the operand. The data to be processed is at the location specified by the HL register for memory accesses or the C register for I/O and control register space accesses. For the Load Byte instruction, BC and DE can also be used in addition to HL.

* + 1. Indirect Register (IR)
    2. Direct Address (DA)

' DATA MEMORY,

I/O PORT, OR

INSTRUCTION REGISTER CONTROL REGISTER

j OPERATION | REGISTER HE ADDRESS OPERAND |

THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS IN THE REGISTER.

Depending on the instruction, the operand specified by IR mode is located in either the I/O address space (I/O instructions), control register space (Load Control instruction), or data memory

-address space (all other instructions).

The Indirect Register mode can save space and reduce execution time when consecutive locations are referenced or one location is repeatedly accessed. This mode can also be used to simulate more complex addressing modes, since addresses can be computed before the data is accessed.

When the Direct Address addressing mode is used, the data processed is at the location whose memory or I/O port address is in the instruction.

INSTRUCTION

OPERATION

THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS IN THE INSTRUCTION.

DATA MEMORY OR I/O PORT

ADDRESS

Depending on the instruction, the operand specified by DA mode is either in the I/O address space (I/O instructions) or in the data memory address space (all other instructions).

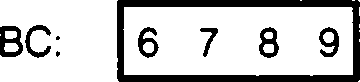
This mode is also used by Jump and Call instructions to specify the address of the next instruction to be executed. (Actually, the address serves as an immediate value that is loaded into the Program Counter.)

Example of DA mode: '

LD BC,(5E22H) ;load BC with the data in

;address 5E22

*Before instruction execution: After instruction execution:*

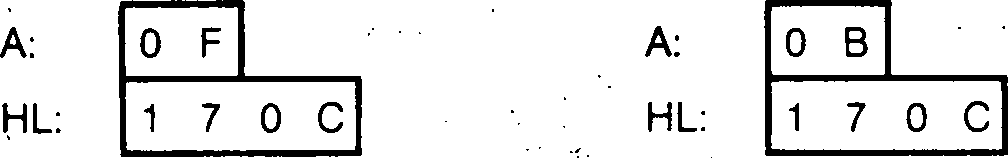
**Example of IR mode: .**

*Data memory:*

LD A,(HL) ;load the accunulator with the data ;addressed by the contents of HL

’ • \* A

*Before instruction execution: After instruction execution:*

5E22: 0 1

*Data memory:*

***t***

5E23: 0 3

170C: о в



* + 1. Indexed (X)

For this addressing mode, the data processed is at the location whose address is the address in the instruction offset by the contents of HL, IX, or IY.

The indexed address is computed by adding the address specified in the instruction to atwos-complement ’'index" contained in the HL, IX or IY register, also specified by the instruction. Indexed addressing allows random access to tables or other complex data structures where the address of the base of the table is known, but the particular element index must be computed by the program. ,

INSTRUCTION

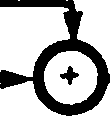
OPERATION I REGISTER

ADDRESS

REGISTER

INDEX

DATA  
MEMORY



THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS THE ADDRESS IN THE INSTRUCTION PLUS THE CONTENTS OF THE REGISTER.

A: 2 3

IX: 0 1 F E

A:

IX:

Operands specified by X mode are always in the data memory address space.

Example of X mode: . •

LD A,(IX + 231AH) ;load into the accumulator ;the contents of the memory ;location whose address

. ;is 231AH + the value in IX

Address calculation:

231A

+01FE .

2518

*Before instruction execution: After instruction execution:*

*Data memory:*

2518: 3 D

* + 1. Short Index (SX) .

When the Short Index addressing mode is used, the data processed is at the location whose address is the contents of IX or IY offset by an 8-bit signed displacement in the instruction. (Note that this addressing mode was called "Indexed" in the Z80 CPU literature.)

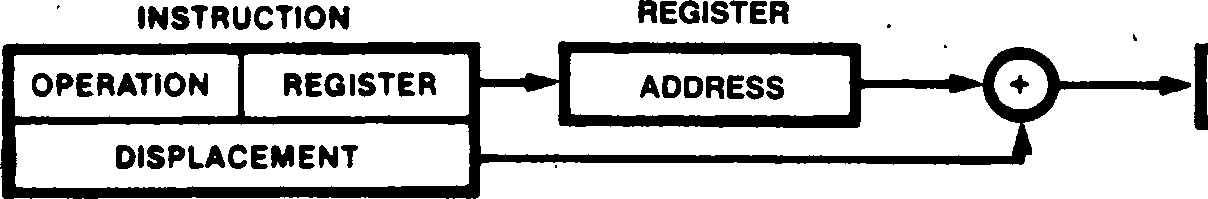
The short indexed address is computed by adding the 8-bit twos-complement signed displacement specified in the instruction to the contents of the IX or IY register, also specified by the instruction. Short Index addressing allows random access to tables or other complex data structures where the address of the base of the table is known, but the particular element index must be computed by the program.

DATA

MEMORY

OPERAND

THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS THE ADDRESS IN THE INSTRUCTION, OFFSET BY THE CONTENTS OF THE REGISTER.



LD A,(IX - 1)

;load into the accumulator the jeontents of the memory location ;whose address is one less than ;the contents of IX

*Data memory:*

2039: 3 D

Operands specified by SX mode are always in the data memory address space.

i

Example of SX mode:

*Before instruction execution: After instruction execution:*

' A: 0 1 A: 3 D

IX: 2 0 3 A IX: 2 0 3 A

Address calculation: FF encoding in the instruc-

This format implies that the assembler will

tion is sign-extended before calculate the displacement from the current PC the address calculation. ’ value to the specified label. Alternatively,

203A

4-FFFF

2039 ***. • i***

4.2.7 Program Counter (PC) Relative Address (RA)

For Program Counter Relative Addressing mode, the data processed is at the location whose address is the contents of the Program Counter offset by an 8- or 16-bit displacement given in the instruction. .

• ' • \*

The instruction specifies a twos-complement signed displacement that is added to the Program Counter to form the target address. Except for extended instructions, the Program Counter value used is the address of the first instruction following the currently executing instruction. For extended instructions, the address used to calculate the displacement is the address of the template.

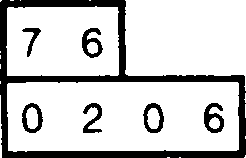
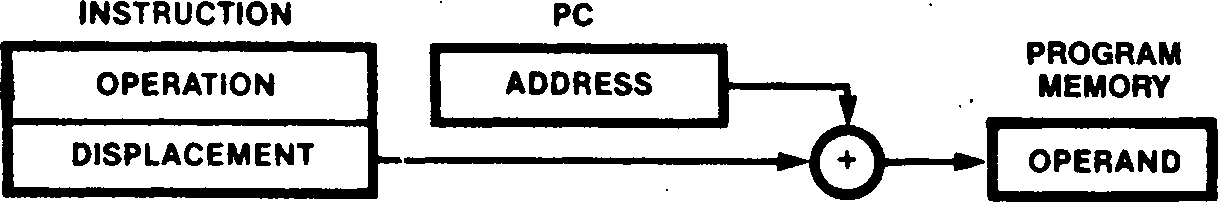
slightly different syntaxes can be used for the RA mode if the actual displacement from the instruction using this mode is known. Thus, this example can also be written in the following manner: ; .

LD A,<$ + 6> ;load the accunulator with the ;contents of the memory location ;whose address is six more than ;the address of the start of this ;LD instruction

or

LD A,(PC + 2) ;load the accunulator with the ;contents of the memory location ;whose address is two more than I the current PC, which now points ;to the next instruction

• • ■ ’ Because the Program Counter is advanced to point to the next instruction when the address



■ THEOPERAND VALUE IS THE CONTENTS OF THE LOCATION

WHOSE ADDRESS IS THE CONTENTS OF PC OFFSET BY THE DISPLACEMENT IN THE INSTRUCTION.

A:

PC:

calculation is performed, the constant that occurs in the instruction is +2. • •• . •

• « ***i***

*Before instruction execution: After instruction execution:*

A:

PC:

An operand specified by RA mode is always in the program memory address space.

The Program Counter Relative Addressing mode is used by certain program control instructions to specify the address of the next instruction to be executed (specifically, the result of the addition of the Program Counter value and the displacement is loaded into the Program Counter). Relative addressing allows references forward or backward from the current Program Counter value; it is used for program control instructions such as Jumps and for Loads that access constants in the program address space.

*Program memory:*

0202:

. 0203:

0204:

’ 0205:

0206:

0207:

LABEL: 0208:

F D

7 8

0 2

0 0

1 8

0 1

7 6

/

instruction

Address calculation:

0206

*± 2*

0208

Example of RA node:

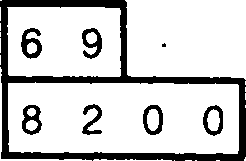
' LD A,<LABEL> ;load the accunulator with the

;contents of the memory location

;whose address is LABEL

* + 1. Stack Pointer Relative (SR)

Example of SR mode



address is the offset by a instruction.

The instruction  
displacement that

A;

SP:

For the Stack Pointer Relative addressing mode, the data processed is at the location whose

contents of the Stack Pointer 16-bit displacement in the specifies a twos-complement is added to the contents of the

Stack Pointer register to form the address. An operand specified by SR mode is always in the data memory address space.

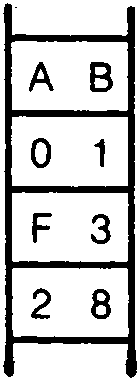
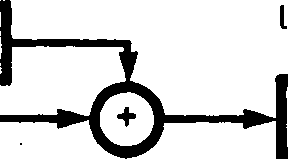
;load into the accunulator {the contents of the memory {location whose address is {two more than the contents {of SP • -

*Before instruction execution: After instruction execution:*

A:

SP:

*Data memory:*



■ INSTRUCTIPN

OPERATION

SP

I ADDRESS

DISPLACEMENT

OPERAND

INSTRUCTION MEMORY

Top of stack

The SR addressing mode is used to items to be found in the stack such

passed to subroutines, or User Stack Pointer is state of the User/System register.

The System

specify data as parameters Stack Pointer

selected depending on the

Address calculation

bit

in the Master Status

8200

**; 2**

8202

8200:

8201:

8202:

8203:

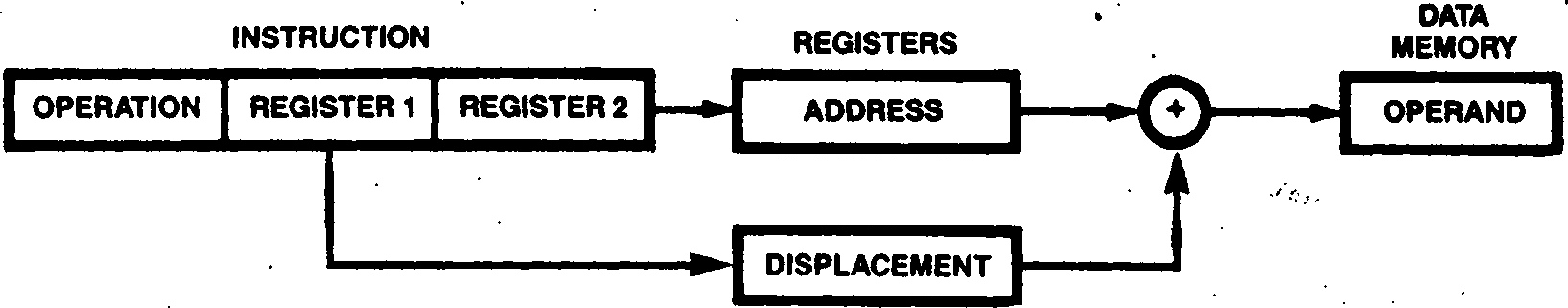
* + 1. Base Index (BX)

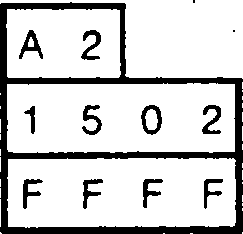
or IY, offset by the contents three registers. . ’ .

For the Base Index addressing mode, the data processed is at the location whose address is the

contents of HL, IX, of another of these

THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS THE CONTENTS OF THE ONE REGISTER OFFSET BY THE DISPLACEMENT IN THE SECOND REGISTER.





*Before instruction execution: After instruction execution:*

A:

HL:

IX:

A:

HL:

IX:

This mode allows access to memory locations whose physical addresses are computed at run time and are not fully known at assembly time. An operand specified by BX mode is always in the data memory address space. \* •

Example of BX mode: ■

L0 A,(HL + IX) {load into the accumulator the {contents of the memory location {whose address is the sum of the {contents of the HL and IX {register

В C

15 0 2

F F F E

*Data memory:*

1500:

Address calculation;

1502 +FFFE

1500

4.3 DATA TYPES

Many data types are supported by the Z280 MPU architecture; that is, many data types have a hardware representation in a Z280 MPU system and instructions that directly apply to them. The Z280 MPU supports operations on bytes, words, - bits, BCD digits, and byte strings.

The basic data type is a byte, which is also the basic addressable element in the register, memory, and I/O address spaces. The 8-bit load, arithmetic, logical, shift, and rotate instructions operate on bytes in registers or memory. Bytes can be treated as logical, signed numeric, or unsigned numeric values. \* .

Operations on two-byte words are also supported. Sixteen-bit load and arithmetic instructions operate on words in registers or memory; words can be treated as signed or unsigned numeric values. I/O reads and writes can be 8-bit or 16-bit operations. Sixteen-bit logical memory addresses can be held and manipulated in 16-bit registers. • •

Bits are fully supported and addressed by number within a byte (see Figure 2-2). Bits within byte registers or byte memory locations can be tested, set, or cleared.

Operations on binary-coded decimal (BCD) digits

are supported by the Decimal Adjust Accunulator and Rotate Digit instructions. BCD digits are stored in byte registers or memory locations, two per byte. The Decimal Adjust Accumulator in­struction is used after a binary addition or subtraction of BCD numbers. The Rotate Digit instructions are used to shift BCD digit strings in memory.

Strings of up to 65,536 bytes can be manipulated by the Z280 CPU's block move, block search, and block I/O instructions. The block move

instructions allow strings of bytes in memory to be moved from one location to another. Block search instructions provide for scanning strings of bytes in memory to locate a particular value. The block I/O instructions allow strings of bytes or words to be transferred between memory and a peripheral device. ***• I***

Arrays are supported by the Indexed, Short Index, and Base Index addressing modes. Stacks are supported by those same modes and the Stack Pointer Relative addressing mode, and by special instructions such as Call, Return, Push, and Pop. A special stack write warning feature aids in the allocation of system stack memory space.

Strings of up to 16 bytes can be transferred between memory and an Extended Processing Unit (EPU) during execution of an extended instruction.

I»

**Chapters.**

**Instruction Set**

**Figure 5-1. Flag Register**

5.1 INTRODUCTION

The Z280 CPU’s instruction set is a superset of the Z80's; the Z280 CPU is opcode compatible with the Z80 CPU. Thus, a Z80 program can be executed on a Z280 MPU without modification. The instruction set is divided into ten groups by function:

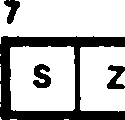
* 8-bit load
* 16-bit load and exchange
* Block transfer and search
* 8-bit arithmetic and logical
* 16-bit arithmetic
* Rotate, shift, and bit manipulation
* Program control

• Input/Output • CPU control • Extended instructions

This chapter describes the instruction set of the Z280 CPUs. First, flags and condition codes are discussed in relation to the instruction set. Then, interruptibility of instructions is discussed and traps are described. The last part of this chapter is a detailed description of each instruction, listed in alphabetic order by mnemonic. This section is intended to be used as a reference for Z280 MPU programmers. The entry for each instruction contains a complete description of the instruction, including addressing modes, assembly language mnemonics, instruction opcode formats, and simple examples illustrating the use of the instruction.

5.2 PROCESSOR FLAGS

The Flag register contains six bits of status information that are set or cleared by CPU operations (Figure 5-1). Four of these bits are testable (C, P/V, Z, and S) for use with conditional jump, call, or return instructions. Two flags are not testable (H, N) and are used for binary-coded decimal (BCD) arithmetic.



•

о н о I p/v n c I

The flags provide a link between sequentially executed instructions, in that the result of executing one instruction may alter the flags, and the resulting value of the flags can be used to determine the operation of a subsequent instruction. The program control instructions whose operation depends on the state of the flags are the Jump, Jump Relative, subroutine Call, and subroutine Return instructions; these instructions are referred to as conditional instructions.

* + 1. Carry Flag (C)

The Carry flag is set or cleared depending on the operation being performed. For add instructions that generate a carry and subtract instructions that generate a borrow, the Carry flag is set to 1. The Carry flag is cleared to 0 by an add that does not generate a carry or a subtract that generates no borrow. This saved carry facilitates software routines for extended precision arithmetic. The multiply and divide instructions use the Carry flag to signal information about the precision of the result. Also, the Decimal Adjust Accumulator instruction leaves the Carry flag set to 1 if a carry occurs when adding BCD quantities.

For the rotate instructions, the Carry flag is used as a link between the least significant and most significant bits for any register or memory location. During shift instructions, the Carry flag contains the last value shifted out of any register or memory location. For logical in­structions the Carry flag is cleared. The Carry flag can also be set and complemented with explicit instructions.

* + 1. Add/Subtract Flag (N)

The Add/Subtract flag is used for BCD arithmetic. Since the algorithm for correcting BCD operations is different for addition and subtraction, this flag is used to record whether an add or subtract was last executed, allowing a subsequent Decimal Adjust Accumulator instruction to perform correctly. See the discussion of the DAA in­struction for further information.

* + 1. Parity/Overflow Flag (PA)

This flag is set to a particular state depending on the operation being performed.

■ . .» ■

For signed arithmetic, this flag, when set to 1, indicates that the result of an operation on twos-complement numbers has exceeded the largest number, or is less than the smallest number, that can be represented using twos-complement notation. This overflow condition can be determined by examining the sign bits of the operands and the result. .

The P/V flag is also used with logical operations and rotate instructions to indicate the parity of the result. The number of bits set to 1 in a byte are counted. If the total is odd, odd parity (P = 0) is flagged. If the total is even, even parity is flagged (P = 1).

During block search and block transfer instructions, the P/V flag monitors the state of the byte count register (BC). When decrementing the byte counter results in a zero value, the flag is cleared to 0, otherwise the flag is set to 1.

• •> .

During the Load Accumulator with I or R register instructions, the P/V flag is loaded with the contents of the Interrupt A enable bit in the Master Status register, w \*\* •

When inputting a byte to a register from an I/O device addressed by the C register, the flag is adjusted to indicate the parity of the data.

* + 1. Half-Carry Flag (H)

The Half-Carry flag (H) is set to 1 or cleared to 0 depending on the carry and borrow status between bits 3 and 4 of an 8-bit arithmetic operation and between bits 11 and 12 of a 16-bit arithmetic operation. This flag is used by the Decimal Adjust Accumulator instruction to correct the result of an addition or subtraction operation on packed BCD data. .

* + 1. Zero Flag (Z)

The Zero flag (Z) is set to 1 if the result generated by the execution of certain instructions is a zero.

For arithmetic and logical operations, the Zero flag is set to 1 if the result is zero! If the result is not zero, the Zero flag is cleared to 0.

For the block search instructions, the Zero flag is set to 1 if a comparison is found between the value in the Accumulator and the memory location pointed to by the contents of the register pair HL. ' .

When testing a bit in a register or memory location, the Zero flag contains the complemented state of the tested bit (i.e., the Zero flag is set to 1 if the tested bit is a 0, and vice-versa). .

For the block I/O instructions, if the result of decrementing В is zero, the Zero flag is set to 1; otherwise, it is cleared to 0. Also for byte inputs to registers from I/O devices addressed by the C register, the Zero flag is set to 1 to indicate a zero byte input.

* + 1. Sign Flag (S)

The Sign flag (S) stores the state of the most significant bit of the result. When the Z280 CPU performs arithmetic operations on signed numbers, binary twos-complement notation is used to represent and process numeric information. A positive number is identified by a zero in the most significant bit. A negative number is identified by a 1 in the most significant bit. .

When inputting a byte from an 1/0 device addressed by the C register to a CPU register, the Sign flag indicates either positive (S - 0) or negative (S = 1) data.

For the Test and Set instruction, the Sign bit is set to 1 if the tested bit is 1, otherwise it is cleared to 0.

■ \* .\* \* \* •

* + 1. Condition Codes . ’

The Carry, Zero, Sign, and Parity/Overflow flags are used to control the operation of the con­ditional instructions. The operation of these in­structions is a function of the state of one of the flags. Special mnemonics called condition codes are used to specify the flag setting to be tested during execution of a conditional instruction; the condition codes are encoded into a 3-bit field in the instruction opcode itself.

Table 5-1 lists the condition code mnemonic, the flag setting it represents, and the binary encoding for each condition code. .

**Table 5-1. Condition Codes**

**Flag Binary**

**Condition Codes for Jump, Call, and Return Instructions**

**Mnemonic Meaning**

**Setting Code**

Condition Codes for Jump Relative Instruction

| Not Zero | Z = 0 | 000 |
| --- | --- | --- |
| Zero | Z = 1 | 001 |
| No Carry | C = 0 | 010 |
| Carry | C = 1 | 011 |
| No Overflow | V = 0 | 100 |
| Parity Odd | V = 0 | 100 |
| Overflow | V = 1 | 101 |
| Parity Even | V = 1 | 101 |
| No Sign | S = 0 | 110 |
| Plus | S = 0 | 110 |
| Sign | S » 1 | 111 |
| Minus | S = 1 | 111 |

NZ Z

NC C

NV PO

V PE

NS **P**

S M

NZ Not Zero Z = 0 100

Z Zero

NC No Carry

C Carry

5.3 INSTRUCTION EXECUTION AM) EXCEPTIONS

Z = 1 101

C = 0 110

C = 1 111

Two types of exception conditions, interrupts and traps, can alter the normal flow of program execution. Interrupts are asynchronous events generated by a device external to the CPU; peripheral devices use interrupts to request service from the CPU. Traps are synchronous

events generated internally in the CPU by particular conditions that occur during

instruction execution. Interrupts and traps are discussed in detail in Chapter 6. This section examines the relationship between instructions and the exception conditions. ***I*** I

* + 1. Instruction Execution and Interrupts

When the CPU receives an interrupt request, and it is enabled for interrupts of that class, the interrupt is normally processed at the end of the current instruction. However, the block transfer and search instructions are designed to be inter­ruptible so as to minimize the length of time it takes the CPU to respond to an interrupt. If an interrupt request is received during a block move, block search, or block I/O instruction, the in­struction is suspended after the current iter­ation. The address of the instruction itself, rather than the address of the following in­struction, is saved on the system stack, so that the same instruction is executed again when the interrupt handler executes an interrupt return instruction. The contents of the repetition counter and the registers that index into the block operands are such that, after each iter­ation, when the instruction is reissued upon returning from an interrupt, the effect is the same as if the instruction were not interrupted. This assumes, of course, that the interrupt handler preserved the registers.

• \* ■ ' .

5.3.2 Instruction Execution and Traps

Traps are synchronous events that result from the execution of an instruction. The action of the CPU in response to a trap condition is similar to the case of an interrupt in interrupt mode 3 (see Chapter 6). All traps except for Extended Instruction, System Stack Overflow Warning, Single Step and 8reakpoint-on-Halt are nonmask­able. »• \*

The Z280 MPU supports eight kinds of traps:

* Division Exception
* Extended Instruction
* Privileged Instruction
* System Call

a Access Violation (page fault and write protect)

* System Stack Overflow Warning ■.$.
* Single Step .

• Breakpoint-on-Halt ' \*

• • .\*\*.\*•\*

The Division Exception trap occurs when executing a divide instruction if either the divisor is zero or the result cannot be represented in the destination (overflow).

«

The Extended Instruction trap occurs when an extended instruction is encountered, but the Extended Processor Architecture is disabled, (the EPA bit in the Trap Control register should be cleared to 0 if there is no EPU in the system or if the Z280 MPU is configured with an 8-bit bus). This allows the same software to be run on Z280 MPU system configurations with or without Extended Processing Units (EPUs). For systems without EPUs, the desired extended instructions can be emulated by software that is invoked by the Extended Instruction trap. For systems with an 8-bit data bus that also have an EPU, the software invoked by the Extended Instruction trap can use I/O instructions to access the EPU. The

information saved on the system stack during this trap is designed to facilitate the 8-bit I/O interface to an EPU by providing address calculation for the operands and by pushing addresses onto the system stack in the reverse order from which they will be used by an I/O interface trap handler.

The Privileged Instruction trap serves to protect the integrity of a system from erroneous or unauthorized actions of user mode processes. Certain instructions, called privileged instructions, can be executed only in system mode. An attempt to execute one of these instructions in user mode causes a Privileged

Instruction trap.

The System Call instruction always causes a trap. This instruction is used to transfer control to system mode software in a controlled way, typically to request operating system services.

The Access Violation trap occurs whenever the Z280 MPU's on-chip MMU detects an illegal memory access. Access Violation traps cause instructions to be aborted. When Access Violation traps occur, the logical address of the instruction is pushed onto the system stack along with the Master Status register; part of the logical address that caused the page fault is latched in the MMU to indicate which page frame caused the fault; and the CPU registers are unmodified, i.e., their contents are the same as just before the instruction execution began. (For block move, block search, or block I/O instructions, the registers are the same as just before the iteration in which the page fault occurred.) \

The System Stack Overflow Warning trap arises when pushing information onto the system stack causes the Stack Pointer to reference a specified 16-byte area of memory. Use of this facility protects the system from system stack overflow errors. •

The Single Step trap occurs with the execution of each instruction, provided the Single-Step control bit in the Master Status register is set to 1. This facilitates software debugging of programs.

The Breakpoint-on-Halt trap occurs whenever the Halt instruction is encountered and the Breakpoint-on-Halt control bit in the Master Status register is set to 1. This facilitates software debugging of programs. • • . . • •

5.4 INSTRUCTION SET FUNCTIONAL GROUPS .

This section presents an overview of the Z280 instruction set, arranged by functional groups. (See Section 5.5 for an explanation of the notation used in Tables 5-2 through 5-11.)

* + 1. 8-Bit Load Group .

This group of instructions (Table 5-2) includes load instructions for transferring data between byte registers, transferring data between a byte register and memory, and loading immediate data into byte registers or memory. All addressing modes are supported for loading between the accumulator and memory or for loading immediate values into memory. Loads between other registers and memory use the IR and SX addressing modes. An exchange instruction is available for swapping the contents of the accumulator with another register or with memory. . .

***■ . •***

The LDUD and LDUP instructions are available for loading to or from the user-mode memory address space while executing in system mode. The CPU flags are used to indicate if the transfer was successfully completed. LDUD and LDUP are privileged instructions. The other instructions in this group do not affect the flags, nor can their execution cause exception conditions.

Table 5-2. 8-Bit Load Group Instructions

|  |  |
| --- | --- |
| • ■ \* ■ . ' | **‘ Addressing Modes Available** |
| **Instruction Name** | **Format R RX IM IR DA X SX RA SR BX** |
| Exchange Accumulator '  Exchange H,L ;  Load Accumulator .  ***г .*** | EXA.src • • ' • • • • • • •  EX H.L ‘ \ .  LDA.src . • • • • • • • ’•••■ • • •  LDdst,A • • • • • • • • • |
| Load Immediate | LD dst.n • • • • • • • • • |
| Load Register (Byte) ' | LD R,src • • • • •  LD dst,R • • • . • |
| Load in User Data Space | LDUDA.src ' ’ • •  LDUDdst.A • • |
| Load in User Program Space | LDUP A.src • • . . . .  LDUP dst.A • . "... • ' . • ' ’ |

**$.4.2 16-Bit Load and Exchange Group** . registers and memory and immediate loads of

: registers or memory. The Load Address instruction

This group of load and exchange instructions facilitates the loading of the address registers

(Table 5-3) allows words of data (two bytes egual with a calculated address. The Push and Pop stack

one word) to be transferred between registers and instructions are also included in this group,

memory. The exchange instructions allow for None of these instructions affect the CPU flags,

switching between the primary and alternate except for EX AF, AF\*. The Push instruction can

register files, exchanging the contents of two cause a System Stack Overflow Warning trap;

16-bit registers, or exchanging the contents of an otherwise, no exceptions can arise from the

addressing register with the top word on the execution of these instructions. • .

stack. The 16-bit loads include transfers between ‘

. • • . ■

*■ I • ■ ’ ' ’ ’ - ' ’ . ' ' '*

**Table 5-3. 16-В it Load and Exchange Group Instructions**

**Addressing Modes Available**

**Instruction Name Format R IM IR DA X SX RA SR BX**

Exchange HL with Addressing Register EX DE,HL

. EXXY.HL

Exchange Addressing Register with Top of Stack EX (SP),XX Exchange Accumulator/Flag with Alternate Bank EX ApAF’

Exchange Byte/Word Registers with Alternate Bank EXX

Load Addressing Register LD XX.src

LD dst.XX

Load Register (Word) LD RR.src

' LDdst.RR

Load Immediate Word ' LDdst.nn

Load Stack Pointer LD SRsrc

LDdst.SP

Load Address LDA XX.src

Pop , POP dst

Push . PUSH src

‘Restricted to an addressing register (HL, IX, or IY).

5.4.7 Block Transfer and Search Group . ■ • ■ ■ ■ •

This group of instructions (Table 5-4) supports block transfer and string search functions. Using these instructions, a block of up to 65,536 bytes can be moved in memory or a byte string can be searched until a given value is found. All the operations can proceed through the data in either direction. Furthermore, the operations can be repeated automatically while decrementing a length counter until it reaches zero, or they can operate on one storage unit per execution with the length counter decremented by one and the source and destination pointer registers properly adjusted. The latter form is useful for implementing more complex operations in software by adding other instructions within a loop containing the block instructions.

Various Z280 MPU registers are dedicated to specific functions for these instructions: the BC register for a counter, the DE and HL registers for memory pointers, and the accumulator for holding the byte value being sought. The repeti­tive forms of these instructions are interruptible; this is essential since the repetition count can be as high as 65,536. The instruction can be interrupted after any iteration, in which case the address of the instruction itself, rather than the next one, is saved on the system stack; the contents of the operand pointer registers, as well as the repetition counter, are such that the instruction can simply be reissued after returning from the interrupt without any visible difference in the instruction execution.

**Table 5-4. Block Transfer and Search Group**

**Instruction Name Format**

Compare and Decrement CPD

Compare, Decrement and Repeat ' CPDR

' Compare and Increment CPI

Compare, Increment and Repeat CPIR'

Load and Decrement LDD

[Load, Decrement and Repeat LDDR](#bookmark181)

Load and Increment LDI

[Load, Increment and Repeat LDIR](#bookmark239)

* + 1. 8-Bit Arithmetic and Logic Group

This group of instructions (Table 5-5) performs 8-bit arithmetic and logical operations. The Add, Add with Carry, Subtract, Subtract with Carry, And, Or, Exclusive Or, Compare, and signed and unsigned Multiply take one input operand from the accumulator and the other from a register, from immediate data in the instruction itself, or from memory. All memory addressing modes are supported: Indirect Register, Short Index, Direct Address,PC Relative Address, Stack Pointer Relative, Indexed, and Base Index. Except for the multiplies, which return the 16-bit result to the HL register, these instructions return the computed result to the accumulator. Both signed and unsigned division are provided. All memory addressing modes except Indirect Register can be used to specify the divisor. .

The Increment and Decrement instructions operate on date in a register or in memory; all memory addressing modes are supported. Three instructions operate only on the accumulator: Decimal Adjust, Complement, and Negate. The final instruction in this group, Extend Sign, takes its 8-bit input from the accumulator and returns its 16-bit result to the HL register.

All these instructions except Extend Sign set the CPU flags according to the computed result. Only the Divide instructions can generate an exception.

**Table 5-5. 8-Bit Arithmetic and Logic Group**

**Addressing Modes Available**

**Instruction Name**

**Format**

**R RX IM IR DA X SX RA SR BX**

Add With Carry (Byte)

Add (Byte) '

And

Compare (Byte)

Complement Accumulator

Decimal Adjust Accumulator

Decrement (Byte)

Divide (Byte)

Divide Unsigned (Byte) .

Extend Sign (Byte)

Increment (Byte)

Multiply (Byte)

ADC A,src ADD A,src AND A,src CP A.src CPLA DAA A

DEC dst DIV A.src DIVU A.src EXTS A INC dst MULT A.src

Multiply Unsigned (Byte)

Negate Accumulator

Or

Subtract With Carry (Byte)

Subtract (Byte)

Exclusive OR

MULTU A.src

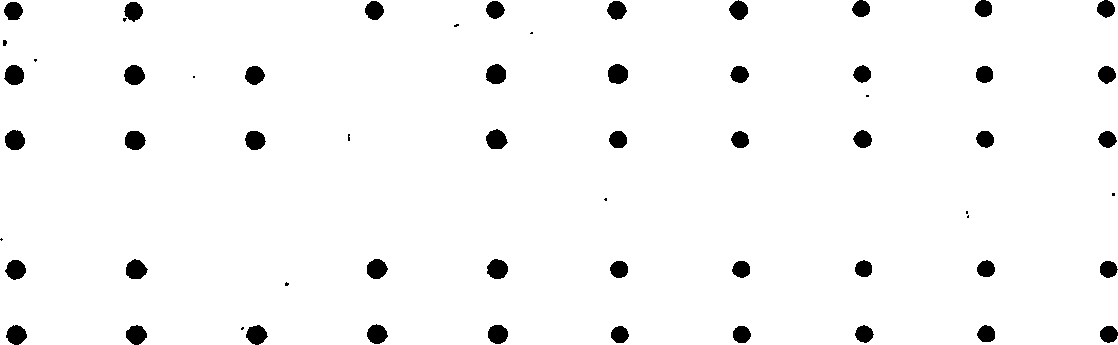
NEG A

OR A.src

SBC A,src

SUB A,src

XOR A.src



* + 1. 16-Bit Arithmetic Operations

This group of instructions (Table 5-6) provides 16-bit arithmetic operations. The Add, Add with Carry, Subtract with Carry, and Compare instructions take one input operand from an addressing register and the other from a 16-bit register or from the instruction itself; the result is returned to the addressing register. The 16-bit Increment and Decrement instructions operate on data found in a register or in memory; the Indirect Register, Direct Address or PC Relative addressing mode can be used to specify the memory operand. The instruction that adds the contents of the accumulator to an addressing register supports the use of signed byte indices into tables or arrays in memory.

The remaining 16-bit instructions provide general arithmetic capability using the HL register as one of the input operands. The word Add, Subtract, Compare, and signed and unsigned Multiply instructions take one input operand from the HL register and the other from a 16-bit register, from the instruction itself, or from memory using Indexed, Direct Address, or Relative addressing mode. The 32-bit result of a multiply is returned to the DE and HL registers, with the DE register containing the most significant bits. The signed and unsigned divide instructions take a 32-bit dividend in the DE and HL registers (the DE register containing the most significant bits) and a 16-bit divisor from a register, from the instruction, or from memory using the Indexed, Direct Address, or Relative addressing mode. The

16-bit quotient is returned to the HL register and the 16-bit remainder is returned to the DE register. The Extend Sign instruction takes the contents of the HL register and delivers the 32-bit result to the DE and HL registers, with the DE register containing the most significant bits of the result. The Negate HL instruction negates the contents of the HL register.

Except for Increment, Decrement, and Extend Sign, all the instructions in this group set the CPU flags to reflect the computed result. The only instructions that can generate exceptions are the Divide instructions.

Add With Carry (Word)

Add (Word)

Add Accumulator to Addressing Register

Add Word

Compare (Word)

Decrement (Word)

Divide (Word)

Divide Unsigned (Word)

Extend Sign (Word)

Increment (Word)

Multiply (Word)

Multiply Unsigned (Word)

Negate HL

Subtract With Carry (Word)

Subtract (Word)

**Table 5-6. 16-Bit Arithmetic Operation Instructions**

**Addressing Modes Available Instruction Name Format R IM IR DA X RA**

ADC XX.src • , . x. .

ADD XX.src ; '

ADD XX,A •

ADDW HL.src . • • • •

CPWHL.src • • • •

DECW dst “ •" • , •

DIV DEHL.src . • • • • '

DIVU DEHL.src • •

EXTSHL . ■ .

INCW dst •. • • •

MULT HL.src • • i •

MULTU HL.src • • • . •

NEG HL . .

SBC XX.src / • . • ...

SUBW HL.src . ’ • • • ■ '

a

* + 1. Bit Manipulation, Rotate and Shift Group

Instructions in this group (Table 5-7) test, set, and reset bits within bytes and rotate and shift byte data one bit position. Bits to be manipulated are specified by a field within the instruction. Rotation can optionally concatenate the Carry flag to the byte to be manipulated. Both left and right shifting is supported. Right shifts can either shift 0 into bit 7 (logical shifts) or can replicate the sign in bits 6 and 7 (arithmetic shifts). The Test and Set instruction is useful in multiprogramming and multiprocessing environments for implementing synchronization mechanisms between processes. All these instructions except Set Bit and Reset Bit set the CPU flags according to the calculated result; the operand can be a register or a memory location specified by the Indirect Register or Short Index addressing modes.

The RLD and RRD instructions are provided for manipulating strings of BCD digits; these rotate 4-bit quantities in memory specified by the indirect register. The low-order four bits of the accumulator are used as a link between rotations of successive bytes.

None of these instructions generate exceptions.

* + 1. Prograa Control Groqp '

• ' \*. \*\* л •

***' I ' >***

This group (Table 5-8) consists of the instructions that affect the Program Counter (PC) and thereby control program flow. The CPU

registers and memory are not altered except for the Stack Pointer and the stack, which play a significant role in procedures and interrupts. (An exception is Decrement and Jump if Non-Zero [DJNZ], which uses a register as a loop counter.) The flags are also preserved except for the two instructions specifically designed to set and complement the Carry flag.

The Jump (JP) and Jump Relative (JR) instructions provide a conditional transfer of control to a new location if the processor flags satisfy the condition specified in the instruction. Jump Relative is a 2-byte instruction that jumps to any instruction within the range -126 to +129 bytes from the location of this instruction. Most conditional jumps in programs are made to locations only a few bytes away; the Jump Relative instruction exploits this fact to improve code compactness and efficiency. ■ ,

**A** special Jump instruction tests whether the primary or auxiliary register file is being used and branches if the auxiliary file is in use. In

**Table 5-7. Bit Manipulation, Rotate and Shift Group**

**Addressing Modes Available  
R IR SX**

**Instruction Name**

**Format**

Bit Test BIT dst

Reset Bit RES dst

Rotate Left RL dst

Rotate Left Accumulator RLA

Rotate Left Circular RLC dst

Rotate Left Circular (Accumulator) : . RLCA Rotate Left Digit ' RLD

Rotate Right RR dst

Rotate Right Accumulator RRA

Rotate Right Circular RRC dst

Rotate Right Circular (Accumulator) RRCA

Rotate Right Digit RRD

Set Bit

Shift Left Arithmetic

Shift Right Arithmetic SRA dst

Shift Right Logical SRL dst

Test and Set TSET dst

**Table 5-8. Program Control Group Instructions**

**Addressing Modes Available Instruction Name Format . IR DA RA**

Call CALLcc.dst • • •

Complement Carry Flag CCF

Decrement and Jump if Non-Zero DJNZ dst •

Jump on Auxiliary Accumulator/Flag JAF dst . •

Jump on Auxiliary Register File in Use JAR dst •

Jump JPcc.dst ' • • •

Jump Relative JRcc.dst •

Return RET cc

Restart RST p • '

System Call SC nn ■ ’

[Set Carry Flag SCF](#bookmark337)

**t • -**

systems that reserve the auxiliary register file for interrupt handlers only (via a software convention), this instruction can be used to decide whether registers must be saved. \* ’ J

Call and Restart are used for calling subroutines; the current contents of the PC are pushed onto the processor stack and the effective address indicated by the instruction is loaded into the PC. The use of a procedure address stack in this manner allows straightforward implementation of nested and recursive procedures. Call, Jump, and Jump Relative can be unconditional or based on the setting of a CPU flag.

Jump and Call instructions are available with the Indirect Register and PC Relative Address modes in addition to the Direct Address mode. These can be useful for implementing complex control structures such as dispatch tables. When using Direct Address mode for a Jump or Call, the operand is used as an immediate value that is loaded into the PC to specify the address of the next instruction to be executed. .

The conditional Return instruction is a companion to the Call instruction; if the condition specified in the instruction is satisfied, it loads the PC from the stack and pops the stack.

*Л* special instruction, Decrement and Jump if Non-Zero (DJNZ), implements the control part of the basic Pascal FOR loop in a one-word instruction.

’ \*

System Call (SC) is used for controlled access to facilities provided by the operating system. It is implemented identically to a trap or interrupt in interrupt mode 3: the current program status is pushed onto the system stack, and a new program status is loaded from a dedicated part of memory.

* + 1. Input/Output Instruction Group

This group (Table 5-9) consists of instructions for transferring a byte, a word, or a string of bytes or words between peripheral devices and the CPU registers or memory. Byte I/O port addresses transfer bytes on ADg-ADy only. Thus in a 16-bit data bus environment, 8-bit peripherals must be connected to bus lines ADg-ADy. In an 8-bit data bus environment, word I/O instructions to external peripherals should not be used; however, on-chip peripherals can still be accessed by word I/O instructions.

The instructions for transferring a single byte (IN, OUT) can transfer data between any 8-bit CPU register or memory address specified in the instruction and the peripheral port specified by the contents of the C register. The IN instruction sets the CPU flags according to the input data; however, special cases of these instructions, restricted to using the CPU accumulator and Direct Address mode, do not affect the CPU flags. Another variant tests an input port specified by the contents of the C register and sets the CPU flags without modifying CPU registers or memory. '

The instructions for transferring a single word (INW, OUTW) can transfer data between the HL register and the peripheral port specified by the contents of the C register. For word I/O, the contents of H appear on ADQ-AD7 and the contents of L appear as ADg-AD^. These instructions do not affect the CPU flags.

The remaining instructions in this group form a powerful and complete complement of instructions for transferring blocks of data between I/O ports and memory. The operation of these instructions is very similar to that of the block move instruc­tions described earlier, with the exception that one operand is always an 1/0 port whose address remains unchanged while the address of the other operand (a memory location) is incremented or decremented. Both byte and word forms of these instructions are available. The automatically

repeating forms of these instructions are inter­ruptible. .

***. 9***

I/O instructions are not privileged if the Inhibit User I/O bit in the Trap Control register is clear; they can be executed in either system or user mode, so that I/O service routines can execute in user mode. The Memory Management Unit and on-chip peripherals’ control and status registers are ' accessed using the I/O instructions. The contents of the I/O Page register are output on AD23-AD-16 with the I/O port address and can be used by external decoding to select specific devices. Pages FF and FE are reserved for on-chip I/O and no external bus transaction is generated. I/O devices can be protected from unrestricted access by using the I/O Page register to select among I/O peripherals.

**Table 5-9. input/Output Instruction Group Instructions**

**Instruction Name Format**

Input IN dst,(C)

Input Accumulator . IN A,(n)

Input HL . INW HL,(C)

Input and Decrement (Byte) ' IND Input and Decrement (Word) INDW

Input, Decrement and Repeat (Byte) INDR -

Input, Decrement and Repeat (Word) IN DRW

Input and Increment (Byte) INI

Input and Increment (Word) . INIW ’ Input, Increment and Repeat (Byte) INIR

Input, Increment and Repeat (Word) INIRW

Output . OUT (C),src

Output Accumulator . OUT (n),A

Output HL OUTW (C),HL

Output and Decrement (Byte) OUTD .

Output and Decrement (Word) OUTDW

Output, Decrement and Repeat (Byte) OTDR .

Output, Decrement and Repeat (Word) OTDRW

Output and Increment (Byte) OUTI .

Output and Increment (Word) . OTIRW

Output, Increment and Repeat (Byte) OTIR : Output, Increment and Repeat (Word) OTIRW

Test Input TSTI (C)

* + 1. CPU Control Group .

The instructions in this group (Table 5-10) act upon the CPU control and status registers or perform other functions that do not fit into any of the other instruction groups. There are three instructions used for returning from an interrupt or trap service routine. Return from Nonmaskable Interrupt (RETN) and Return from Interrupt (RETI)are used in interrupt modes 0, 1, and 2 to pop the Program Counter from the stack and manipulate the Interrupt Mask register, or to signal a reset to Z8400 Family peripherals. The Return from Interrupt Long (RETIL) instruction pops a 4-byte program status from the System stack, and is used in interrupt mode 3 and trap processing, t

Two of these instructions are not privileged: No Operation (NOP) and Purge Cache (PCACHE). The remaining instructions are privileged.

**Table 5-10. CPU Control Group**

**Instruction Name Format**

Disable Interrupt DI mask

Enable Interrupt . El mask

Halt HALT

Interrupt Mode Select IM p

Load Accumulator From I or R Register LD A,src

Load I or R Register From Accumulator LD dst.A

Load Control LDCTL dst.src

No Operation NOP

Purge Cache PCACHE

Return From Interrupt RETI

Return From Interrupt Long RETIL

Return From Nonmaskable Interrupt RETN

A 4-byte long "template" is embedded in each of the extended instruction opcodes. These templates determine the operation to be performed in the EPU itself. The formats of these templates are described in the following pages. The descriptions are from the point of view of the CPU; that is, only CPU activities are described. The operation of the EPU is implied, but the full specification of the instruction template depends on the implementation of the EPU, and is beyond the scope of this manual. Fields in the template that are ignored by the CPU are indicated by asterisks, and would typically contain opcodes that determine any operation to be performed by the EPU in addition to the data transfers specified by the instruction. A 2-bit identification field is included in each template, for use in selecting one of up to four EPUs in a multiple-EPU system.

The action taken by the CPU upon encountering an extended instruction depends upon the EPA control bit in the CPU’s Trap Control register. When this bit is set to 1, indicating that EPUs are included in the system, extended instructions are executed. If this bit is cleared to 0, indicating that there are no EPUs in the system, the CPU executes an extended instruction trap whenever an extended instruction is encountered; this allows a trap service routine to emulate the desired operation in software. <

5,4.10 Extended Instruction Group

The Z280 MPU architecture contains a powerful mechanism for extending the basic instruction set through the use of external co-processors called Extended Processing Units (EPUs). A group of 22 opcodes is dedicated for the implementation of extended instructions using this facility. The extended instructions (Table 5-11) are intended for use on a 16-bit data bus; thus, this facility is available only on the Z-BUS configuration of the Z280 MPU.

**Table 5-11. Extended Instructions**

**Instruction Name Format**

Load EPU From Memory EPUM src

Load Memory From EPU MEPU dst

Load Accumulator From EPU EPUF

EPU Internal Operation EPUI

NOTATION

AM) BINARY ENO

ING



There are four types of extended instructions in the Z280 MPU instruction set: EPU internal operations, data transfers from an EPU to memory, data transfers from memory to an EPU, and data transfers between an EPU and the CPU’s accumulator. The extended instructions that access memory can use any of the six basic memory addressing modes (Indexed, Base Index, PC Relative, SP Relative, Indirect Register, and Direct Address). Transfers between the EPU and CPU accumulator are useful when the program must branch based on conditions generated by an EPU operation.

The rest of this chapter consists of detailed descriptions of the Z280 MPU instructions, arranged in alphabetical order by mnemonic. This section describes the notational conventions used in the instruction descriptions and the binary encoding for register fields within instruction’s operation codes (opcodes).

The description of each instruction begins on a new page. The instruction mnemonic and name is printed in bold letters at the top of each page to enable the reader to easily locate a desired description. The assembly language syntax is then given in a single generic form that covers all the variants of the instruction, along with a list of applicable addressing modes. This is followed by a description of the operation performed by the instruction, a listing of all the flags that are affected by the instruction, a listing of ex­ception conditions that may be caused by execution of the instruction, illustrations of the opcodes for all variants of the instruction, and a simple example of the use of the instruction. •

The following notation is used throughout the descriptions of the instructions:

(addr) A direct address

<addr> An address to be encoded using relative addressing

b A 3-bit field specifying the position of a bit within a byte

BX Base Index addressing mode cc A condition code specifying whether a flag is set to 1 or cleared to 0 d An 8-bit signed displacement

DA Direct Address addressing mode

dd A 16-bit signed displacement

disp The displacement calculated from the

address in relative addressing dst Destination location or contents

IM Immediate addressing mode

IR Indirect Register addressing mode

MSR The Master Status register

**n** 8-bit immediate data

nn 16-bit immediate data

**p** An interrupt mode

PC The Program Counter

PS The program status registers (the Program

Counter and Master Status register)

**R A** single 8-bit register of the set (A,B,C,D,E,H,L); also, R1 and R2 are used when two different registers are specified in the same instruction. (Note that the R register itself is accessed by a single instruction and violates this convention.)

R\* The corresponding 8-bit or 16-bit register in the alternate register file, such as A' RA PC Relative Address addressing mode RR A 16-bit register of the set (BC,DE, HL,SP); also, RRA and RRB are used when two different registers are specified in the same instruction

RX A single byte in the IX or IY registers; that is, a register in the set (IXH,IXL, IYH,IYL); also, RXA and RXB are used when two different registers are specified in the same instruction SP The current Stack Pointer in use SR Stack Pointer Relative addressing mode

src Source location or contents

SX Short Index addressing mode

USP The User Stack Pointer

X Indexed addressing mode

XX One of the 16-bit addressing registers

HL, IX, or IY; also XXA and XXB are used when two different registers are speci­fied in the same instruction

XY One of the 16-bit index registers IX or IY

In the binary encoding of the instruction, lower case is used for the corresponding encoding of the assembler syntax. ... ’ . t

Brackets ([ and ]) are used in the assembly language syntax to indicate an optional field. For example, the 16-bit addition instruction for adding word data to the HL register is described as: x,.

ADDW [HL,]src

This format means the instruction can be written as:

ADDW HL,src . ' ■ ■ or

ADDW src

Assignment of a value is indicated by the symbol ”<—". For example, • \* \* . dst <— dst + src •» • • . 1 ***• • • •*** indicates that the source data is added to the destination data and the result is stored in the destination location.

The notation "addr(n)" is used to refer to bit "n" of a given location, for example, dst(7).

The register field in the binary encoding of an instruction opcode is encoded as shown in Table 5-12.

**Table 5-12. Encoding of 8-Blt Registers in  
Instruction Opcodes**

**Register Encoding**

A 111

**B.** 000

**C** 001

**D** 010

**E** 011 '

н wo

**L** 101

The remainder of this chapter consists of the individual descriptions of each Z280 MPU instruction.

ADC

**Add with Carry (Byte)**

**ADC** [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**Operation:** A \*- A + src + C '

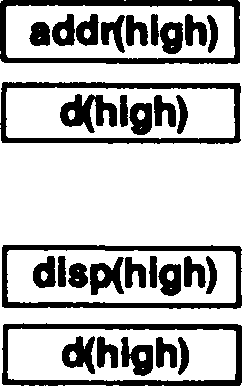
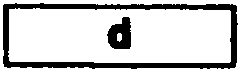
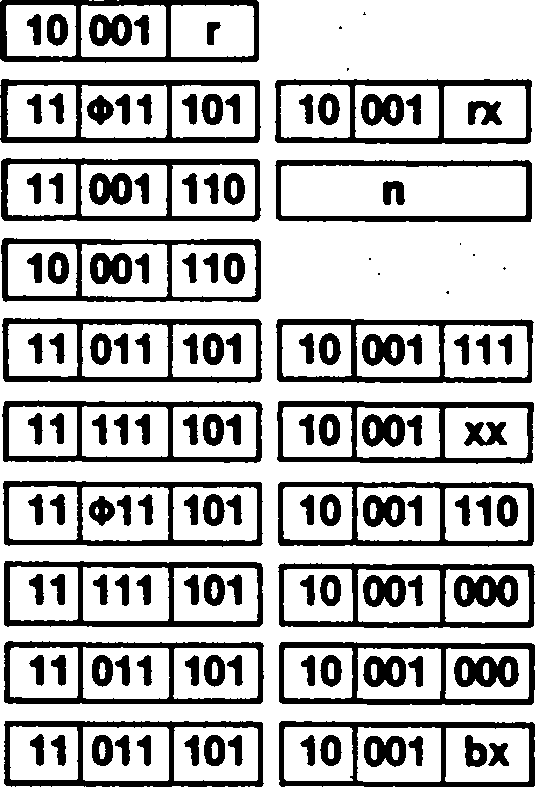
The source operand together with the Carry flag is added to the accumulator and the sum is stored in the accumulator. The contents of the source are unaffected. Twos- complement addition is performed. , /

**Flags: S:** Set if the result is negative; cleared otherwise ; ,...

**Z:** Set if the result is zero; cleared otherwise **H:** Set if there is a carry from bit 3 of the result; cleared otherwise .

**V:** Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise **N:** Cleared **C:** Set if there is a carry from the most significant bit of the result; cleared otherwise

**Exceptions:** None



**Addressing  
Mode**

**Syntax**

**Instruction Format**

**R: RX: IM: IR: DA:**

**X: SX: RA: SR:**

ADC A,R

ADC A,RX

ADC A,n

ADC A,(HL)

ADC A,(addr)

ADC A,(XX + dd)

ADC A,(XY + d) ADC A,<addr>

ADC A,(SP + dd) ADC A,(XXA + XXB)

addr(low)

d(low)

dlspflow) d(low)

**Field Encodings:** ф : **rx: xx: bx:**

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Example:**

| **AF:** | **4** | **8** | **szxhxvnl** |
| --- | --- | --- | --- |
| **HL:** | **2** | **4** | **5 4** |

ADC A,(HL)

Before instruction execution:

Data memory:

2454: Г"8

| **AF:** | **6 1** | **00x1x000** |
| --- | --- | --- |
| **HL:** | **2 4** | **5 4** |
| **Data memory:** | | *S’* |
| **2454:** | **1 8** | « \* • |

After instruction execution:

ADC

**Add With Carry (Word)**

**ADC** dst,src

dst = HL

src = BC, DE, HL, SP or

dst = IX

src = BC, DE, IX, SP or

dst = IY

src = BC, DE, IY, SP

**Operation:**

dst \*- dst + src + C

The source operand together with the Carry flag is added to the destination and the sum is stored in the destination. The contents of the source are unaffected. Twos-complement addition is performed. .

**Flags: S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise . .

**. H:** Set if there is a carry from bit 11 of the result; cleared otherwise .

**V:** Set if arithmetic overflow occurs, that is, if the operands are of the same sign and the result is of the opposite sign; cleared otherwise

**N:** Cleared .

**' C:** Set if there is a carry from the most significant bit of the result; cleared otherwise.

**Exceptions:** None

**Addressing  
Mode Syntax**

ADC HL,RR

1 ADC XY.RR

• •

**Instruction Format**

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **rr** | **010** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **rr** | **010** |
| --- | --- | --- |

**Field Encodings:** ф : о for ix, 1 for iy

**• rr:** 001 for BC, 011 for DE, 101 for add register to itself, 111 for SP

Example: ADC HL,BC '

Before instruction execution: After instruction execution:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **F:** | « •• | **szxhxvnl** | **F:** | **00x0x001** |
| **BC:** | **2 3** | **0 8** | **BC: 2 3** | **0 8** |
| **HL** | **u.** | **3 8** | **HL 1 3** | **, 4 1** |

ADD

**Add Accumulator to Addressing Register**

i

dst = HL, IX, IY •

**ADD** dst,A

**Operation:**

dst ♦- dst + A

The contents of the accumulator are added to the contents of the destination and the result is stored in the destination. The contents of the accumulator are unaffected. The contents of the accumulator are treated as a signed binary integer and are sign- extended to 16 bits; twos-complement addition is performed.

**Flags: S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there is a carry from bit 11 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, if the operands are of the same sign and the result is of the opposite sign from the operands; cleared otherwise

**N:** Cleared > .

**C:** Set if there is a carry from the most significant bit of the result; cleared otherwise

**Example:**

| **E 2** | **szxhxvnc** |
| --- | --- |
| **2 3** | **8 4** |

ADD HL,A

Before instruction execution:

AF: HL

Computation: accumulator is sign-extended.

| **m ie** | **00x1x001** |
| --- | --- |
| **2 8** | **6 6** |

/

After instruction execution: '

AF:

HL

Exceptions: None

|  |  |
| --- | --- |
| **Addressing Mode Syntax** | **instruction Format** |
| ADD HL,A ADD XY,A  •  • | **11| 101 101 01 101 101 ..**  **11 Ф11 101 11 101 101 01 101 101** |

Field Encoding: ф : о for ix, 1 for iy

**FFE2  
+ 2384**

**2366**

**Operation:**

A A + src

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**ADD** [A,]src

The source operand is added to the accumulator and the sum is stored in the ac­cumulator. The contents of the source are unaffected. Twos-complement addition is performed. .

**Flags: S:** Set if the result is negative; cleared otherwise .

**Z:** Set if the result is zero; cleared otherwise **H:** Set if there is a carry from bit 3 of the result; cleared otherwise

**, V:** Set if arithmetic overflow occurs, that is, if both operands are of the same sign and

the result is of the opposite sign; cleared otherwise

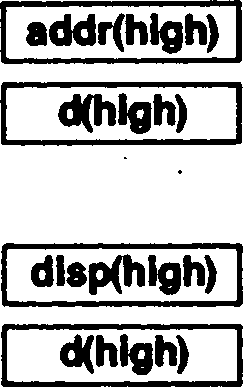
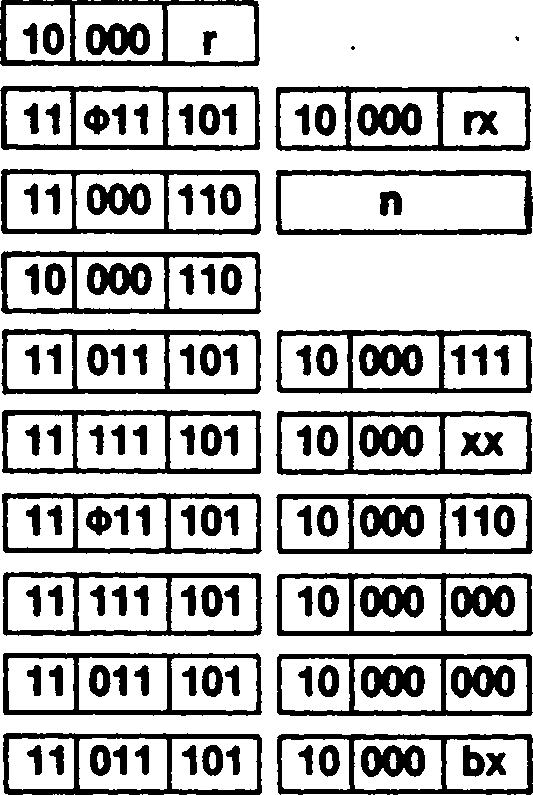
**N:** Cleared

**C:** Set if there is a carry from the most significant bit of the result; cleared otherwise

**Exceptions:** None

**Addressing  
Mode Syntax**

**Instruction Format**



**R:**

**RX: IM: IR:**

**DA:**

**X: SX: RA: SR: BX:**

ADD A,R

ADD A,RX

ADD A,n

ADD A,(HL)

ADD A,(addr)

ADD A,(XX 4-dd) ADD A,(XY + d) ADD A,<addr> ADD A,(SP + dd) ADD A,(XXA + XXB)

addrflow)

d(low)

d(low)

dlsp(low)

**Field Encodings:** ф : **rx: xx: bx:**

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

001 for (HL + IX). 010 for (HL + IY), 011 for (IX + IY)

**Example:**

| **AF:** | **4 8** | **szxhxvnc** |
| --- | --- | --- |
| **HL:** | **2 4** | **5 4** |

ADD A,(HL)

Before instruction execution:

Data memory:

2454: 1~8

| **AF:** | **6** | **0** | **00x1x000** |
| --- | --- | --- | --- |
| **HL** | **2** | **4** | **5 4** |

After instruction execution:

Data memory:

2454: Г~8

**ADD** dst.src dst = HL

src = BC, DE. HL, SP or dst = IX src = BC, DE, IX, SP or dst = IY src = BC, DE, IY, SP

**Operation:** dst4-dst + src .

The source operand is added to the destination and the sum is stored in the destination.

The contents of the source are unaffected. Twos-complement addition is performed.

**Flags: S:** Unaffected

**Z:** Unaffected .

**H:** Set if there is a carry from bit 11 of the result; cleared otherwise

**V:** Unaffected

**N:** Cleared \* v

**C:** Set if there is a carry from the most significant bit of the result; cleared otherwise

**Exceptions:** None

**Addressing  
Mode Syntax**

**Instruction Format**

ADD HL,RR

ADD XY.RR

| **00** | **IT** | **001** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **00** | nr | **001** |
| --- | --- | --- |

**Field Encodings:** ф : о for ix, 1 for iy

**rr:** 001 for BC, 011 for DE, 101 for add register to itself, 111 for SP

**Example:** ADD HL.BC

Before instruction execution: . After instruction execution:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **F:** | | **szxhxvnc** | **F:** | | | **szx0xv01** |
| **BC:** | **2 3** | **0 8** | **BC:** | **2** | **3** | **0 8** |
| **HL:** | **F 0** | **3 8** | **HL:** | **1** | **3** | **4 0** |

4

**ADDW** [HL,]src src = R, IM, DA, X, RA

- > «

**Operation:** HL\*~HL + src .

. The source operand is added to the HL register and the sum is stored in the HL register.

The contents of the source are unaffected. Twos-complement addition is performed.

**Flags: S:** Set if the result is negative; cleared otherwise ,

**Z:** Set if the result is zero; cleared otherwise .

**H:** Set if there is a carry from bit 11 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise

**N:** Cleared ' ч

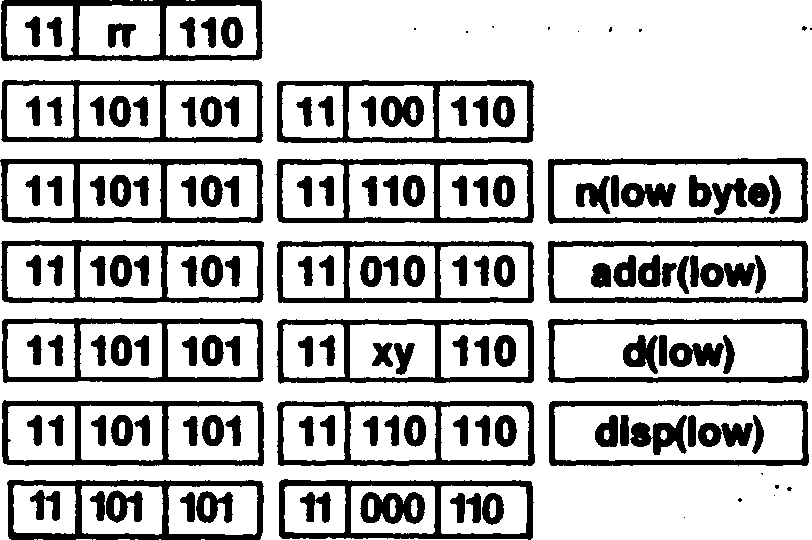
**C:** Set if there is a carry from the most significant bit of the result; cleared otherwise

**Exceptions:** None

***■ . • ,***

**Addressing  
Mode Syntax**

**■ Instruction Format**



**R:**

**IM:**

**DA:**

**X:**

**RA:**

**IR:**

ADDW HL,RR ADDW HL.XY ADDW HL,nn ADDW HL,(addr) ADDW HL,(XY + dd) ADDW HL,<addr> ADDW HL,(HL)

11 101 101

11 Ф11 101

11 111 101

11 011 101

addrfhlgh)

d(hlgh)

11 111 101

11 011 101

dlspfhigh)

11 011 101

n(hlgh byte)

**Field Encodings:** ф : о for ix, 1 for iy

**rr:** 000 for BC, 010 for DE, 100 for HL, 110 for SP

**xy:** 000 for (IX + dd), 010 for (IY + dd)

**Example:** ADDW HL,DE

***. ' • ’ . • . r •***

| **F:** | | **szxhxvnc** | **F:** | | | **10x0x000** |
| --- | --- | --- | --- | --- | --- | --- |
| **DE** | **0 0** | **1 0** | **DE** | **0** | **0** | **1 0** |
| **HL** | **A 1** | **2 3** | **HL** | **A** | **1** | **3 3** |

Before instruction execution: After instruction execution:

**AND** [A,]src

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**Operation:**

A \*- A AND src

AND

**AND**

A logical AND operation is performed between the corresponding bits of the source operand and the accumulator and the result is stored in the accumulator. A 1 bit is stored wherever the corresponding bits in the two operands are both 1s; otherwise a 0 bit is stored. The contents of the source are unaffected. •. ? :

**Flags: S:** Set if the most significant bit of the result is set; cleared otherwise

**Z:** Set if all bits of the result are zero; cleared otherwise

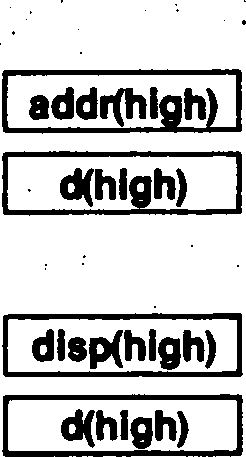
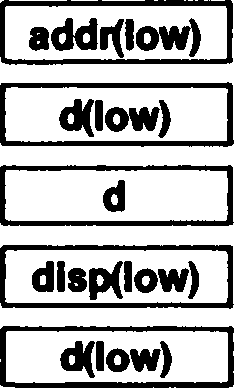
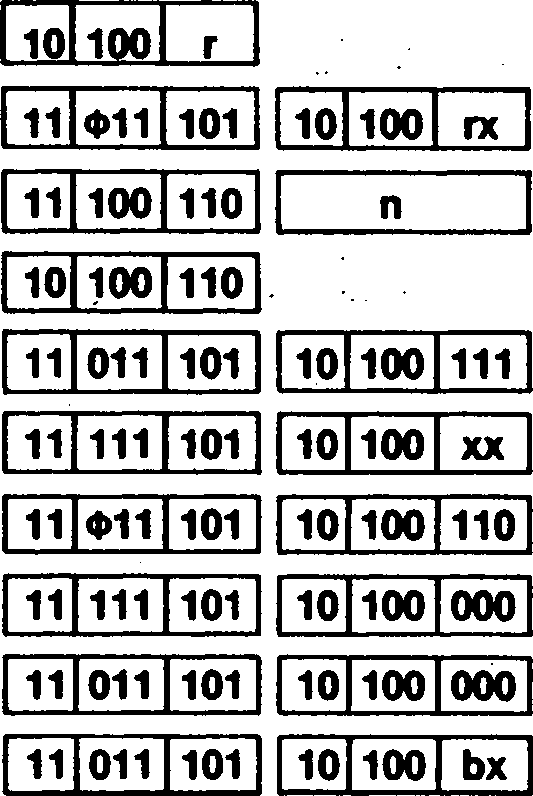
• ' H: Set . :• ■ ... : . .••• > . .> . •••;■ ■ .•

**P:** Set if the parity is even; cleared otherwise

**' N:** Cleared .

**C:** Cleared . .

**Exceptions:** None



**Addressing  
Mode**

**R:**

**RX: IM: IR:**

**DA:**

**X: SX: RA: SR: BX:**

**Syntax**

AND A,R

AND A,RX

AND A,n

AND A,(HL)

AND A,(addr)

AND A,(XX + dd)

AND A,(XY + d) AND A,<addr>

AND A,(SP + dd) AND A,(XXA + XXB)

**Instruction Format**

**Field Encodings:** ф : о for ix, 1 for iy

**. rx:** 100 for high byte, 101 for low byte

**xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

**bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Example:**

AND A,(HL)

| **AF:** | **4 8** | **szxhxpnc** |
| --- | --- | --- |
| **HL:** | **2 4** | **5 4** |

t

Before instruction execution:

Data memory:

2454: Г~8

| **AF:** | **0 8** | **00x1x000** |
| --- | --- | --- |
| **HL:** | **2 4** | **5 4** |

After instruction execution:

Data memory:

2454: iT~S

BIT

**Bit Test**

**BIT** b,dst dst = R, IR, SX

**Operation:** Z\*-NOT dst(b) •

***• f \****

**' • •**

The specified bit b within the destination operand is tested, and the Zero flag is set to 1 if the specified bit is zero, otherwise the Zero flag is cleared to 0. The contents of the

. destination are unaffected. The bit to be tested is specified by a 3-bit field in the instruc-

' tion; this field contains the binary encoding for the bit number to be tested. The bit

• number must be between 0 and 7.

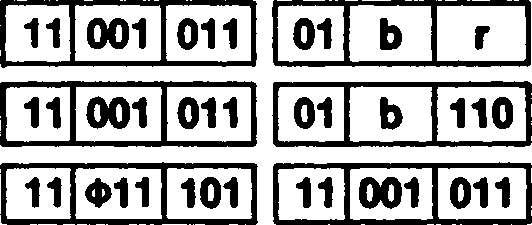
**Flags: S:** Unaffected

**Z:** Set if the specified bit is zero; cleared otherwise **H:** Set . -

**P:** Unaffected

**N:** Cleared **C:** Unaffected

**Exceptions:** None



Addressing  
Mode

**Syntax**

**Instruction Format**

R:

IR:

SX:

BIT

BIT

BIT

b,R b,(HL)

01 b 110

Field Encoding:

Ф: 0 for IX, 1 for IY

**Example**

Before instruction execution:

After instruction execution:

AF

00010110

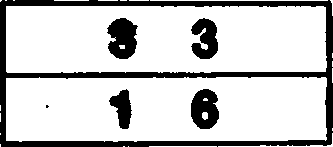
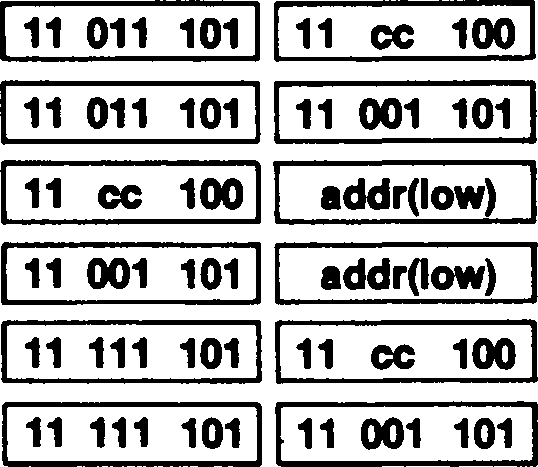
szxhxpnc

AF:

00010110

s0x1xp0c

BIT 1,A



**. . I**

**‘ . \* \* ' '**

**CALL**

|  | **Call** |
| --- | --- |
|  | **CALL** [cc,]dst dst = IR, DA, RA .  1 . |
| **Operation:**  ***• 9***  ***I*** | If the cc is satisfied then: SP ♦-SP - 2 . ’  (SP) <- PC '  PC<-dst  ***. • • . • . r \* '***  • • • • • • • .• • • .  A conditional call transfers program control to the destination address if the setting of a selected flag satisfies the condition code “cc” specified in the instruction; an uncondi­tional call always transfers control to the destination address. The current contents of the Program Counter (PC) are pushed onto the top of the stack; the PC value used is the address of the first instruction byte following the Call instruction. The destination address is then loaded into the PC and points to the first instruction of the called procedure. At |
| }. . ■ ' | the end of a procedure a return instruction (RET) can be used to return to the original .  program. |
|  | Each of the Zero, Carry, Sign, and Overflow flags can be individually tested and a call performed conditionally on the setting of the flag. . |
| • ’ ■ | When using DA mode with the CALL instruction, the operand is not enclosed in paren­theses. ' .  • ■ • . . • • • . . . • |
| **‘ Flags:** | No flags affected |
| **Exceptions:** | System Stack Overflow Warning . |
| **Addressing Mode** | к • .  **Syntax Instruction Format** |

“unconditional call”

**IR:** CALL cc,(HL)

CALL (HL)

**DA:** CALL cc.addr

CALL addr

**RA:** CALL cc,<addr>

CALL <addr>

addr(hlgh) addr(hlgh) dlspQow) dlspflow)

“unconditional call”

dlsp(hlgh)  
disp(hlgh)

“unconditional call”

**Field Encoding: cc :** ООО for NZ, 001 for Z, 010 for NC, 011 for C, 100 for PO or NV, 101 for PE or V, . 110 for P or NS, 111 for Mor S .

**Example:**

CALL 2520H

| **PC:** | **1 6** | **3 0** |
| --- | --- | --- |
| **SP:** | **F F** | **2 6** |

Before instruction execution:

Data memory:

| **PC:** | **2** | **5** | **2** | **0** |
| --- | --- | --- | --- | --- |
| **SP:** | **F** | **F** | **2** | **4** |

After instruction execution:

Data memory:

FF24:

FF25:

**FF24:**

**FF25:**

CCF

**Complement Carry Flag**

**CCF**

**Operation:**

C \*- NOT C

The Carry flag is inverted.

**Flags: S:** Unaffected . •

**Z:** Unaffected ь

**H:** The previous state of the Carry flag

**P:** Unaffected

**N:** Cleared . ✓

**C:** Set if the Carry flag was clear before the operation; cleared otherwise

**Exceptions:** None

**Addressing  
Mode Syntax**

**Instruction Format**

CCF

00 111 111

**Example:**

CCF

After instruction execution: , ■

Before instruction execution:

szxhxvnO

F: szxOxvO!



**CP** [A,]src

**СР**

**Compare (Byte)**

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**Operation:** A - src

I

The source operand is compared with the accumulator and the flags are set according­ly. The contents of the accumulator and the source are unaffected. Twos-complement subtraction is performed.

**Flags: S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there is a borrow from bit 4 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, if the operands are of different signs and the result is the same sign as the source; cleared otherwise

**N:** Set

**C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise

**Exceptions:** None

Addressing  
Mode

R: RX: IM: IR: DA:

X: SX: RA: SR: BX:

Syntax

CP A,R

CP A,RX

CP A,n

CP A,(HL)

CP A,(addr)

СР A,(XX + dd)

CP A,(XY + d)

CP A,<addr>

CP A,(SP + dd)

CP A,(XXA + XXB)

Field Encodings: ф :

rx:

bx:

Instruction Format

d(low)

addr(low)

disp(low)

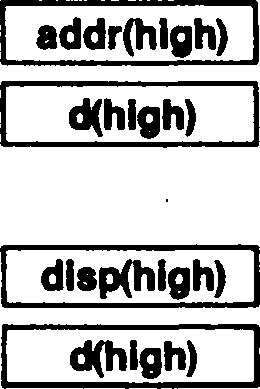
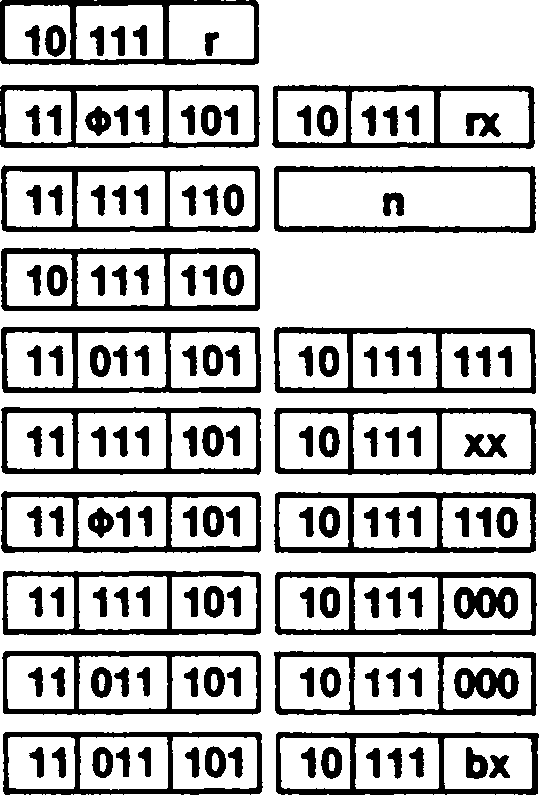
d(low)

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)



CP A,(HL)

**Example:**

Before instruction execution:

|  |  |  |
| --- | --- | --- |
| **AF:** | **4 8** | **szxhxvnc** |
| **HL:** | **2 4** | **5 4** |

Data memory:

After instruction execution:

|  |  |  |
| --- | --- | --- |
| **AF:** | **4 8** | **00x0x010** |
| **HL:** | **2 4** | **5 4** |

Data memory:

**2454: 1 8**

**2454: 1 8**

CPD

**Compare and Decrement**

**CPD**

**Operation:** A - (HL)

HL «-HL - 1

BC«- BC - 1

This instruction is used for searching strings of byte data. The byte of data at the loca­tion addressed by the HL register is compared with the contents of the accumulator and the Sign and Zero flags are set to reflect the result of the comparison. The contents of the accumulator and the memory bytes are unaffected. Twos-complement subtraction is performed. Next the HL register is decremented by one, thus moving the pointer to the previous element in the string. The BC register, used as a counter, is then decremented by one.

**Flags: S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero, indicating that the contents of the accumulator and the memory byte are equal; cleared otherwise •

**H:** Set if there is a borrow from bit 4 of the result; cleared otherwise

**V:** Set if the result of decrementing BC is not equal to zero; cleared otherwise

**N:** Set .

**C:** Unaffected .

**, Exceptions:** None

**Addressing  
Mode**

**Syntax**

**Instruction Format**

CPD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **101** | **101** | **10** | **101** | **001** |

**Example:**

‘ Before instruction execution: After instruction execution:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AF:** | **co** | **szxhxvnc** | **AF:** | **3 В** | **01x0x01c** |
| **HL:** | **1 2** | **1 5** | **HL:** | **1 2** | **1 4** |
| **BC:** | **0 0** | **0 1** | **BC:** | **0 0** | **0 0** |

Data memory: . Data memory:

. \*•\* \*

1215: 3 В I 1215: 3 В

**CPDR**

Compare, Decrement and Repeat

**CPDR**

***f***

**Operation:** Repeat until BC = 0 or match: A-(HL) • ’ .

HL \*- HL - 1 •’

BC\*-BC - 1

This instruction is used for searching strings of byte data. The bytes of data starting at the location addressed by the HL register are compared with the contents of the ac­cumulator until either an exact match is found or the string length is exhausted. The Sign and Zero flags are set to reflect the result of the last comparison. The contents of the . accumulator and the memory bytes are unaffected. Twos-complement subtraction is per­formed. •

After each comparison, the HL register is decremented by one, thus moving the pointer to the previous element in the string. The BC register, used as a counter, is then de­cremented by one. If the result of decrementing the BC register is not zero and no match has been found, the process is repeated. If the contents of the BC register are zero at the start of this instruction, a string length of 65,536 bytes is indicated.

This instruction can be interrupted after each execution of the basic operation. The Pro­gram Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed. '

**Flags: ' S:** Set if the last result is negative; cleared otherwise

**Z:** Set if the last result is zero, indicating that the contents of the accumulator and the memory byte are equal; cleared otherwise \*

**H:** Set if there is a borrow from bit 4 of the last result; cleared otherwise

**V:** Set if the result of decrementing BC is not equal to zero; cleared otherwise

N: Set . % ’

**C:** Unaffected

**Exceptions:** None

**Addressing  
Mode**

**Syntax**

**Instruction Format**

CPDR

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **111** | **001** |
| --- | --- | --- |

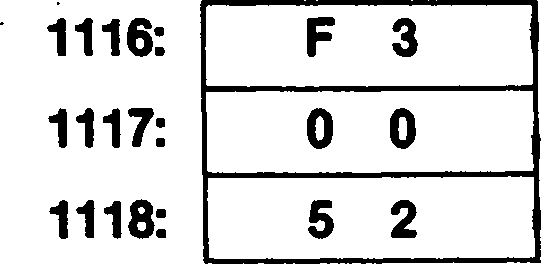
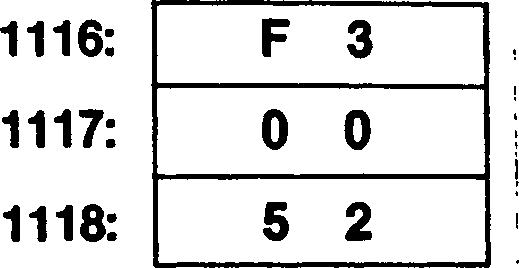
**Example:** CPDR

| **AF:** | **F 3** | **szxhxvnc** |
| --- | --- | --- |
| **HL:** | **1 1** | **1 8** |
| **BC:** | **0 0** | **0 7** |

Before instruction execution:

| **AF:** | **F 3** | **01x0x11c** |
| --- | --- | --- |
| **HL:** | **1 1** | **1 5** |
| **BC:** | **0 0** | **0 4** |

After instruction execution:



Data memory:

Data memory:

**CPI**

**Compare and Increment**

**CPI**

**Operation:** A - (HL)

HL\*-HL + 1

BC \*- BC - 1

This instruction is used for searching strings of byte data. The byte of data at the loca­tion addressed by the HL register is compared with the contents of the accumulator and the Sign and Zero flags are set to reflect the result of the comparison. The contents of the accumulator and the memory bytes are unaffected. Twos-complement subtraction is performed. •

Next the HL register is incremented by one, thus moving the pointer to the next element in the string. The BC register, used as a counter, is then decremented by one.

**Flags: S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero, indicating that the contents of the accumulator and the memory byte are equal; cleared otherwise

**H:** Set if there is a borrow from bit 4 of the result; cleared otherwise

**V:** Set if the result of decrementing BC is not equal to zero; cleared otherwise

**N:** Set

**C:** Unaffected

**Exceptions**

None

**Addressing  
Mode**

**Syntax**

**Instruction Format**

CPI

11 101 101 10 100 001

**Example**

CPI

| **3 В** | **szxhxvnc** |
| --- | --- |
| **1 2** | **1 5** |
| **0 0** | **0 1** |

Before instruction execution:

AF: HL\* BC:

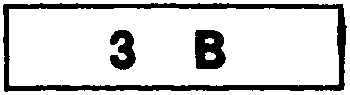
| **3 в** | | **01x0x01c** |
| --- | --- | --- |
| **12** | | **1 6** |
| **0** | **0** | **0 0** |

After instruction execution:

AF:

HL:

BC:



Data memory:

Data memory:

1215:

**1215:**

CPIR

**Compare, Increment and Repeat**

**CPIR**

**Operation:** Repeat until BC = 0 or match: A - (HL) .

HL\*-HL + 1 '

BC \*- BC - 1

This instruction is used for searching strings of byte data. The bytes of data starting at . the location addressed by the HL register are compared with the contents of the ac-

• cumulator until either an exact match is found or the string length is exhausted. The

Sign and Zero flags are set to reflect the result of the comparison. The last contents of the accumulator and the memory bytes are unaffected. Twos-complement subtraction is performed. '

After each comparison, the HL register is incremented by one, thus moving the pointer to the next element in the string. The BC register, used as a counter, is then de­cremented by one. If the result of decrementing the BC register is not zero and no match has been found, the process is repeated. If the contents of the BC register are ... zero at the start of this instruction, a string length of 65,536 bytes is indicated.

This instruction can be interrupted after each execution of the basic operation. The Pro­. gram Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed. ,

**Flags: S:** Set if the last result is negative; cleared otherwise

**Z:** Set if the last result is zero, indicating that the contents of the accumulator and the memory byte are equal; cleared otherwise

**H:** Set if there is a borrow from bit 4 of the last result; cleared otherwise

**V:** Set if the result of decrementing BC is not equal to zero; cleared otherwise .

**N:** Set .

**C:** Unaffected

**Exceptions:** None

Addressing Mode Syntax Instruction Format

CPIR

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **110** | **001** |
| --- | --- | --- |

**Example:**

CPIR

| **AF:** | **F 3** | **szxhxvnc** |
| --- | --- | --- |
| **HL** | **1 1** | **1 8** |
| **BC:** | **0 0** | **0 7** |

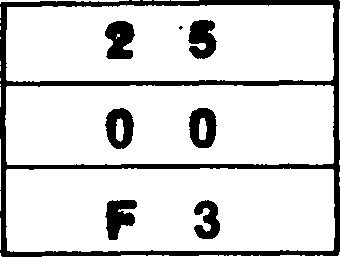
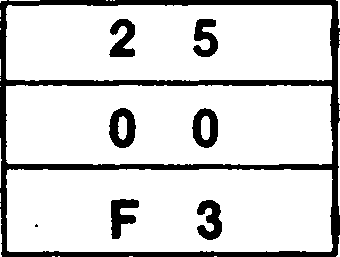
Before instruction execution:

Data memory:

| **AF:** | **F 3** | | **01x0x11c** |
| --- | --- | --- | --- |
| **HL:** | **1 1** | | **1 В** |
| **BC:** | **0** | **0** | **0 4** |

After instruction execution:

Data memory:



1118:

1119:

111A:

1118:

1119:

111A:

CPL

**Complement Accumulator**

**CPL** [A] • • ,

**Operation:** A <\*- NOT A

**• I**

The contents of the accumulator are complemented (ones complement); all 1 bits are changed to 0 and vice-versa.

**Flags: S:** Unaffected

**Z:** Unaffected .

**H:** Set **V:** Unaffected • . •

**N:** Set **C:** Unaffected

**Exceptions:** None

Addressing  
Mode Syntax

Instruction Format

CPL A

00 101 111

**Example:**

CPL A

| **2 8** | **szxhxvnc** |
| --- | --- |

Before instruction execution:

AF:

After instruction execution:

AF: D 7 | szxlxvlc

**CPW** [HLJsrc

src = R, IM, DA, X, RA

CPW

**Compare (Word)**

**Operation:** HL - src •

- • •

The source operand is compared with the HL register and the flags are set accordingly. The contents of the source and HL are unaffected. Twos-complement subtraction’ is performed.

**Flags: S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise .

**H:** Set if there is a borrow from bit 12 of the result; cleared otherwise

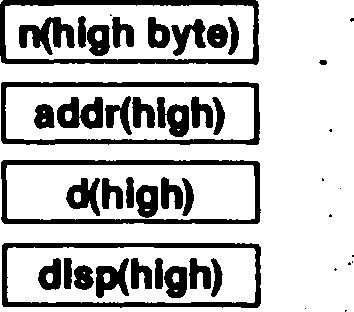
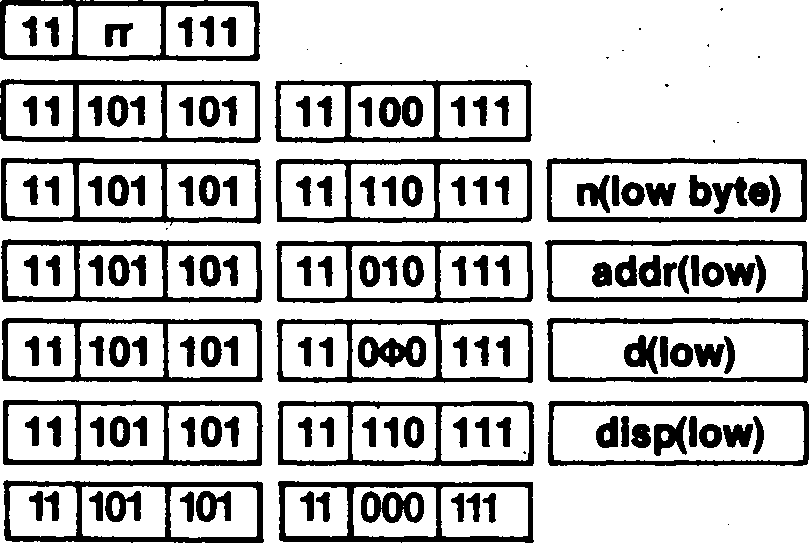
**V:** Set if arithmetic overflow occurs, that is, if the operands are of different signs and the result is the same sign as the source; cleared otherwise

**N:** Set 1

**C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise

**Exceptions:** None

Addressing Mode Syntax Instruction Format



R:

IM:

DA:

X:

RA:

IR:

CPW HL,RR CPW HL.XY CPW HL,nn CPW HL,(addr) CPW HL,(XY + dd) CPW HL,<addr> CPW HL,(HL)

| **11** | **101** | **101** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| 11 | **111** | **101** |
| --- | --- | --- |

| **11** | **011** | **101** |
| --- | --- | --- |

| **11** | **111** | **101** |
| --- | --- | --- |

| **11** | **011** | **101** |
| --- | --- | --- |

| **11** | **011** | **101** |
| --- | --- | --- |

Field Encodings: ф : о for ix, 1 for iy

**• rr:** 000 for BC, 010 for DE, 100for HL, 110for SP

**Example:** CPW HL,DE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **szxhxvnc** | **F:** | | | **10x0x010** |
| **0 0** | **10** | **DE:** | **0** | **0** | **10** |
| **A 1** | **CM**  **CM** | **HL:**  ***1*** | **A** | **1** | **2 3** |

After instruction execution:

Before instruction execution:

DE:

HL:

DAA

**Decimal Adjust Accumulator**

**DAA**

**Operation:** A \*-Decimal Adjust A . ..

The accumulator is adjusted to form two 4-bit BCD digits following a binary, twos-complement addition or subtraction on two BCD-encoded bytes. The table below indicates the operation performed for addition (ADD, ADC, INC) or subtraction (SUB, SBC, DEC, NEG). .

**Operation of DAA Instruction .**

**Hex Value in C Before Upper Digit**

**Operation DAA (Bits 7-4)**

**H Before  
DAA**

**Hex Value in Number**

**Lower Digit Added C After H After  
(Bits 3-0) to Byte DAA DAA**

0 0-9

0 0-8

ADD 0 0-9

ADC 0 A-F

INC 0 9-F

(N = 0) 0 A-F

**1** , 0-2

**1** 0-2

**1** 0-3

0 0

1 0 0

1 0 0

1

0-9 00 0

A-F 06 0

0-3 06 0

0-9 60 1

A-F 66 1

0-3 66 . 1

0-9 60 1

A-F 66 1

0-3 . 66 1

0 **1** 0

0 1 0

0 1 0

SUB 0 0-9 0 0-9 00

O' 1 0

**1**

SBC 0 0-8 1 6-F FA

DEC **1** 7-F 0 0-9 A0

NEG **1** 6-F **1** 6-F 9A

(N = 1) ■ '

The operation is undefined if the accumulator was not the result of a binary addition or sub­traction of BCD digits. . ■

***• i \* - ’ '***

**Flags: S:** Set if the most significant bit of the result is set; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise .

**H:** See table above

**P:** Set if the parity of the result is even; cleared otherwise

**N:** Not affected

**C:** See table above

**Exceptions:** None

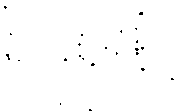
**Addressing  
Mode Syntax**

**Instruction Format**

DAA

00 100 111

|  |  |
| --- | --- |
| **Example:** DAA  Before instruction execution:  **AF: 2 • szxOxpOl**  • 4 1 / ***I i i г*** • < 4. ***i*** i s (  ***'-J,*** . • • • i  . 1  • « • t •  • • ***t • ' •***  ***t • • ’***  ***• t***  / ***i*** • 1 . \* | ***i ' ’ ’***  After instruction execution:  **J AF: [8 8 00x0x001**  • • • \* • ***: I f*** \* • r • • / ' ' , ' ’ ' / ; r • • • . • » \* » ‘ < » • < - 4 ' -  • » \* • ' • • \*• a • • ■ f • 4 ***t t*** 1 • ф  • • • • \* • , \* . • • \* ’ • » \* « • • ’ . . '  ***f*** |



DEC

**Decrement (Byte)**

**DEC** dst dst = R, RX, IR, DA, X, SX, RA, SR, BX

**Operation:** dst \*- dst - 1

**• ' • ’ >**

The destination operand is decremented by one and the result is stored in the destina­tion. Twos-complement subtraction is performed.

**Flags: S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there is a borrow from bit 4 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, if the destination was 80н; cleared otherwise **N:** Set .

**C:** Unaffected

Exceptions: None

Addressing  
Mode Syntax

**R:** DEC R

**RX:** DEC RX

**IR:** DEC (HL) '

**DA:** DEC (addr)

**X:** DEC (XX +dd)

**SX:** DEC (XY + d)

**RA:** . DEC <addr>

**SR:** DEC (SP + dd)

**BX:** DEC (XXA + XXB)

Instruction Format

addr(low)

d(low)

dlsp(low) d(low)

Field Encodings: ф :

rx: xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Example:**

DEC (HL)

Before instruction execution:

After instruction execution:

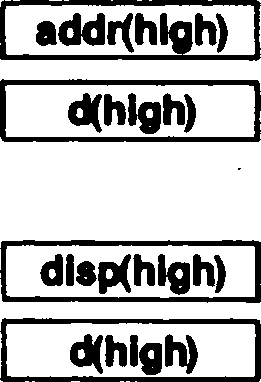
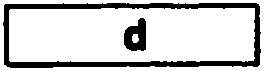
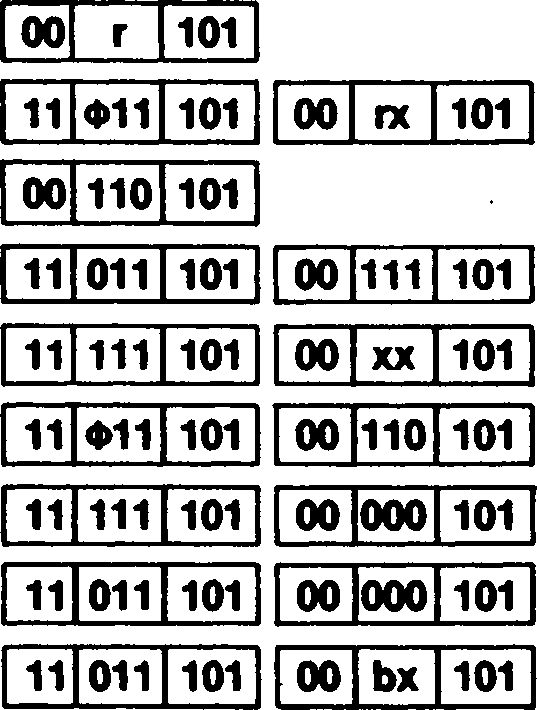
HL:

szxhxvnc  
5 4

HL

10x0x01c

5 4



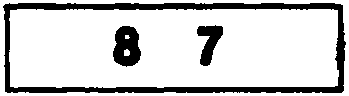
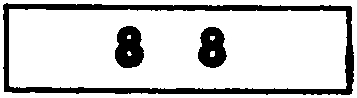
Data memory:

Data memory:

2454:

2454

*f*



**DEC[W]**

**Decrement (Word)**

< \*

**DEC[W]** dst • ' dst = R

or **DECW** dst dst = IR, DA, X, RA

**Operation:** dst dst — 1 «

The destination operand is decremented by one. Twos-complement subtraction is performed. ; \*

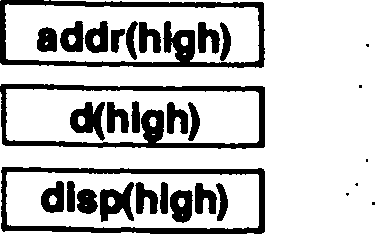
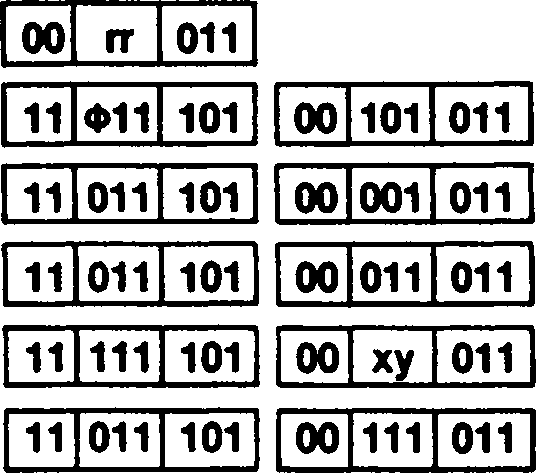
**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format



R:

IR:

DA:

X:

RA

Field Encodings: ф : rr: xy:

addrflow)

d(low)

dlsp(low)

DECW RR DECW XY DECW (HL) DECW (addr) DECW (XY + dd) DECW <addr>

0 for IX, 1 for IY

001 for BC, 011 for DE, 101 for HL, 111 for SP

001 for (IX + dd), 011 for (IY + dd)

**Example:**

DECW HL

Before instruction execution:

HL: I 2 3 0 8

After instruction execution:

HL:

**DI**

Disable Interrupt

Mask = Hex value between 0 and 7Fh

**DI** mask

If mask(i) = 1 then MSR(i) \*- 0

**Operation:**

The designated interrupt control bits in the Master Status register (MSR) are cleared to 0, thus disabling all interrupts on these inputs; all other interrupt enables in the MSR are unaffected. If no mask is present then all interrupts are disabled.

Any combination of interrupt enables in the MSR can be specified. The seven bits in the mask field in the instruction correspond to the seven interrupt enable bits in the MSR, mask bit i corresponding to MSR bit i.

**Flags:**

No flags affected

**Exceptions:**

Privileged Instruction

Addressing  
Mode

Syntax

Instruction Format

**Example:**

DI

DI mask

11 110 011

1l|l0l|l01 ||O1|11O|111

mask

Mask = byte specifying which interrupts to disable: mask(i) corresponds to interrupt source i;

mask(7) must be zero. .

DI 23H

Before instruction execution:

After instruction execution

MSR:

MSR:

**' DIV** [HL,]src src = R, RX, IM, DA, X, SX, RA, SR, BX

**Operation:** A\*-HL + src

L remainder

The contents of the HL register (dividend) are divided by the source operand (divisor) and the quotient is stored in the accumulator; the remainder is stored in the L register. The contents of the source and the H register are unaffected. Both operands are treated as signed, twos-complement integers and division is performed so that the remainder is of the same sign as the dividend. .

There are three possible outcomes of the DIV instruction, depending on the division and the resulting quotient:

1. If the quotient is within the range -27 to 27-1 inclusive, then the quotient is left in the accumulator, the Overflow flag is cleared to 0, and the Sign and Zero flags are set according to the value of the quotient.
2. If the divisor is zero, the accumulator remains unchanged, the Zero and Overflow flags are set to 1, and the Sign flag is cleared to 0. Then the Division Exception trap is taken. .
3. If the quotient is outside the range -27 to 27-1, the accumulator remains un­changed, the Overflow flag is set to 1, and the Sign and Zero flags are cleared to 0. Then the Division Exception trap is taken.

**Rags: S:** Cleared if V flag is set; else set if the quotient is negative, cleared otherwise .

**Z:** Set if the quotient or divisor is zero; cleared otherwise

**H:** Unaffected

**V:** Set if the divisor is zero or if the computed quotient lies outside the range from-27 to 27-1; cleared otherwise

**N:** Unaffected . .

**C:** Unaffected ’

**Exceptions:** Division Exception

Addressing  
Mode Syntax

Instruction Format

R: RX: IM: DA:

X: SX: RA: SR: BX:

IR:

DIV HL,R DIV HL.RX DIV HL,n DIV HL.(addr) DIV HL,(XX + dd) DIV HL,(XY 4- d) DIV HL,<addr> DIV HL,(SP + dd) DIV HL,(XXA + XXB) DIV HL,(HL)

| **11** | **101** | **101** | **11** | **r** | **100** | • . | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **11** | **rx** | **100** |  |  |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **111** | **100** | **n** |  |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **111** | **100** | **addr(low)** | **addrfhigh)** |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **XX** | **100** | **d(low)** | **d(hlgh)** |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **11** | **110** | **100** | **d** |  |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **000** | **100** | **disp(low)** | **dlspfhigh)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **000** | **100** | **d(low)** | **d(hlgh)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **bx** | **100** | **• .** | |

11 101 101 11 100 100

| **11** | **101** | **101** |
| --- | --- | --- |

| **11** | **100** | **100** |
| --- | --- | --- |

t

**Example:** DIV HL,C

Before instruction execution: After instruction execution:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **AF:** | **5 5** | **szxhxvnc** | **AF:** | **0** | **1** | **OOxhxOnc** |
| **C:** |  | **F E** | **C:** |  |  | **F E** |
| **HL:** | **F F** | **F D** | **HL** | **F** | **F** | **F F** |

**Operation:**

| **Exceptions:** | Division Exception | ■ • |  |
| --- | --- | --- | --- |
| **Addressing** |  | ***■ . ■■ ■*** |  |
| **Mode** | **Syntax** | **Instruction Format** | **’ >** |

R: RX: IM: DA:

X: SX: RA: SR: BX:

IR:

DIVU HL.R

DIVU HL.RX

DIVU HL,n

DIVU HL,(addr)

DIVU HL,(XX +dd) DIVU HL,(XY + d) DIVU HL,<addr> DIVU HL,(SP + dd) DIVU HL,(XXA + XXB) DIVU HL,(HL)

**DIVU**

Divide Unsigned (Byte)

src = R, RX, IM, DA, X, SX, RA, SR, BX

**DIVU** [HL,]src

A \*- HL - src ..

L remainder . •,

The contents of the HL register (dividend) are divided by the source operand (divisor) and the quotient is stored in the accumulator; the remainder is stored in the L register. The contents of the source and the H register are not affected. Both operands are treated as unsigned, binary integers. ' ,

There are three possible outcomes of the DIVU instruction, depending on the division and the resulting quotient:

CASE 1: If the quotient is less than 28, then the quotient is left in the accumulator, the Overflow and Sign flags are cleared to 0 and the Zero flag is set according to the value of the quotient. ..

***■ f ■ . • . .***

CASE 2: If the divisor is zero, the accumulator remains unchanged, the Zero and Overflow flags are set to 1 and the Sign flag is cleared to 0. Then the Division Exception trap is taken.

CASE 3: If the quotient is greater than or equal to 28, the accumulator remains un­changed, the Overflow flag is set to 1, and the Sign and Zero flags are cleared to 0. Then the Division Exception trap is taken. . .

**Flags: S:** Cleared

**Z:** Set if the quotient or divisor is zero; cleared otherwise

**H:** Unaffected . .

**V:** Set if the divisor is zero or if the computed quotient is greater than or equal to 28; cleared otherwise

**N:** Unaffected

**C:** Unaffected

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **101** | **101** | **11** | **r** | **101** |  |  |  |  | • | • |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **11** | **rx** | **101** |  | **\*** | **•** |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **111** | **101** | **n** |  |  |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **111** | **101** | **addiflow)** | **addrfhigh)** | **\*** |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **XX** | **101** | **d(low)** | **d(hlgh)** | **• X** |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **11** | **110** | **101** | **d** |  | **•** |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **000** | **101** | **dlsp(low)** | **dlspfhigh)** |  |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **000** | **101** | **d(low)** | **d(high)** |  |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **bx** | **101** |  |  |  |
| **11** | **101** | **101** | **11** | **110** | **101** |  |  |  |  |  |  |

Ф: 0 for IX, 1 for IY

Field Encodings:

ix: 100 for high byte, 101 for low byte ,

xx: 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Example:**

DIVU HL,C

Before instruction execution:

• •

. After instruction execution:

| **AF:** | **5 5** | | **szxhxvnc** | **AF:** | **8 0** | **OOxhxOnc** |
| --- | --- | --- | --- | --- | --- | --- |
| **C:** |  |  | **0 2** | **C:** |  | **0 2** |
| **HL** | **0** | **1** | **0 1** | **HL** | **0 1** | **0 1** |

**DIVUW** [DEHL,]src

src = R, IM, DA, X, RA

HL \*- DEHL + src

**Operation:**

DE remainder

The contents of the DE and HL registers (with the most significant bits of the dividend in the DE register) are divided by the source operand (divisor) and the quotient is stored in the HL register and the remainder in the DE register. The contents of the source are unaffected. Both operands are treated as unsigned, binary integers.

There are three possible outcomes of the DIVUW instruction, depending on the division and the resulting quotient:

A

1. If the quotient is less than 216, then the quotient is left in the HL register and the remainder is left in the DE register, the Overflow and Sign flags are cleared to 0 and the Zero flag is set according to the value of the quotient.
2. If the divisor is zero, the DE and HL registers remain unchanged, the Zero and Overflow flags are set to 1, and the Sign flag is cleared to 0. Then the Division Exception trap is taken.
3. If the quotient is greater than 216-1, then the DE and HL registers remain un­changed, the Overflow flag is set to 1, and the Zero and Sign flags are cleared to 0 Then the Division Exception trap is taken.

**Flags: S:** Cleared

**Z:** Set if the quotient or divisor is zero; cleared otherwise

**H:** Unaffected

**V:** Set if the divisor is zero or if the computed quotient is greater than or equal to 216- cleared otherwise

**N:** Unaffected

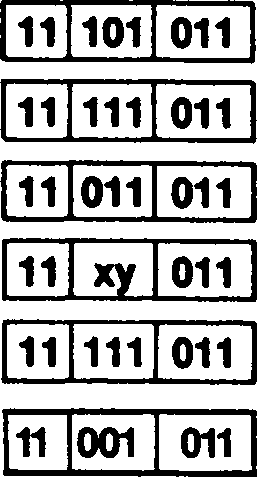
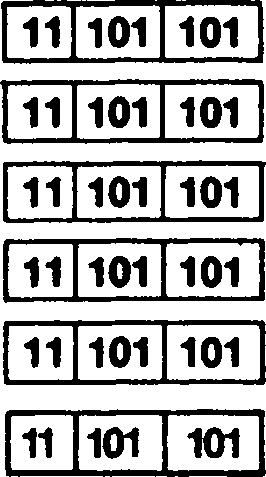
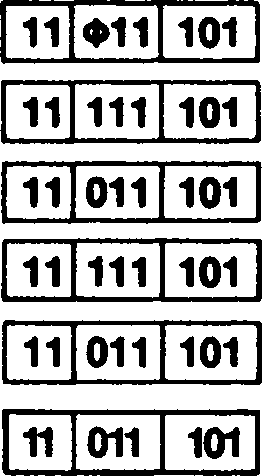
**C:** Unaffected

**Exceptions:** Division Exception

Addressing  
Mode

Syntax

Instniction Format



n(low) | n(hlqh) addr(low)~l addr(hlqh)~] dlsp(low) ~| |

dlspQow) | di»p(hlgh)~|

DIVUW DEHL.XY

**IM:** DIVUW DEHL,nn

**DA:** DIVUW DEHL,(addr)

**X:** DIVUW DEHL,(XY + dd)

**RA:** DIVUW DEHL, <addr>

**IR:** DIVUW DEHL,(HL)

0 for IX, 1 for IY

Field Encodings: ф rr «У

001 for BC, 011 for DE, 101 for HL, 111 for SP

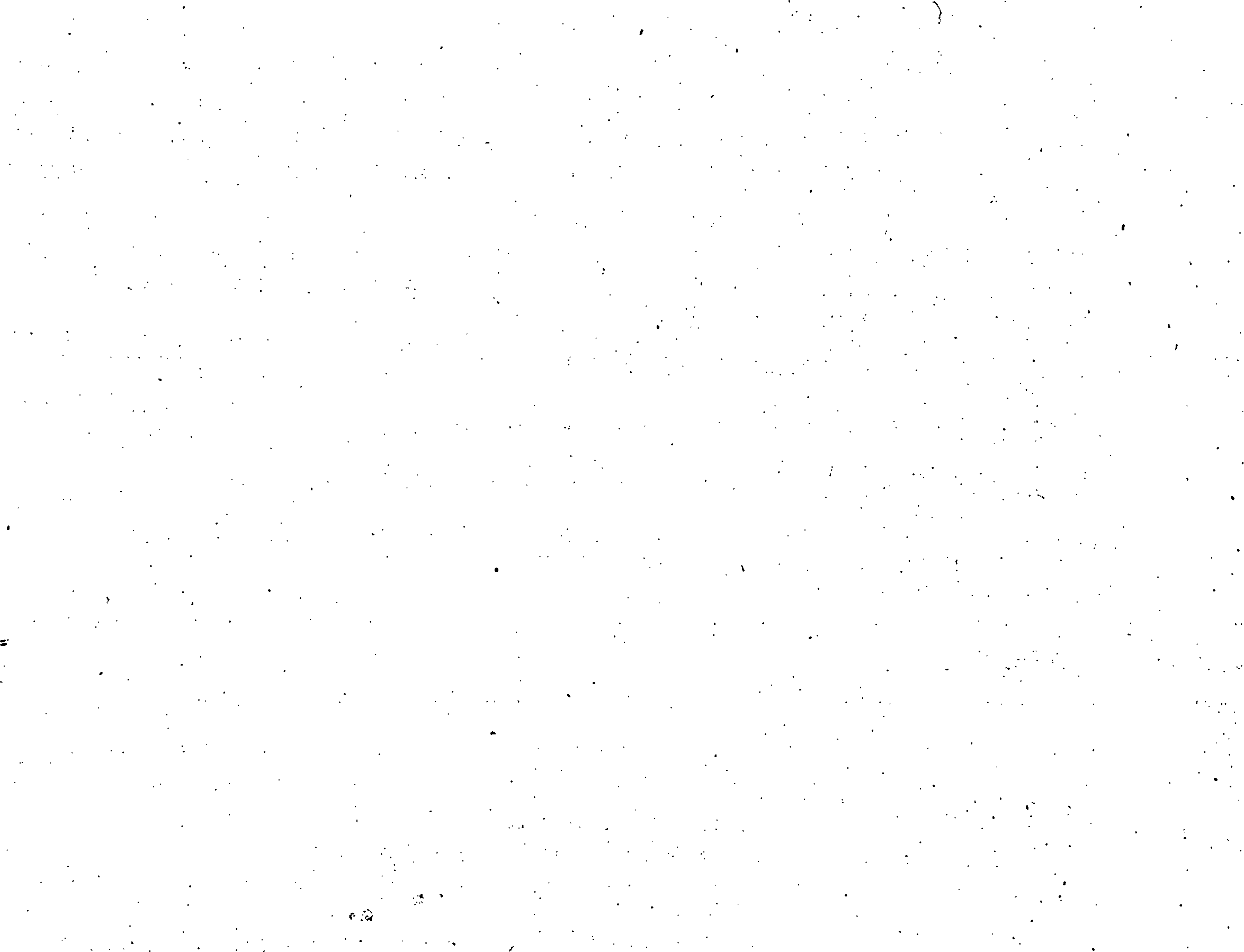
001 for (IX + dd), 011 for (IY + dd)

**Example:** DIVUW DEHL.6

l

Before instruction execution: After instruction execution:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **F:** | | | **szxhxvnc** | **F:** | | | **OOxhxOnc** |
| **DE** | **0** | **0** | **0 0** | **DE** | **0** | **0** | **0 4** |
| **HL** | **0** | **0** | **2 2** | **HL** | **•** | **0** | **0 5** |



**DIVW** [DEHLJsrc

src = R, IM, DA, X, RA  
*ъ ■*

HL \*- DEHL - src

**Operation:**

DE remainder

The contents of the DE and HL registers (with the DE register containing the most signifi­cant bits of the dividend) are divided by the source operand (divisor) and the quotient is stored in the HL register. The contents of the source are unaffected. Both operands are treated as signed, twos-complement integers and division is performed so that the re­mainder is of the same sign as the dividend.

There are three possible outcomes of the DIVW instruction, depending on the division and the resulting quotient: ;

1. If the quotient is within the range - 215 to 215 -1 inclusive, then the quotient is '

left in the HL register and the remainder is left in the DE register, the Overflow flag is cleared to 0, and the Sign and Zero flags are set according to the value of the quotient.

1. If the divisor is zero, the DE and HL registers remain unchanged, the Zero and Overflow flags are set to 1, and the Sign flag is cleared to 0. Then the Division Exception trap is taken.
2. If the quotient is outside the range - 215 to 215 -1, the DE and HL registers re-

• main unchanged, the Overflow flag is set to 1, and the Sign and Zero flags are cleared to

0. Then the Division Exception trap is taken. \*

, I • - .

**Flags: S:** Cleared if V flag is set; else set if the quotient is negative, cleared otherwise

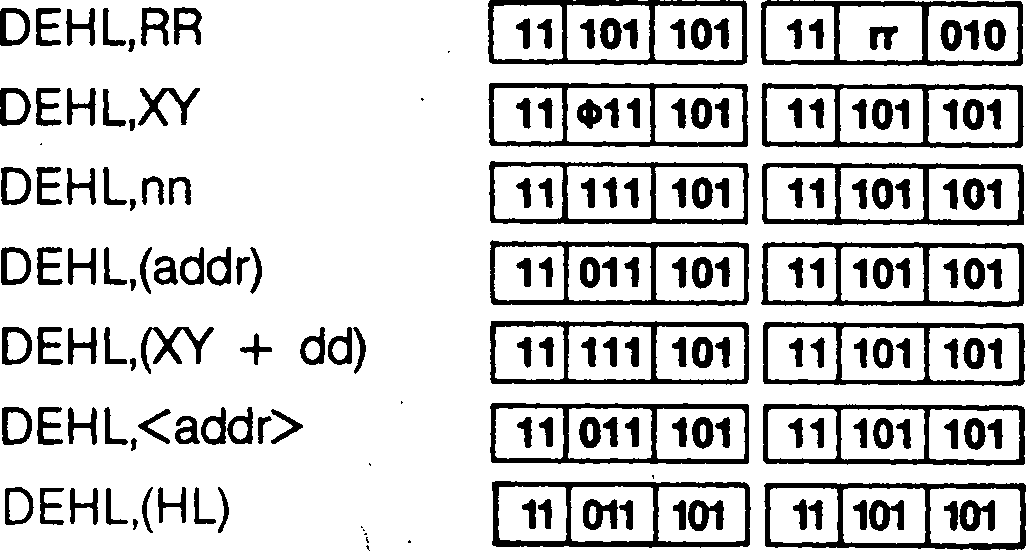
**. Z:** Set if the quotient or divisor is zero; cleared otherwise -

**H:** Unaffected

V: Set if the divisor is zero or if the computed quotient lies outside the range from - 215 to 215-1; cleared otherwise .

**. N:** Unaffected

**C:** Unaffected



**Exceptions**

Division Exception

Addressing  
Mode

Syntax

Instruction Format

R:

IM:

DA:

RA:

IR:

DIVW

DIVW

11 101 010

DIVW

11 111 010

DIVW

11 011 010

DIVW

11 xy 010

DI VW

11 111 010

DI VW

11 001 010

n(low)

n(hlgh)

addr(low)

d(low)

dlspflow)

addr(hlgh)

d(high)

dlsp(hlgh)

Field Encodings: ф : о for ix, 1 for iy

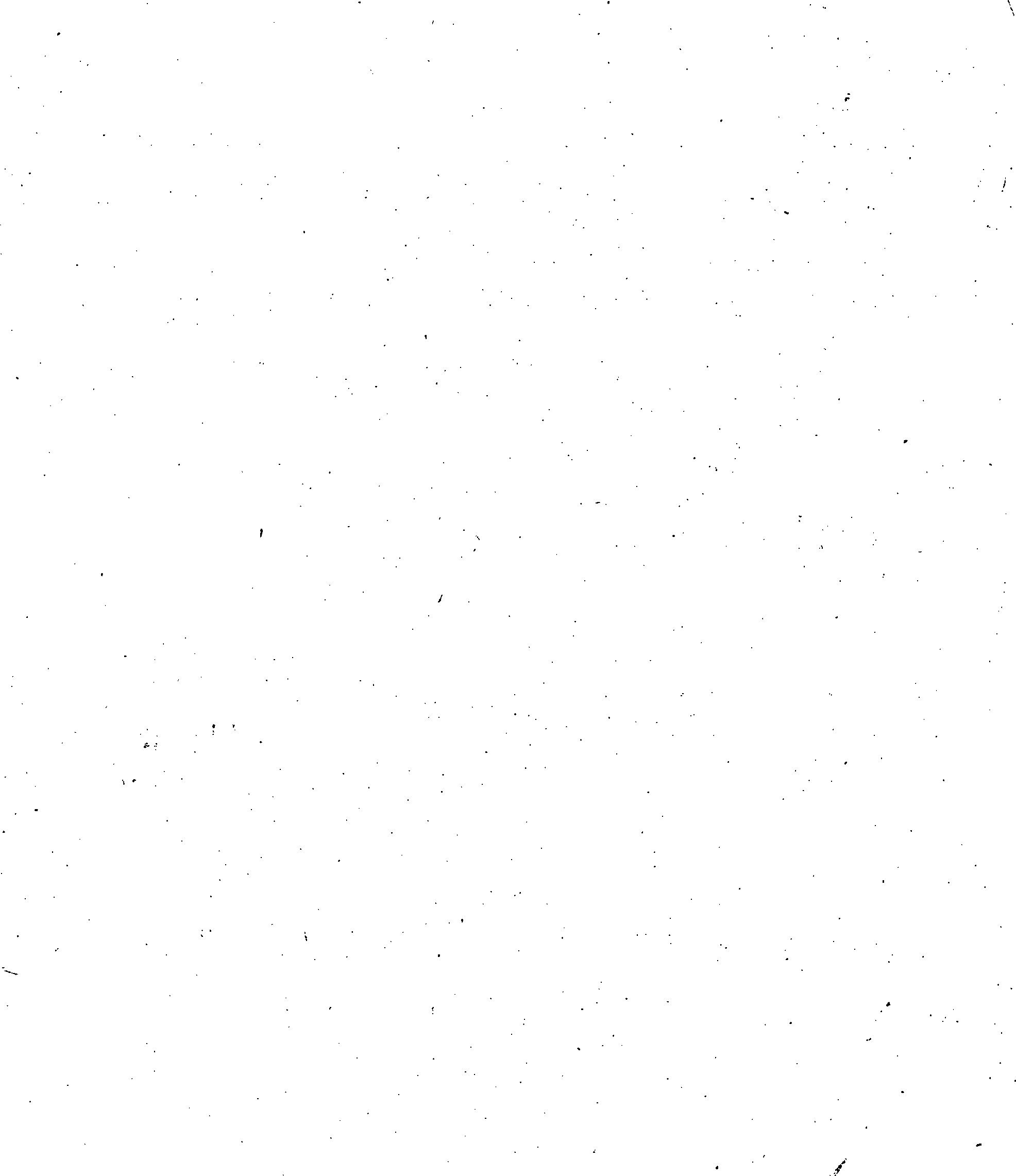
’ rr: 001 for BC, 011 for DE, 101 for HL, 111 for SP

xy: 001 for (IX + dd), 011 for (IY + dd)

**Example:** DIVW DEHL.6

; ' Before instruction execution: After instruction execution:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F:** |  | **szxhxvnc** | **F:** |  | **OOxhxOnc** |
| **DE** | **0 0** | **0 0** | **DE** | **0 0** | **0 4** |
| **HL** | **0 0** | **2 2** | **HL** | **0 0** | **0 5** |



. • /

**DJNZ**

**Decrement and Jump if Non-Zero**

**DJNZ** dst dst = RA

**Operation:** B\*~B- 1

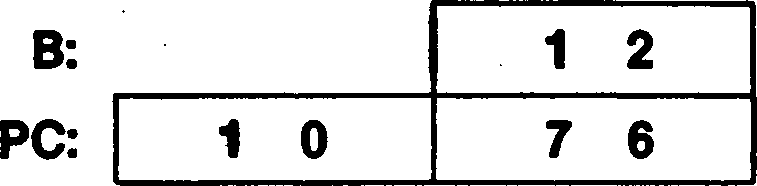
if В =# 0 then PC dst

The В register is decremented by one. If the result is non-zero, then the destination ad­dress is calculated and then loaded into the Program Counter (PC). Control then passes to the instruction whose address is pointed to by the PC. When the В register reaches zero, control falls through to the instruction following DJNZ. This instruction provides a simple method of loop control. .

The destination address is calculated using Relative addressing. The displacement in the instruction is added to the PC; the PC value used is the address of the instruction following the DJNZ instruction. The 8-bit displacement is treated as a signed, twos-complement integer. Thus the branching range from the location of this instruction is -126 to +129 bytes. ;

**Flags:** No flags affected

**Exceptions:** None



Addressing  
Mode

RA:

Syntax

DJNZ addr

Instruction Format

00 010 000 disp

**Example:**

DJNZ 1050H

After instruction execution:

Before instruction execution:

B:

PC:

El

**Enable Interrupt**

**El** mask

Addressing  
Mode

Syntax

Instruction Format

Mask = Hex value between 0 and 7Fh

If mask(i) = 1 then MSR(i) \*-1

**Operation:**

The designated control bits in the Master Status register (MSR) are set to 1, thus enabl­ing interrupts on these inputs; all other interrupt enables in the MSR are unaffected. Note that during the execution of this instruction and the following instruction, all maskable interrupts (whether previously enabled or not) are automatically disabled for the duration of these two instructions. .

Any combination of interrupt enables in the MSR can be specified. The seven bits in the mask field in the instruction correspond to the seven interrupt enable bits in the MSR, mask bit i corresponding to MSR bit i. If no mask is present, all interrupts are enabled.

**Flags:** No flags affected

**Exceptions:** Privileged Instruction

Before instruction execution:

After instruction execution:

11 111 011

El mask

mask

Mask = byte specifying which interrupts to disable: mask(i) corresponds to interrupt source i; mask(7) must be zero.

El 49H

**Example**

MSR

MSR:

**EX**

Exchange Accumulator/Flag with Alternate Bank

**EX** AF.AF'

\* ■ \*

**Operation:** AF\*\*AF'

The control bit mapping the accumulator arid flag registers into the primary bank or the auxiliary bank is complemented, thus effectively exchanging the accumulator and flag registers between the two banks. ... ■'>

**Flags:** Loaded from F'

**Exceptions:** None

Addressing  
Mode Syntax

Instruction Format

| **00** | **001** | **000** |
| --- | --- | --- |

**Example:**

EX AF.AF'

EX AF.AF'

Before instruction execution: After instruction execution:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **AF:** | **2 3 F 3** | **AF:** | **1 0** | **В** | **0** | *1* |
| **AF':** | **1 0 В 0** | **AF':** | **2 3** | **F** | **3** | *. f*  • t . »  *• t* |

**EX**

Exchange Addressing Register with Top of Stack

**EX** (SP),dst

dst = HL, IX, IY .

4

**Operation:**

(SP) \*\* dst

I

The contents of the destination register are exchanged with the contents of the top of stack. That is, the low-order byte contained in the register is exchanged with the con­tents of the memory address specified by the Stack Pointer (SP), and the high-order byte of the register is exchanged with the contents of the next highest memory address (SP+ 1).

**Flags:** No flags affected

**Exceptions: ,** None

Addressing  
Mode Syntax

Instruction Format

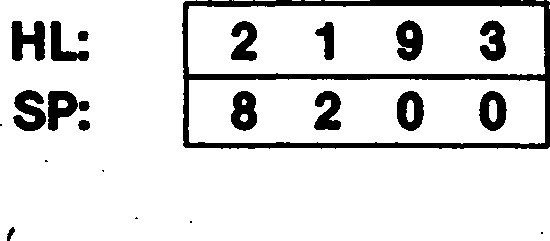
EX (SP),HL

EX (SP),XY

|  |  |  |
| --- | --- | --- |
| **11** | **100** | **011** |

|  |  |  |
| --- | --- | --- |
| **11** | **Ф11** | **101** |

|  |  |  |
| --- | --- | --- |
| **11** | **100** | **011** |



Before instruction execution:

Data memory:

8200:

8201:

2 A

В 3

HL: В 3 2 A

SP: 8 2 0 0

Field Encoding: ф : о for ix, 1 for iy

**Example:** EX (SP),HL

After instruction execution:

Data memory:

8200: 9 3

8201: 2 1

**EX** H.L

**Operation:**

H\*\*L

**Exchange H and L**

The contents of the H and L registers are exchanged.

**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode Syntax

Instruction Format

EX H.L

|  |  |  |
| --- | --- | --- |
| **11** | **101** | **101** |

|  |  |  |
| --- | --- | --- |
| **11** | **101** | **111** |

**Example:**

EX H.L

Before instruction execution:

After instruction execution:

HL: 12 3 4

HL: 3 4 12

**EX** H,L

**Operation:**

H \*\* L

The contents of the H and L registers are exchanged.

**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode

Syntax

EX H,L

Instruction Format

11 101 101 11 101 111

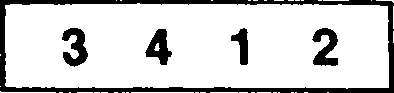
**Example:** EX H,L

Before instruction execution:

HL:

After instruction execution:

HL:



**EX**

**Exchange HL with Addressing Register**

**EX** src,HL src = DE, IX, IY

**Operation:**

src \*\* HL

The contents of the HL register are exchanged with the contents of the source.

**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode Syntax

EX DE,HL

EX XY.HL

Instruction Format

11 101 011

11 Ф11 101 11 101 011

Field Encoding: <t>: о for ix, 1 for iy

**Example:**

EX DE,HL

Before instruction execution:

After instruction execution:

DE:

HL:

DE:

HL:

**EX** A,src

**Operation:**

src \*\* A

**EX**

**Exchange with Accumulator**

src = R, RX, IR, DA, X, SX, RA, SR, BX

The contents of the accumulator are exchanged with the contents of the source.

**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

**R:** EX A,R

**RX: .** EX A,RX

**IR:** EX A,(HL)

**DA** EX A,(addr)

**X:** EX A,(XX + dd)

**SX:** EX A,(XY + d)

**RA** EX A,<addr>

**SR:** EX A,(SP + dd)

**BX:** EX A,(XXA + XXB)

| **11** | **101** | **101** | **00** | **r** | **111** |  | | |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **rx** | **111** |  |
| **11** | **101** | **101** | **00** | **110** | **111** |  | | |  |
| **11** | **011** | **101** | **11** | **101** | **101** | **00** | **111** | **111** | **addr(low)** |
| **11** | **111** | **101** | **11** | **101** | **101** | **00** | **XX** | **111** | **d(low)** |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **110** | **111** | **d** |
| **11** | **111** | **101** | **11** | **101** | **101** | **00** | **000** | **111** | **disp(low)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **00** | **000** | **111** | **d(low)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **00** | **bx** | **111** |  |

addr(high) d(high)

disp(high) d(high)

Field Encodings: ф : rx:

xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX 4- dd), 010 for (IY 4- dd), 011 for (HL 4- dd)

001 for (HL 4- IX), 010 for (HL 4- IY), 011 for (IX 4- IY)

**Example:**

EX A, В

Before instruction execution:

After instruction execution:

A

B:

A 8 2

B: 0 3

EXTS

**Extend Sign (Byte)**

**EXTS** [A]

**Operation:** L \*- A

If A(7) = 0, then H \*- 00 else H \*- FF

The contents of the accumulator, considered as a signed, twos-complement integer, are sign-extended to 16 bits and the result is stored in the HL register. The contents of the accumulator are unaffected. This instruction is useful for conversion of short signed operands to longer signed operands.

**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode

Syntax

EXTS A

Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **100** | **100** |
| --- | --- | --- |

**Example:** EXTS A

| **A:** | | **8 2** | |
| --- | --- | --- | --- |
| **HL:** | **5 5** | **5** | **СЛ** |

Before instruction execution:

| **A:** | | | • | **2** |
| --- | --- | --- | --- | --- |
| **HL:** | **F** | **F** | **a** | **2** |

After instruction execution:

EXTS

**Extend Sign (Word)**

**EXTS** HL

If H(7) = 0, then DE \*- 0000 else DE \*- FFFF

**Operation:**

The contents of the HL register, considered as a signed, twos-complement integer, are sign-extended to 32 bits and the result is stored in the DE and HL registers, with the DE register containing the most significant bits. This instruction is useful for conversion of signed operands to larger signed operands.

**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode .

Syntax

EXTS HL

Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **101** | **100** |
| --- | --- | --- |

**Example:** EXTS HL

| **DE:** | **0** | **3** | **2** | **F** |
| --- | --- | --- | --- | --- |
| **HL:** | **E** | **F** | **3** | **0** |

Before instruction execution:

| **DE:** | **F** | **F** | **F** | **F** |
| --- | --- | --- | --- | --- |
| **HL:** | **E** | **F** | **3** | **0** |

After instruction execution:

**EXX**

**Exchange Byte/Word Registers with Alternate Bank**

**EXX**

**Operation:**

BC \*\* BC' DE\*\* DE' HL\*\* HL'

The control bit mapping the byte/word registers into the primary or auxiliary bank of the CPU registers is complemented, thus effectively exchanging the В, C, D, E, H, and L registers between the two banks.

**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode

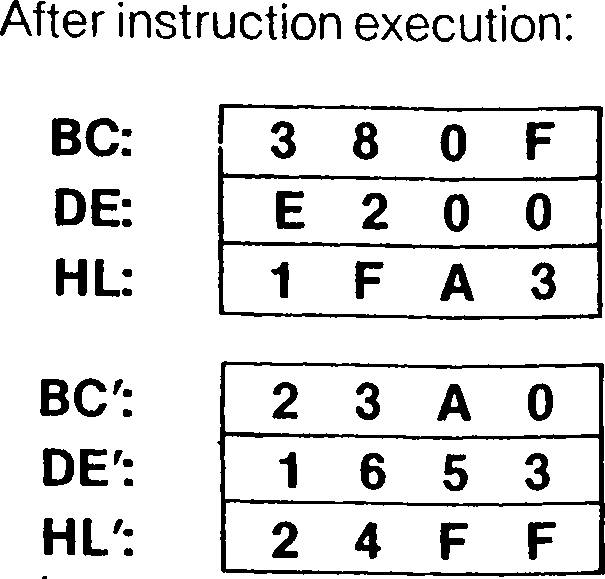
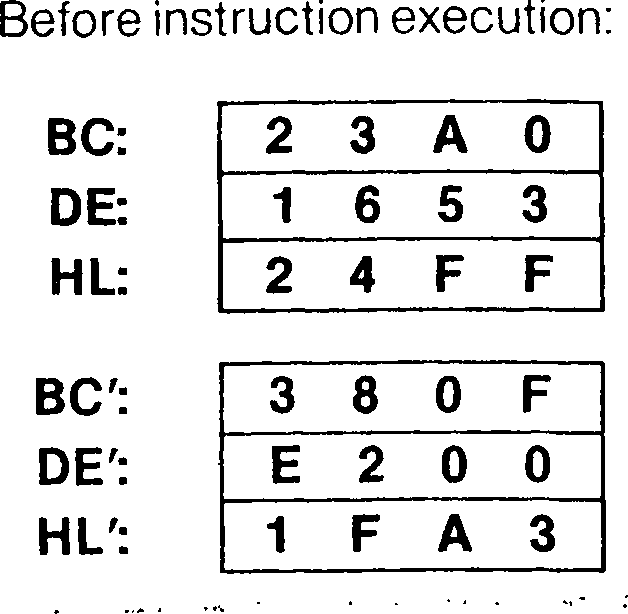
Syntax

EXX

Instruction Format

| **11** | **011** | **001** |
| --- | --- | --- |

**Example:** EXX



HALT

**HALT**

**HALT**

**Operation:** CPU Halts

The CPU operation is suspended until an interrupt or reset request is received. This in­struction is used to synchronize the Z280 MPU with external events, preserving its state until an interrupt or reset request is accepted. After an interrupt is serviced, the instruc­tion following HALT is executed. While halted, memory refresh cycles still occur, and bus requests are honored.

For the Z80 Bus configuration of the Z280 MPU, the HALT signal is asserted when the Halt instruction is executed and remains asserted until an interrupt or reset request is accepted. For the Z-BUS configurations of the Z280 MPU, a special Halt bus transaction is performed when the halt instruction is executed.

If the Breakpoint-on-Halt control bit in the Master Status register is set to 1, the Halt instruction is not executed, and Breakpoint-on-Halt trap is taken instead. '

**Flags:** No flags affected

Breakpoint, Privileged Instruction

**Exceptions:**

Addressing  
Mode

Syntax

HALT

**Instruction Format**

01 110 110

IM

**Interrupt Mode Select**

**IM** p

P = o, 1,2, 3

**Operation:**

Interrupt Mode \*- p

The interrupt mode of operation is set to one of four modes (see Chapter 6 for a descrip­tion of the various modes for responding to interrupts). The current interrupt mode can be read from the Interrupt Status register.

**Flags:**

No flags affected

**Exceptions:**

Privileged Instruction

Addressing  
Mode

Syntax

instruction Format

IM p

11 101 101 01 t 110

**mode encoding**

**0 1**

**2 3**

000

010

011

001

**Example:**

IM 3

Before instruction execution:

After instruction execution:

■ Interrupt Status register: -

interrupt Status register:

**IN** dst,(С)

dst = R, RX, DA, X, RA, SR, BX

**Operation:**

dst \*- (C)

The byte of data from the selected peripheral is loaded into the destination. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines A8-A-|5 and the contents of the I/O Page register are placed on address lines А16-А2з. The byte of data from ■ the peripheral is then loaded into the destination.

**Flags: S:** Set if the input data is negative; cleared otherwise

**Z:** Set if the input data is zero; cleared otherwise

**H:** Cleared

**V:** Set if the input data has even parity; cleared otherwise

**N:** Cleared .... .... .

**C:** Unaffected ’

**Exceptions:** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

Syntax

Instruction Format

| **R:** | IN | R,(C) |
| --- | --- | --- |
| **RX:** | IN | RX,(C) |
| **DA:** | IN | (addr),(C) |
| **X:** | IN | (XX + dd),(C) |
| **RA:** | IN | <addr>,(C) |
| **SR:** | IN | (SP + dd),(C) |
| **BX:** | IN | (XXA + XXB),(C) |

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **r** | **000** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **rx** | **000** |
| --- | --- | --- |

| **11** | **011** | **101** | **11** | **101** | **101** | **01** | **111** | **000** | **addr(low)** | **addr(high)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **111** | **101** | **11** | **101** | **101** | **01** | **XX** | **000** | **d(low)** | **d(high)** |
| **11** | **111** | **101** | **11** | **101** | **101** | **01** | **000** | **000** | **dispflow)** | **disp(high)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **01** | **000** | **000** | **d(low)** | **d(high)** |

11 Ф11 101 11 101 101 01 rx 000

11 011 101 11 101 101 01 bx 000

| **11** | **011** | **101** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **bx** | **000** |
| --- | --- | --- |

Field Encodings: ф :

rx:

xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

001 for (HL + IX), 010 for (HL 4- IY), 011 for (IX + IY)

**Example:** IN L,(C)

Before instruction execution: After instruction execution:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **F:** | | | **szxhxvnc** | **F:** | | | **00x0x00c** |
| **BC:** | **1** | **6** | **5 0** | **BC:** | **1** | **6** | **5 0** |
| **HL:** | **0** | **0** | **2 3** | **HL:** | **0** | **0** | **7 6** |

I/O Page register:

Byte 76h available at I/O port 111 650h

IN

**Input Accumulator**

**IN** A,(n)

**Operation:**

A\*-(n)

The byte of data from the selected peripheral is loaded into the accumulator. During the I/O transaction, the 8-bit peripheral address from the instruction is placed on the low byte of the address bus, the contents of the accumulator are placed on address lines A8-A15 and the contents of the I/O Page register are placed on address lines A16-A23. The byte of data from the selected port is written into the accumulator.

**Flags:** No flags affected

**Exceptions: .** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

Syntax

IN A,(n)

Instruction Format

| **11** | **011** | **011** |
| --- | --- | --- |

**Example:** IN A,(66H)

Before instruction execution:

After instruction execution:

A: 4 2

A: F D

I/O Page register:

Byte FDh available at I/O port 11 4266h



**Operation:**

dst \*- dst + 1

The destination operand is incremented by one and the sum is stored in the destination. Twos-complement addition is performed.

**Flags:**

**S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there is a carry from bit 3 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, if the destination was 7Fh; cleared otherwise

**N:** Cleared

**C:** Unaffected

**Exceptions:** None

Addressing  
Mode

R:  
RX:

IR:

DA:

X: SX: RA: SR: BX:

Syntax

INC R INC RX INC (HL) INC (addr)

INC (XX 4- dd)

INC (XY 4- d) INC <addr> INC (SP 4- dd) INC (XXA 4- XXB)

Instruction Format

| **11** | **011** | **101** | **00** | **111** | **100** | **addr(low)** |
| --- | --- | --- | --- | --- | --- | --- |
| **11** | **111** | **101** | **00** | **XX** | **100** | **d(low)** |
| **11** | **Ф11** | **101** | **00** | **110** | **100** | **d** |

001 r 1100

11 |Ф111101 ] I oo| rx 1100

ool11o|100

addr(hlgh) dfhlgh)

| **11** | **111** | **101** | **00** | **000** | **100** | **disp(low)** |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
| **11** | **011** | **101** | **00** | **000** | **100** | **d(low)** |
| **11** | **011** | **101** | **00** | **bx** | **100** |  |

dispQilgh) d(high)

Field Encodings: ф :

rx:

xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX 4- dd), 010 for (IY + dd), 011 for (HL 4- dd)

001 for (HL 4- IX), 010 for (HL 4- IY), 011 for (IX 4- IY)

**Example:**

INC (HL)

Before instruction execution:

F: HL:

2

szxhxvnc  
5 4

| After instruction execution: | | |
| --- | --- | --- |
| **F:** | | **10x0x00c** |
| **HL:** | **2 4** | **5 4** |

Data memory:

2454: 8 8

Data memory:

2454: 8 9

dst = R, RX, IR, DA, X, SX, RA, SR, BX

**INC** dst

**Operation:**

**INC[W1** dst or

**INCW** dst

dst \*- dst + 1

dst = R

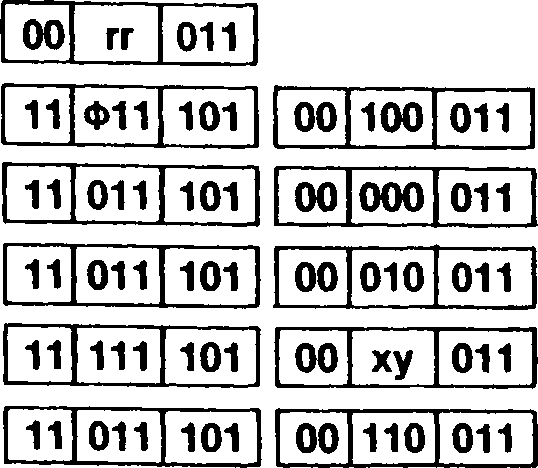
dst = IR, DA, X, RA

The destination operand is incremented by one. Twos-complement addition is performed.

**Flags:**

No flags affected

**Exceptions:** None



Addressing  
Mode Syntax

Instruction Format

R:

IR:

DA:

X

RA:

INCW RR

INCW XY

INCW (HL)

INCW (addr) INCW (XY + dd) INCW <addr>

addr(low)

d(low)

dispflow)

addr(hlgh)

dfhigh)

dlspfhigh)

Field Encodings: ф:

xy:

0 for IX, 1 for IY

000 for BC, 010 for DE, 100 for HL, 110 for SP

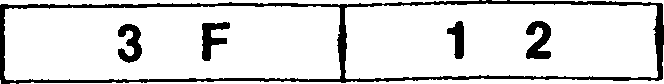
000 for (IX + dd), 010 for (IY + dd)

**Example:**

INCW BC

Before instruction execution:

After instruction execution:



BC:

BC:

IND

**Input and Decrement (Byte, Word)**

**IND INDW**

(HL) -(C)

**Operation:**

B-B - 1

HL — AUTODECREMENT HL (by one if byte, by two if word)

This instruction is used for block input of strings of data. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines Aq-A^, and the contents of the I/O Page register are placed on address lines А-16-А23. The byte or word of data from the selected peripheral is then loaded into the memory location addressed by the HL register. The HL register is then decremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next destination for the input. The В register, used as a counter, is then decremented by one.

**Flags: S:** Unaffected

**Z:** Set if the result of decrementing В is zero; cleared otherwise

**H:** Unaffected

**V:** Unaffected

**N:** Set

**C:** Unaffected

**Exceptions:** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

Syntax

IND

INDW

Instruction Format

11 101

| **10** | **101** | **010** |
| --- | --- | --- |

11 101 101

| **10** | **001** | **010** |
| --- | --- | --- |

101

**Example: •** INDW

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **F:** | | **szxhxvnc** | **F:** | | | **sOxhxvlc** |
| **BC:** | **1 5** | **6 4** |  | **BC:** | **1 4** | **6 4** |
| **HL:** | **5 0** | **0 2** |  | **HL:** | **5 0** | **0 0** |

Before instruction execution:

After instruction execution:

I/O Page register:

Data memory:

5002: О 7

5003: a D

Word 8D07H available at I/O port 331564H

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used

INDR

**Input, Decrement and Repeat (Byte, Word)**

**INDR INDRW**

Repeat until В = 0: (HL) \*- (C)

**Operation:**

В В — 1

HL AUTODECREMENT HL (by one if byte, by two if word)

This instruction is used for block input of strings of data. The string of data from the selected peripheral is loaded into memory at consecutive addresses, starting with the location addressed by the HL register and decreasing. During the I/O transactions, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines Aq-A-is, and the contents of the I/O Page register are placed on address lines A16-A23. The byte or word of data from the selected peripheral is loaded into the memory location addressed by the HL register. The HL register is then decremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next destination for the input. The В register, used as a counter, is then decremented by one. If the result of decrementing the В register is zero, the instruction is terminated, otherwise the input sequence is repeated. Note that if the В register contains 0 at the start of the execution of this in­struction, 256 bytes are input.

This instruction can be interrupted after each execution of the basic operation. The Prograrr Counter value of the start of this instruction is saved before the interrupt request is accepted so that the instruction can be properly resumed.

**Flags: S:** Unaffected

**Z:** Set

**H:** Unaffected

**V:** Unaffected

**N:** Set

**C:** Unaffected

Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

**Exceptions:**

Addressing  
Mode

Syntax

INDR

INDRW

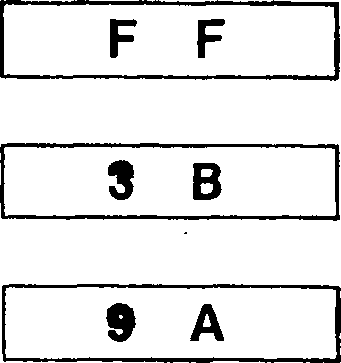
Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **111** | **010** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **011** | **010** |
| --- | --- | --- |



**Example:**

INDR

Before instruction execution:

I/O Page register:

5216:

5217:

5218:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F:** |  | **szxhxvnc** | **F:** |  | **slxhxvlc** |
| **BC:** | **0 3** | **4 6** | **BC:** | **0 0** | **4 6** |
| **HL:** | **5 2** | **1 8** | **HL:** | **5 2** | **1 5** |

After instruction execution:

Data memory:

Byte 9AH available at I/O port 170346ц, then byte 3BH available at I/O port 170246ц, then byte FFH available at I/O port 170146ц.

INI

**Input and Increment (Byte, Word)**

**INI INIW**

(HL)-(C)

**Operation:**

B — B - 1

HL — AUTOINCREMENT HL (by one if byte, by two if word)

This instruction is used for block input of strings of data. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines A8-A15, and the contents of the I/O Page register are placed on address lines A16-A23. The byte or word of data from the selected peripheral is loaded into the memory location addressed by the HL register. The HL register is then incremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next destination for the input. The В register, used as a counter, is then decremented by one.

**Flags: S:** Unaffected

**Z:** Set if the result of decrementing В is zero; cleared otherwise

**H:** Unaffected

**V:** Unaffected

**N:** Set

**C:** Unaffected

**Exceptions:** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

Syntax

INI

INIW

Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **100** | **010** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **000** | **010** |
| --- | --- | --- |

**Example:** INI

|  | **szxhxvnc** |
| --- | --- |
| **1 5** | **« 4** |
| **5 0** | **0 2** |

Before instruction execution:

BC:

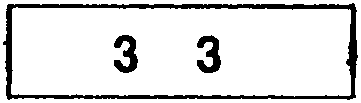
HL:

|  | **sOxhxvlc** |
| --- | --- |
| **1 4** | **6 4** |
| **5 0** | **0 3** |

After instruction execution:

BC:

HL:

I/O Page register:

Data memory:

5002: 7 A

Byte 7AH available at

I/O port 331564H

**INIR**

Input, Increment and Repeat

**Operation:**

**Flags:**

**Exceptions:**

Addressing  
Mode

**INIR INIRW**

Repeat until В = 0: (HL) \*- (C)

B\*-B - 1

HL AUTOINCREMENT HL (by one if byte, by two if word)

This instruction is used for block input of strings of data. The string of data from the selected peripheral is loaded into memory at consecutive addresses, starting with the location addressed by the HL register and increasing. During the I/O transactions, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines Ag-A-is, and the contents of the I/O Page register are placed on address lines A16-A23. The byte or word of data from the selected peripheral is loaded into the memory location addressed by the HL register. The HL register is then incremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next destination for the input. The В register, used as a counter, is then decremented by one. If the result of decrementing the В register is zero, the instruction is terminated, otherwise the input sequence is repeated. Note that if the В register contains 0 at the start of the execution of this in­struction, 256 bytes are input.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value at the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

**S:** Unaffected

**Z:** Set

**H:** Unaffected

**V:** Unaffected

**N:** Set

**C:** Unaffected

Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Syntax

INIR

INIRW

Instruction Format

11 101 101

11 101 101

10 110 010

10 010 010

**Example:**

INIRW

Before instruction execution:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F:** | | **szxhxvnc** | **F:** | | **slxhxvlc** |
| **BC:** | **0 2** | **5 5** | **BC:** | **0 0** | **5 5** |
| **HL** | **4 0** | **0 2** | **HL:** | **4 0** | **0 6** |

After instruction execution:

Data memory:

I/O Page register:

Word 66D7h available at  
l/Oport310255Hthen word A8FFh available  
at I/O port 3101 55h-

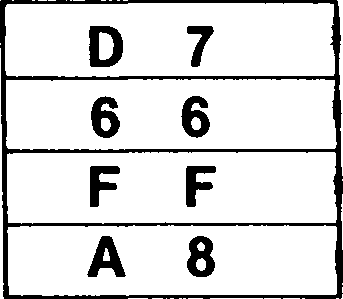
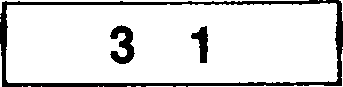
4002:

4003:

4004:

4005:

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.



IN[W]

**Input HL**

**IN[W]** HL,(С)

**Operation:**

HL— (C)

The word of data from the selected peripheral is loaded into the HL register. During the I/O transaction, the 8-bit peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines Aq-A-is and the contents of the I/O Page register are placed on address lines A16-A23. Then one word of data from the selected port is written into the HL register. For 8-bit data buses, the contents of L are undefined for external peripherals.

**Flags:** No flags affected

**Exceptions:** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

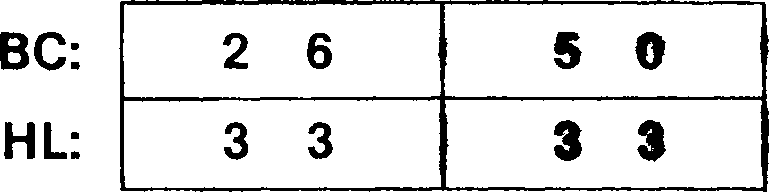
Syntax

IN HL,(C)

Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **110** | **111** |
| --- | --- | --- |



Before instruction execution:

I/O Page register:

**Example:** I NW HL,(C)

After instruction execution:

Word 4D87H available at I/O port 102650H

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

JAF

**Jump On Auxiliary Accumulator/Flag**

**JAF** dst

dst = RA

If auxiliary AF then PC \*- dst

**Operation:**

A conditional jump is performed if the auxiliary Accumulator/Flag registers are in use. If the jump is taken, the Program Counter is loaded with the destination address; otherwise the instruction following the JAF instruction is executed. This instruction employs an 8-bit signed, twos-complement displacement from the Program Counter to permit jumps within the range —125 to +130 bytes from the location of this instruction.

**Flags:**

No flags affected

**Exceptions:**

None

Addressing  
Mode

Syntax

Instruction Format

**RA:** JAF addr

11 011 101

00 101 000

disp

**Example:** JAF 5000H

Before instruction execution:

After instruction execution:

Auxiliary Accumulator/Flag in use

PC: 4 F E 6

PC: 5 0 0 0

**JAR**

**Jump On Auxiliary Register File In Use**

dst = RA

**JAR** dst

If auxiliary file then PC \*- dst

**Operation:**

A conditional jump is performed if the auxiliary register file is in use. If the jump is taken, the Program Counter is loaded with the destination address; otherwise the instruction following the JAR instruction is executed. This instruction employs an 8-bit signed, twos- complement displacement from the Program Counter to permit jumps within the range -125 to +130 bytes from the location of this instruction.

**Flags:**

No flags affected

**Exceptions:**

None

Addressing  
Mode Syntax

Instruction Format

RA:

JAR addr

11 011 101 00 100 000

disp

**Example:**

JAR 42D0H

Before instruction execution:

After instruction execution:

Auxiliary file in use

PC:

PC:

JP

**Jump**

**JP** [cc,]dst

dst = IR, DA, RA

If cc is satisfied then PC \*- dst

**Operation:**

A conditional jump transfers program control to the destination address if the setting of a selected flag satisfies the condition code “cc” specified in the instruction; an uncondi­tional jump always transfers control to the destination address. If the jump is taken, the Program Counter (PC) is loaded with the destination address; otherwise the instruction following the Jump instruction is executed. For the Relative Address mode, the PC value used to calculate the destination address is the address of the next instruction following the Jump instruction; a 16-bit signed twos-complement displacement from the PC per­mits jumps within the range —32764 to +32771 bytes from the location of this instruc­tion.

Each of the Zero, Carry, Sign, and Overflow flags can be individually tested and a jump performed conditionally on the setting of the flag.

When using DA mode with the JP instruction, the operand is not enclosed in paren­theses.

**Flags:**

No flags affected

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

**IR:** JP CC.(HL)

JP (HL)

JP (XY)

**DA:** JP CC.addr

JP addr

**RA:** JP CC,<addr>

JP <addr>

addr(high) addr(high) dispflow) dispflow)

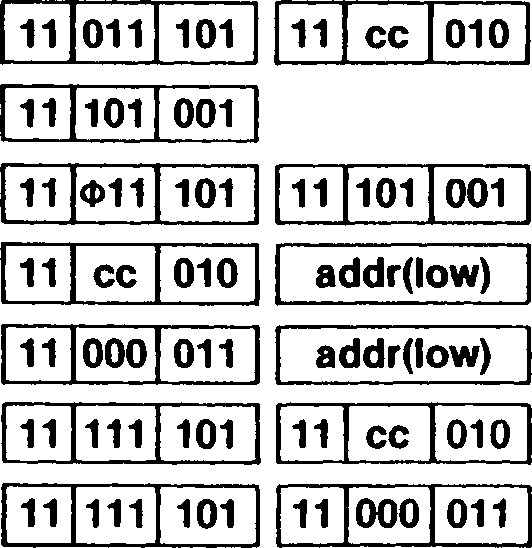
| “unconditional jump”

| “unconditional jump”

“unconditional jump”

dispfhigh)  
dispfhigh)

| “unconditional jump”



Field Encodings: ф :

cc:

0 for IX, 1 for IY

000 for NZ, 001 for Z, 010 for NC, 011 for C, 100 for PO or NV, 101 for PE or V, 110 for P or NS, 111 for M or S

**Example:** JP C.5000H

Before instruction execution:

F:

PC:

szxhxvn!

|  | **szxhxvn1** |
| --- | --- |
| **5 0** | **0 0** |

After instruction execution:

**JR** [cc,]dst

**JR**

Jump Relative

dst = RA

If the cc is satisfied then PC \*- dst

A conditional jump transfers program control to the destination address if the setting of a selected flag satisfies the condition code “cc” specified in the instruction; an uncondi­tional jump always transfers control to the destination address. If the jump is taken, the Program Counter (PC) is loaded with the destination address; otherwise the instruction following the Jump Relative instruction is executed. These instructions employ an 8-bit signed, twos-complement displacement from the PC to permit jumps within the range -126 to +129 bytes from the location of this instruction.

Either the Zero or Carry flag can be tested and a jump performed conditionally on the setting of the flag.

**Flags:**

No flags affected

**Exceptions:**

None

Addressing  
Mode

Syntax

Instruction Format

RA:

JR CC.addr

JR addr

00 cc 000

00 011 000

disp

disp

unconditional jump

Field Encoding: cc: 100 for NZ, 101 for Z, 110 for NC, 111 for C

**Example:**

JR NZ,6000H

Before instruction execution:

After instruction execution:

F:

PC:

sOxhxvnc

**F:**

PC:

sOxhxvnc

LD

**Load Accumulator**

**LD** dst,src dst = R, RX, IR, DA, X, SX, RA, SR, BX

src = A

or

dst - A

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**Operation:**

dst src

The contents of the source are loaded into the destination. The contents of the source are not affected. Special instructions are provided so that the BC and DE registers can also be used in the IR addressing mode.

**Flags:** No flags affected

**Exceptions:** None

**Load into Accumulator**

Addressing  
Mode Syntax

Instruction Format

|  |  |
| --- | --- |
| **R:** | LD A.R |
| **RX:** | LD A,RX |
| **IM:** | LD A,n |
| **IR:** | LD A,(HL) |
|  | LD A,(RR) |
| **DA:** | LD A,(addr) |
| **X:** | LD A,(XX + (  <• |
| **SX:** | LD A,(XY + , |
| **RA:** | LD A,<addr> |
| **SR:** | LD A,(SP + < |
| **BX:** | LD A,(XXA + |

d)

|  |  |  |
| --- | --- | --- |
| **01** | **111** | **г** |

|  |  |  |
| --- | --- | --- |
| **11** | **Ф11** | **101** |

|  |  |  |
| --- | --- | --- |
| **00** | **111** | **110** |

|  |  |  |
| --- | --- | --- |
| **01** | **111** | **110** |

|  |  |  |
| --- | --- | --- |
| **00** | **rra** | **010** |

|  |  |  |
| --- | --- | --- |
| **00** | **111** | **010** |

|  |  |  |
| --- | --- | --- |
| **01** | **111** | **rx** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **111** | **101** | **01** | **111** | **xxa** | **d(low)** | **d(high)** |
| **11** | **Ф11** | **101** | **01** | **111** | **110** | d |  |
| **11** | **111** | **101** | **01** | **111** | **000** | **disp(low)** | **disp(high)** |
| **11** | **011** | **101** | **01** | **111** | **000** | **d(low)** | **d(high)** |

00 111 010 addr(low) addr(high)

11 011 101 01 111 bx

|  |  |  |
| --- | --- | --- |
| **11** | **011** | **101** |

|  |  |  |
| --- | --- | --- |
| **01** | **111** | **bx** |

Load from Accumulator

Addressing  
Mode

Syntax

Instruction Format

|  |  |
| --- | --- |
| **R:** | LD R.A |
| **RX:** | LD RX,A |
| **IR:** | LD (HL),A |
|  | LD (RR),A |
| **DA:** | LD (addr),A |
| **X:** | LD (XX + dd),A |
| **SX:** | LD (XY + d),A |
| **RA:** | • . LD <addr>,A |
| **SR:** | . LD (SP + dd),A |
| **BX:** | LD (XXA + XXE |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **Ф11** | **101** | **01** | **rx** | **111** |  | |
| **01** | **110** | **111** |  | | |  | |
| **00** | **rrb** | **010** | / | | | |  |
| **00** | **110** | **010** | **addr(low)** | | | **addr(high)** |  |
| **11** | **101** | **101** | **00** | **xxb** | **011** | **d(low)** | **d(high)** |
| **11** | **Ф11** | **101** | **01** | **110** | **111** | **d** |  |
| **11** | **101** | **101** | **00** | **100** | **011** | **disp(low)** | **disp(high)** |
| **11** | **101** | **101** | **00** | **000** | **011** | **d(low)** | **d(high)** |
| **11** | **101** | **101** | **00** | **bx** | **011** |  | |

01

Field Encodings: ф •.

rx: *f* - V < r

rra: rrb:

xxa: xxb:

bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for BC, 011 for DE

000 for BC, 010 for DE

001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd) 101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd) 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Examples:**

LD A,(HL)

Before instruction execution:

After instruction execution:

A:

HL:

A: 0 В

HL: 1 7 0 C

Data memory:

170C: 0 В

Data memory:

170C: 0 В

**LD**

Load from I or R Register

**LD** A,src

src = I, R

**Operation:**

A src

The contents of the source are loaded into the accumulator. The contents of the source are not affected. The Sign and Zero flags are set according to the value of the data transferred; the Overflow flag is set according to the state of the Interrupt A Enable bit in the Master Status register. Note: The R register does not contain the refresh address and is not modified by refresh transactions.

**S:** Set if the data loaded into the accumulator is negative; cleared otherwise

**Flags:**

**Z:** Set if the data loaded into the accumulator is zero; cleared otherwise

**H:** Cleared

**V:** Set when loading the accumulator if the interrupt A Enable bit is set; cleared otherwise

**N:** Cleared

**C:** Unaffected

**Exceptions:**

Privileged Instruction

Addressing  
Mode

Syntax

LD A, I

LD A,R

Instruction Format

| **01** | **010** | **111** |
| --- | --- | --- |

11 101 101

| **01** | **011** | **111** |
| --- | --- | --- |

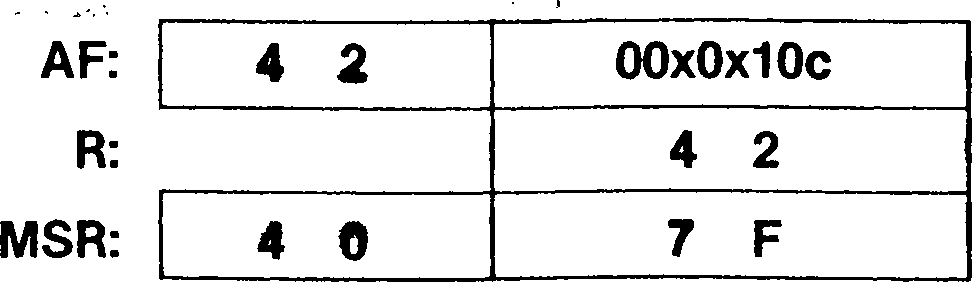
11 101 101

**Example:** LD A,R

|  |  |  |
| --- | --- | --- |
| **AF:** | **1 0** | **szxhxvnc •** |
| **R:** |  | **4 2** |
| **MSR:** | **4 0** | **7 F** |

Before instruction execution:

After instruction execution:



**LD** dst,n

dst = R, RX, IR, DA, X, SX, RA, SR, BX

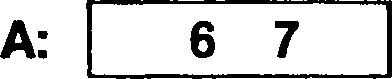
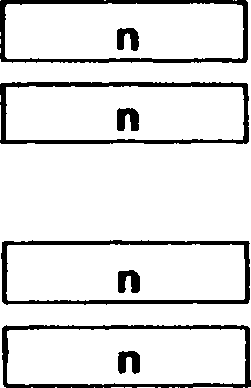
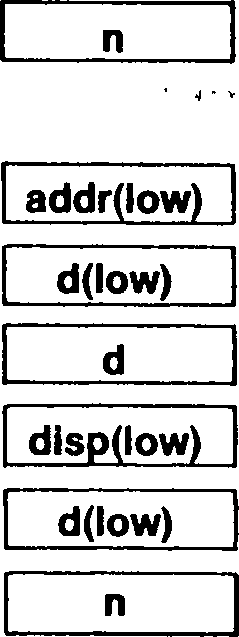
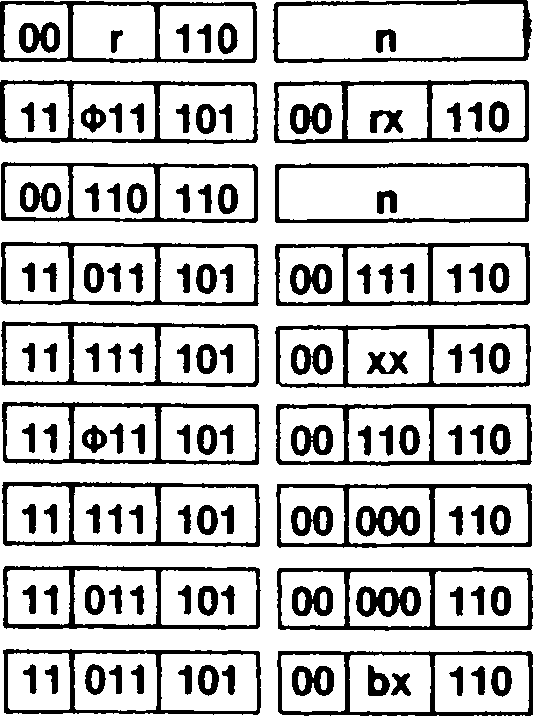
**Operation:**

dst \*- n

The byte of immediate data is loaded into the destination.

**Flags:** No flags affected

**Exceptions:** None



Addressing  
Mode

Syntax

Instruction Format

R: RX: IR: DA:

X: SX: RA: SR: BX:

LD LD LD LD

LD

LD LD LD LD

R,n RX,n (HL),n (addr),n (XX + dd),n (XY + d),n <addr>,n (SP + dd),n (XXA + XXB),n

addr(hlgh)

d(hlgh) n dlsp(high) d(high)

Field Encodings:

Ф: 0 for IX, 1 for IY

rx: 100 for high byte, 101 for low byte

xx: 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

bx : 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Example:**

LD A,55H

Before instruction execution:

After instruction execution:

A: 5 5

**LD** dst,src dst = R

src = R, RX, IM, IR, SX  
or

dst = R, RX, IR, SX

src = R

**Operation:** dst \*- src

The contents of the source are loaded into the destination.

**Flags:** No flags affected

**Exceptions:** None

**Load into Register**

Addressing Mode Syntax

Instruction Format

| **R:** | LD R1.R2 |
| --- | --- |
| **RX:** | LD R\*,RX |
|  | LD RXA, RXB |
|  | LD RX,R\* |
| **IM:** | LD R,n |
|  | LD RX,n |
| **IR:** | LD R,(HL) |
| **SX:** | LD R,(XY + |

| **01** | **r1** | **r2** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |  | **01** | **r\*** | **rx** |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  |  | | |
| **11** | **Ф11** | **101** |  | **01** | **rxa** | **rxb** |
|  | | |  |  | | |
| **11** | **Ф11** | **101** |  | **01** | **rx** | **r\*** |
|  | | |  |  | | |
| **00** | **r** | **110** |  | **n** | | |

11 Ф11 101 00 rx 110

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **01** | **r** | **110** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **00** | **rx** | **110** |
| --- | --- | --- |

| **01** | **r** | **110** |
| --- | --- | --- |

| **01** | **110** | **r** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **01** | **110** | **r** |
| --- | --- | --- |

**Load from Register**

**IR:** LD (HL),R

**SX:** LD (XY + d),R

Field Encodings: ф : о for ix, 1 for iy

rx: 100 for high byte, 101 for low byte

rxa : 100 for high byte, 101 for low byte

rxb : 100 for high byte, 101 for low byte

rxa and rxb refer to the same index register

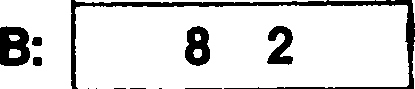
r\* : Only registers А, В, C, D, and E can be accessed r1,r2 : See Table 5-12

**Example:**

LD A, В

Before instruction execution:

After instruction execution:



A: 8 2

**A: О 3**

**В: 8 2**

**LD** dst,A

dst = I, R

**Operation:**

The contents of the accumulator are loaded into the destination. Note: the R register does not contain the refresh address and is not modified by refresh transactions.

**Flags:** No flags affected

**Exceptions:** Privileged Instruction

Addressing Mode Syntax

Instruction Format

11|101|101 11 0110001111

11I1011101 0110011111

LD I,A

LD R.A

**Example:** LD I,A

Before instruction execution:

After instruction execution:

A:

I:

A: 0 D

I: 0 D

**LDA**

**Load Address**

**LDA** dst,src

dst = HL, IX, IY

src = DA, X, RA, SR, BX

**Operation:**

dst address(src)

The address of the source operand is computed and loaded into the destination. The contents of the source are not affected. The address translation mechanism in the MMU is not used to determine if the address is valid.

**Flags:**

No flags affected

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

DA:

X:

RA:

SR:

BX:

LDA HL.(addr)

LDA XY.(addr)

LDA HL,(XX + dd) LDA XY,(XX + dd) LDA HL,<addr> LDA XY,<addr> LDA HL,(SP + dd) LDA XY,(SP + dd) LDA HL,(XXA + XXB) LDA XY,(XXA + XXB)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **00** | **100** | **001** | **addr(low)** | **addr(hlgh)** |  |
| **11** | **Ф11** | **101** | **00 100 001** | **addr(low)** | **addr(high)** |
| **11** | **101** | **101** | **001 xx 1010** | **d(low)** | **d(high)** |
| **11** | **Ф11** | **101** | **11|101 101** | **001 xx 1010** | **d(low)** |
| **11** | **101** | **101** | **0011001010** | **dlsp(low)** | **disp(high)** |
| **11** | **Ф11** | **101** | **111101 101** | **oo|ioo|oio** | **disp(low)** |
| **11** | **101** | **101** | **00 000 010** | **d(low)** | **d(hlgh)** |
| **11** | **Ф11** | **101** | **11|1O11101** | **0010001010** | **d(low)** |
| **11** | **101** | **101** | **001 bx 1010** |  |  |
| **11** | **Ф11** | **101** | **11 101 101** | **00 bx 010** |  |

d(hlgh)

disp(hlgh)

d(high)

Field Encodings: ф:

xx:  
bx:

0 for IX, 1 for IY

101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd)

001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Example:**

LDA HL,(IX + 4)

Before instruction execution:

After instruction execution:

HL:

IX:

HL:

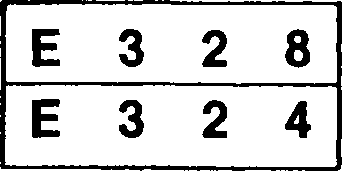
IX:

Address calculation:

E324

4

E328



**LDCTL** dst.src

dst = (C), USP src = HL, IX, IY

or

dst = HL, IX, IY src = (C), USP

**Operation:**

dst src

**LDCTL**

**Load Control**

This instruction loads the contents of a CPU control register into an addressing register, or the contents of an addressing register into a CPU control register. The contents of the source are loaded into the destination; the source register is unaffected. The address of the control register is specified by the contents of the C register, with the exception of the User Stack Pointer. The various CPU control registers have the following addresses:

**Address**

**Register (Hexadecimal)**

Master Status register (MSR) 00

Interrupt Status register 16

[Interrupt/Trap Vector Table Pointer 06](#bookmark80)

I/O Page register \* 08

Bus Timing and Initialization register \* FF

[Bus Timing and Control register \* 02](#bookmark66)

Stack Limit register 04

[Trap Control register \* 10](#bookmark84)

[Cache Control register \* 12](#bookmark70)

[Local Address register \* 14](#bookmark68)

\* 8-bit control register

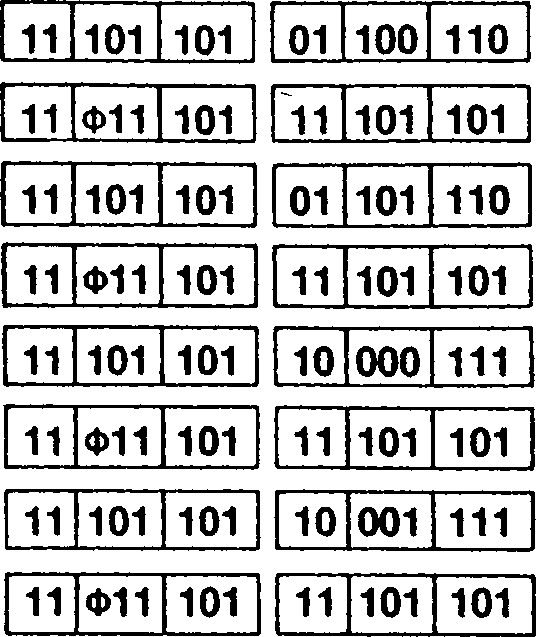
When writing to an 8-bit CPU control register, only the low-order byte of the specified source addressing register is written to the control register. When reading from an 8-bit CPU control register, the control register contents are loaded into the low-order byte of the destination addressing register, and the upper byte of the destination is undefined.

Note that the User Stack Pointer control register is accessed using special opcodes; the contents of the C register are not used for these opcodes. This form of the Load Control instruction allows the user-mode Stack Pointer to be accessed while in system-mode operation.

**Flags:**

No flags affected

**Exceptions:** Privileged Instruction



Addressing  
Mode

Syntax

Instruction Format

LDCTL HL,(C) LDCTL XY,(C) LDCTL (C),HL

LDCTL (C),XY

LDCTL HL,USP LDCTL XY.USP LDCTL USP.HL LDCTL USP.XY

| **01** | **100** | **110** |
| --- | --- | --- |

| **01** | **101** | **110** |
| --- | --- | --- |

| **10** | **000** | **111** |
| --- | --- | --- |

| **10** | **001** | **111** |
| --- | --- | --- |

Field Encoding: ф : о for ix, 1 for iy

**Example:** LDCTL (C),HL

I/O Page register:

|  |  |  |  |
| --- | --- | --- | --- |
| **C:** | **0 •** | **О a a** | **0 8** |
| **HL: 5 5** | **3 A** | **HL: 5 5** | **3 A** |

Before instruction execution:

After instruction execution:

I/O Page register:

**LDD**

**Load and Decrement**

**LDD**

(DE)\*~ (HL)

**Operation:**

DE \*- DE - 1

HL\*- HL - 1

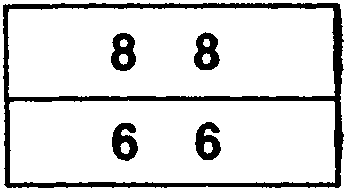
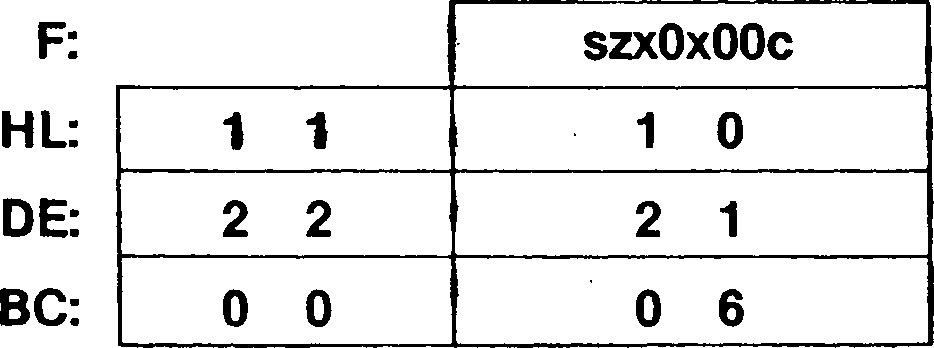
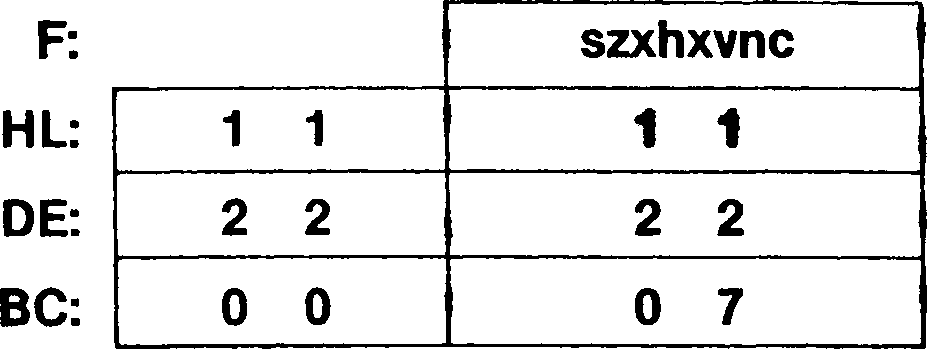
BC \*- BC - 1

This instruction is used for block transfers of strings of data. The byte of data at the loca­tion addressed by the HL register is loaded into the location addressed by the DE register. Both the DE and HL registers are then decremented by one, thus moving the pointers to the preceding elements in the string. The BC register, used as a counter, is then decremented by one.

**Flags: S:** Unaffected

**Z:** Unaffected **H:** Cleared **V:** Set if the result of decrementing BC is not equal to zero; cleared otherwise **N:** Cleared **C:** Unaffected

**Exceptions:** None



Addressing  
Mode

Syntax

LDD

Instruction Format

11 101 101

10 101 000

**Example:**

LDD

Before instruction execution:

Data memory:

After instruction execution:

1111:

2222:

| **1111:** | **8** | **8** |
| --- | --- | --- |
| **2222:** | **8** | **8** |

Data memory:

LDDR

**Load, Decrement and Repeat**

**LDDR**

Repeat until BC = 0: (DE) <- (HL)

**Operation:**

**Flags:**

DE DE - 1

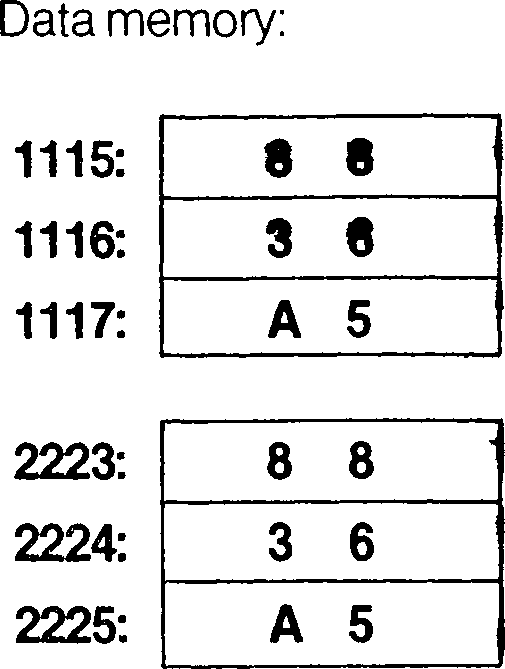
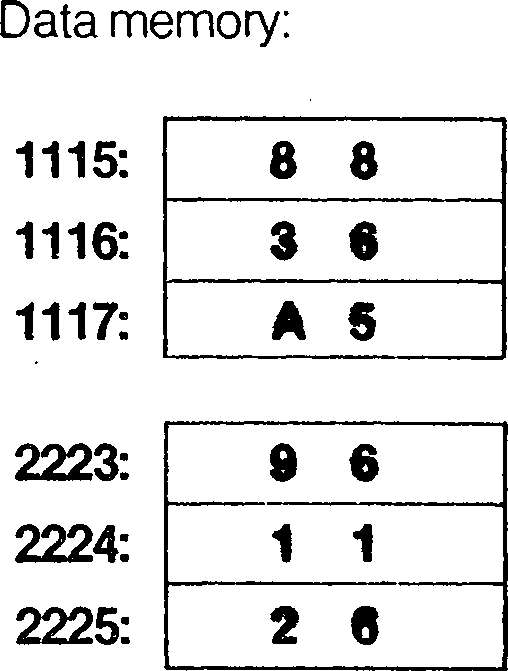
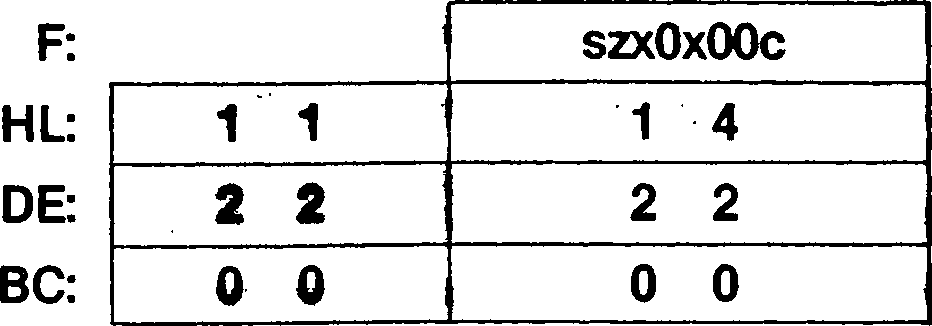
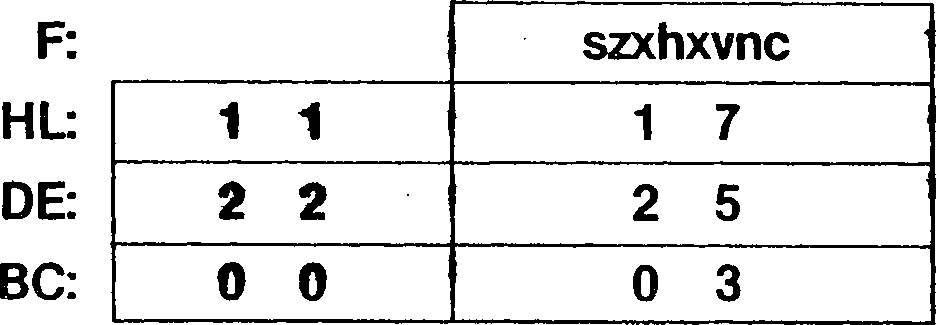
HL\*- HL - 1

BC BC - 1

This instruction is used for block transfers of strings of data. The bytes of data starting at the location addressed by HL are loaded into memory starting at the location addressed by the DE register. The number of bytes moved is determined by the contents of the BC register. If the BC register contains zero when this instruction is executed, 65,536 bytes are transferred. The effect of decrementing the pointers during the transfer is important if the source and destination strings overlap with the source string starting at a lower memory address. Placing the pointers at the highest address of the strings and decrementing the pointers ensures that the source string is copied without destroying the overlapping area.

This instruction can be interrupted after each execution of the basic operation. The Pro­gram Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

**S:** Unaffected **Z:** Unaffected **H:** Cleared **V:** Cleared **N:** Cleared **C:** Unaffected

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

LDDR

| **10** | **111** | **000** |
| --- | --- | --- |

11 101 101

**Example:**

LDDR

Before instruction execution:

After instruction execution:

**LDI**

**LDI**

**Load and Increment**

(DE)-(HL)

**Operation:**

**Flags:**

DE — DE + 1

HL — HL + 1

BC — BC - 1

This instruction is used for block transfers of strings of data. The byte of data at the loca­tion addressed by the HL register is loaded into the location addressed by the DE register. Both the DE and HL registers are then incremented by one, thus moving the pointers to the next elements in the strings. The BC register, used as a counter, is then decremented by one.

**S:** Unaffected **Z:** Unaffected **H:** Cleared -

**V:** Set if the result of decrementing BC is not equal to zero; cleared otherwise **N:** Cleared

**C:** Unaffected

**Exceptions:** None

Addressing  
Mode

Syntax

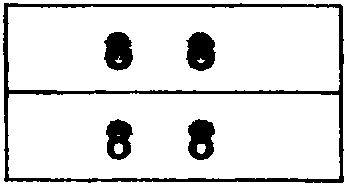
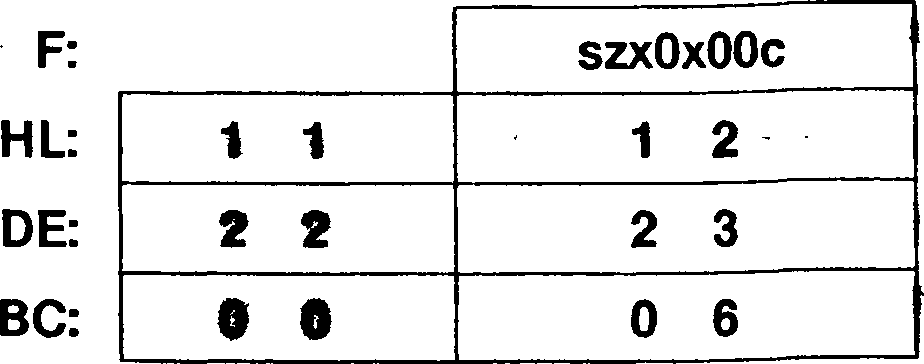
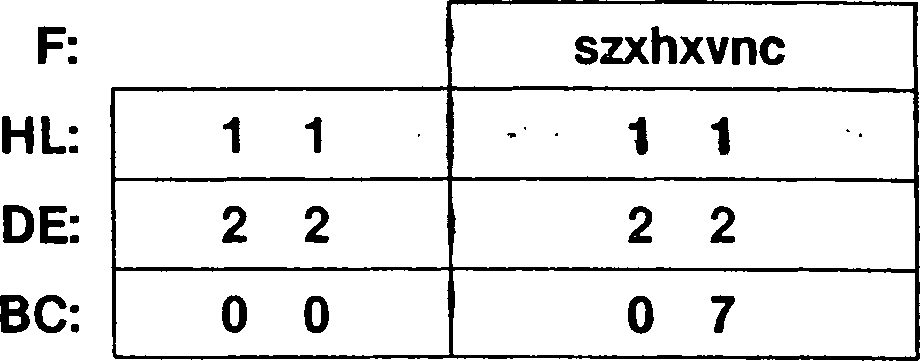
LDI

Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **100** | **000** |
| --- | --- | --- |

**Example:** LDI



Before instruction execution:

Data memory:

After instruction execution:

Data memory:

1111:

2222:

1111:

2222:

**LDIR**

**Load, Increment and Repeat**

**LDIR**

**Operation:**

Repeat until BC = 0: (DE) \*- (HL)

DE DE + 1

HL \*- HL + 1

This instruction is used for block transfers of strings of data. The bytes of data starting at the location addressed by the HL register are loaded into memory starting at the location addressed by the DE register. The number of bytes moved is determined by the contents of the BC register. If the BC register contains zero when this instruction is executed, 65,536 bytes are transferred. The effect of incrementing the pointers during the transfer is important if the source and destination strings overlap with the source string starting at a higher memory address. Placing the pointers at the lowest address of the strings and incrementing the pointers ensures that the source string is copied without destroy­ing the overlapping area.

This instruction can be interrupted after each execution of the basic operation. The Pro­gram Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

**Flags: S:** Unaffected

**Z:** Unaffected

**H:** Cleared

**V:** Cleared

**N:** Cleared

**C:** Unaffected

**Exceptions:** None

Addressing  
Mode Syntax

, t ‘ 4.

Instruction Format

LDIR

|  |  |  |
| --- | --- | --- |
| **11** | **101** | **101** |

|  |  |  |
| --- | --- | --- |
| **10** | **110** | **000** |

**Example:**

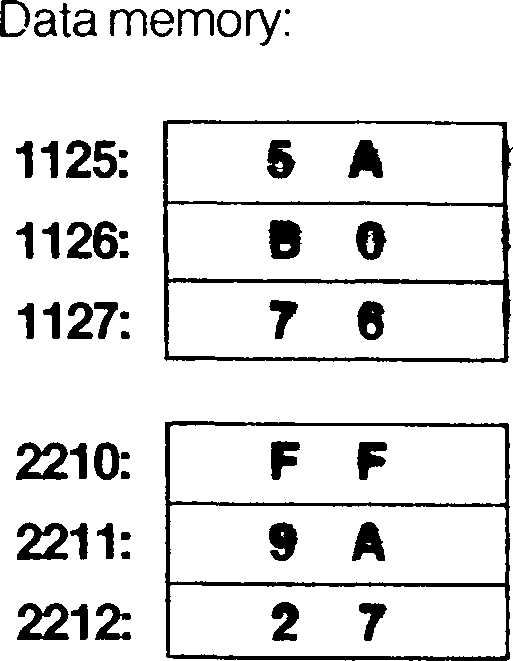
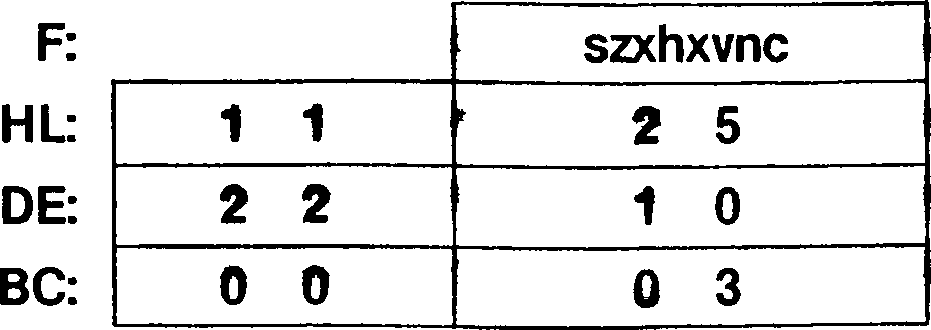
LDIR

HL

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **szxOxOOc** | |
| **1 1** | | **2** | **8** |
| **2** | **2** | **1** | **3** |
| **0** | **0** | **! 0** | **0** |

DE:

BC:



**LDUD**

**Load in User Data Space (Byte)**

dst = A

**LDUD** dst,src

src = IR or SX in user data space - or

dst = IR or SX in user data space src = A

**Operation:**

dst ■\*- src

The destination is loaded with the contents of the source. In loading from the user data space into the accumulator, the memory-mapping mechanism used in translating logical addresses for data in user mode operation is used to translate the source address. In loading into the user data space from the accumulator, the memory-mapping mechanism used in translating logical addresses for data in user-mode operation is used to translate the destination address. See Chapter 7 for an explanation of this mechanism. The con­tents of the source are unaffected.

The flags are set to reflect the success or failure of the transfer. If the transfer is un­successful, no trap is generated and no information is saved in the MMU. If the transfer is successful, the Carry flag is cleared to 0; if the transfer is unsuccessful, the Carry flag is set to 1. The other flags are unaffected if the transfer is successful. If the transfer is unsuccessful, the value of the Write Protect (WP) bit in the Page Descriptor register used by the MMU is loaded into the Z flag and the value of that Page Descriptor’s Valid bit is loaded into the V flag.

**Flags: S:** Unaffected

**Z:** For unsuccessful accesses, loaded with the value of the WP bit used by the MMU; unaffected otherwise

H: U naffected

**V:** For unsuccessful accesses, loaded with the value of the Valid bit used by the MMU; unaffected otherwise

**N:** Unaffected

**C:** Set if the transfer is unsuccessful; cleared otherwise

**Exceptions:**

Privileged Instruction

**Load from User Data Space**

Addressing  
Mode Syntax

Instruction Format

IR:

SX:

LDUD A,(HL)

LDUD A,(XY + d)

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **10** | **000** | **110** |
| --- | --- | --- |

11 101 101

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **000** | **110** |
| --- | --- | --- |

**Load into User Data Space**

**IR:** LDUD (HL),A

**SX:** LDUD (XY + d),A

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **001** | **110** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **001** | **110** |
| --- | --- | --- |

Field Encoding: ф : о for ix, 1 for iy

**Example:** LDUD A,(HL)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Before instruction execution: | | | After instruction execution: | | |
| AF: | 0 F | **szxhxvnc** | AF: | 5 5 | **szxhxvnO** |
| HL: | 8 D | **0 7** | HL: | 8 D | **0 7** |
| User da  8D07: | ita memory:  5 5 |  | User data memory:  8D07: **5** 5 | |  |

LDUP

**Load in User Program Space (Byte)**

**LDUP** dst,src

dst - A

src

dst  
src

= IR or SX in user program space - or

= IR or SX in user program space

= A

**Operation:**

dst \*- src

The destination is loaded with the contents of the source. In loading from the user pro­gram space into the accumulator, the memory-mapping mechanism used in translating logical addresses for program fetches (instructions or data using PC Relative adddress­ing mode) in user-mode operation is used to translate the source address. When loading - into the user program space from the accumulator, the memory-mapping mechanism used in translating logical addresses for program accesses (instructions or data using PC Relative addressing mode) in user-mode operation is used to translate the destination address. See Chapter 7 for an explanation of this mechanism. The contents of the source are unaffected.

The flags are set to reflect the success or failure of the transfer. If the transfer is un­successful, no trap is generated and no information is saved in the MMU. If the transfer is successful, the Carry flag is cleared; if the transfer is unsuccessful, the Carry flag is set. The other flags are unaffected if the transfer is successful. If the transfer is unsuc­cessful, the value of the Write Protect (WP) bit in the Page Descriptor register used by the MMU is loaded into the Z flag and the value of that Page Descriptor’s Valid bit is loaded into the V flag.

**Flags:**

**S: Z:**

**H: V:**

**N:**

**C:**

Unaffected

For unsuccessful accesses, loaded with the value of the WP bit used by the MMU; unaffected otherwise

Unaffected

For unsuccessful accesses, loaded with the value of the Valid bit used by the MMU; unaffected otherwise

Unaffected

Set if the transfer is unsuccessful; cleared otherwise

**Exceptions:**

Privileged Instruction

**Load from User Program Space**

Addressing  
Mode

Syntax

Instruction Format

IR:

SX:

LDUP A,(HL)

LDUP A,(XY + d)

11|1011101

11|ф111101

10|010|110

11I101 101

10 010 110

**Load into User Program Space**

Addressing  
Mode

Syntax

Instruction Format

IR:

SX:

LDUP (HL),A

LDUP (XY + d),A

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **011** | **110** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **011** | **110** |
| --- | --- | --- |

Field Encoding: ф : о for ix, 1 for iy

**Example:** LDUP A,(HL)

After instruction execution:

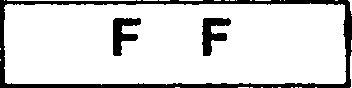
| **AF:** | **0** | **F** | **szxhxvnc** |
| --- | --- | --- | --- |
| **HL:** | **5** | **3** | **9 9 \_** |

Before instruction execution:

User program memory:

| **AF:** | **F F [** | **szxhxvnO** |
| --- | --- | --- |
| **HL:** | **5 3 I** | **9 0** |

User program memory:



5390:

5390: F F

LDW

**Load Immediate Word**

**LD[W]** dst.nn or

dst = R

dst = IR, DA, RA

**LDW** dst.nn

**Operation:**

dst \*- nn

The two bytes of immediate data are loaded into the destination. For register destina­tions, the low byte of the immediate operand is loaded into the low byte of the register and the high byte of the operand is loaded into the high byte of the register. For memory destinations, the low byte of the operand is loaded into the addressed location and the high byte of the operand is loaded into the next higher memory byte (addressed location incremented by one).

**Flags: . v** No flags affected

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

**R:** LDW RR,nn

LDW XY.nn

**IR:** LDW (HL),nn

**DA:** LDW (addr),nn

**RA:** LDW <addr>,nn

| **00** | **rr** | **001** | **n(low)** | | | **n(high)** | • | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **Ф11** | **101** | **00** | **100** | **001** | **n(low)** | **n(high)** |  |  |
| **11** | **011** | **101** | **00** | **000** | **001** | **n(low)** | **nfhigh)** |  |  |
| **11** | **011** | **101** | **00** | **01Q** | **001** | **addr^low)** | **addr(high)** | **n(low)** | **n(high)** |
| **11** | **011** | **101** | **00** | **110** | **001** | **dispflow)** | **disp(high)** | **n(low)** | **n(high)** |

HL: 2 3 9 1

HL: 2 3 9 1

Data memory:

Data memory:

2391:

2392:

2391: 2 5

2392: 3 8

Field Encodings: rr: ooo for вс. 010 for de, 100 for hl, 110 for sp  
Ф: 0 for IX, 1 for IY

LDW (HL),3825H

**Example:**

Before instruction execution:

After instruction execution:

**Operation:**

**LD[W]** dst,src

dst \*- src

LD[W]

**Load Addressing Register**

dst = HL, IX, IY

src = IM, DA, X, RA, SR, BX or

dst = DA, X, RA, SR, BX

src = HL, IX, IY

The contents of the source are loaded into the destination. The contents of the source are unaffected. For register-to-memory transfers, the effective address of the memory operand corresponds to the low byte of the register and the memory byte at the effective address incremented by one corresponds to the high byte of the register.

**Flags:** No flags affected

**Exceptions:** None

**Load into Addressing Register**

Addressing  
Mode

Syntax

Instruction Format

IM:

DA:

X:

RA:

SR:

BX:

LDW HL,nn LDW XY.nn LDW HL,(addr) LDW XY,(addr) LDW HL,(XX + dd) LDW XY,(XX + dd) LDW HL,<addr> LDW XY,<addr> . LDW HL,(SP + dd) LDW XY,(SP + dd) LDW HL, (XXA 4- XXB) LDW XY, (XXA + XXB)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **00** | **100** | **001** | **n(iow)** | **n(high)** |  |
| **11** | **Ф11** | **101** | **00|100|001** | **n(low)** | **n(high)** |
| **00** | **101** | **010** | **addr(low)** | **addr(high)** |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **Ф11** | **101** | **00** | **101** | **010** | **addr(low)** | | | **addr(high)** |  |
| **11** | **101** | **101** | **00** | **XX** | **100** | **d(low)** | | | **d(high)** |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **XX** | **100** | **d(low)** | **d(high)** |
| **11** | **101** | **101** | **00** | **100** | **100** | **disp(low)** | | | **disp(high)** |  |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **100** | **100** | **disp(low)** | **dlsp(high)** |
| **11** | **101** | **101** | **00** | **000** | **100** | **d(low)** | | | **d(high)** |  |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **000** | **100** | **d(low)** | **d(high)** |
| **11** | **101** | **101** | **00** | **bx** | **100** |  | | | | |

|  |  |  |
| --- | --- | --- |
| **11** | **Ф11** | **101** |

|  |  |  |
| --- | --- | --- |
| **11** | **101** | **101** |

|  |  |  |
| --- | --- | --- |
| **00** | **bx** | **100** |

**Load from Addressing Register**

Addressing  
Mode

Syntax

Instruction Format

DA:

X:

RA:

SR:

BX:

LDW (addr),HL LDW (addr),XY LDW (XX + dd),HL LDW (XX + dd),XY LDW <addr>,HL LDW <addr>,XY LDW (SP + dd),HL LDW (SP + dd),XY

LDW (XXA + XXB), HL LDW (XXA + XXB), XY

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **00** | **100** | **010** | **addr(low)** | | | **addr(high)** | | | \* |  |
| **11** | **Ф11** | **101** | **00** | **100** | **010** | **addr(low)** | | | **addr(high)** |  |
| **11** | **101** | **101** | **00** | **XX** | **101** | **d(low)** | | | **d(hlgh)** |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **XX** | **101** | **d(low)** |
| **11** | **101** | **101** | **00** | **100** | **101** | **disp(low)** | | | **disp(high)** |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **100** | **101** | **disp(low)** |
| **11** | **101** | **101** | **00** | **000** | **101** | **d(low)** | | | **d(high)** |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **000** | **101** | **d(low)** | **d(high)** |
| **11** | **101** | **101** | **00** | **bx** | **101** |  | | |  | |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **bx** | **101** |  | |

Field Encodings: <t>:

xx: bx:

0 for IX, 1 for IY

101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd)

001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Example:**

LDW HL,(HL + IX)

Before instruction execution:

After instruction execution:

HL:

IX:

HL: 0 3 A 2

IX: F F F E

Data memory:

Data memory:

1500:

1501:

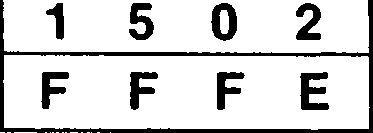
1500: A 2

1501: 0 3

Address calculation:

1502 + FFFE

1500



LD[W]

**Load Register Word**

**LD[W]** dst,src dst = BC, DE, HL, SP

src = IM, IR, DA, SX or dst = IR, DA, SX src = BC, DE, HL, SP

**Operation:** dst \*- src

The contents of the source are loaded into the destination. The contents of the source are unaffected. For transfers between a register and memory, the effective address of the memory operand corresponds to the low byte of the register and the memory byte at the effective address incremented by one corresponds to the high byte of the register.

**Flags:** No flags affected

**Exceptions:** None

**Load into Register**

Addressing  
Mode Syntax

Instruction Format

IM:

IR:

DA:

SX:

LDW RR,nn

LDW RR,(HL)

LDW RR.(addr)

LDW RR,(XY + d)

| **00** | **rra** | **001** | **n(low)** | | | **n(high)** |
| --- | --- | --- | --- | --- | --- | --- |
| **11** | **101** | **101** | **00** | **rra** | **110** |  |

| **11** | **101** | **101** | **01** | **rrb** | **011** | **addr(low)** | | | **addr(high)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **rra** | **110** | **d** |

(except HL)

**Load from Register**

IR:

DA:

SX:

LDW (HL),RR

LDW (addr),RR

LDW (XY 4- d),RR

|  | | | | | | - 4 - ■\* | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **101** | **101** | **00** | **rrb** | **110** |  | |  |
| **11** | **101** | **101** | **01** | **rra** | **011** | **addr(low)** | **addr(high)** | **(except HL)** |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00 rrb 110** | **d** |

Field Encodings: rra: ooo for вс, 010 for de, 100 for hl, 110 for sr

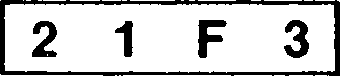
rrb : 001 for BC, 011 for DE, 101 for HL, 111 for SP

Ф : 0 for IX, 1 for IY

**Example:** LDW BC.3824H

Before instruction execution:

After instruction execution:



BC:

BC: 3 8 2 4

**LD[W]**

**Load Stack Pointer**

**LD[W]** dst,src

dst = SP

src = HL, IX, IY, IM, IR, DA, SX or

dst = IR, DA, SX

src = SP

**Operation:**

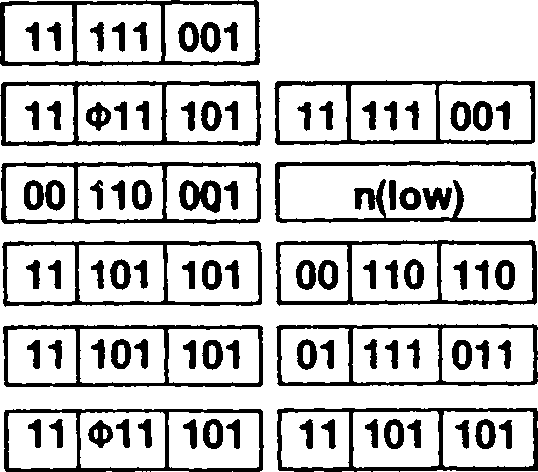
dst src

The contents of the source are loaded into the destination, where the source or destina­tion is the Stack Pointer.

**Flags:** No flags affected

**Exceptions:** None

**Load into Stack Pointer**



Addressing  
Mode

Syntax

Instruction Format

| **R:** | LDW SP,HL  LDW SP,XY |
| --- | --- |
| **IM:** | LDW SP,nn |
| **IR:** | LDW SP,(HL) |
| **DA** | LDW SP,(addr) |
| **SX:** | LDW SP,(XY 4 |

n(hlgh)

addr(low)

00 110 110

addr(high)

**Load from Stack Pointer**

**IR:** LDW (HL),SP

DA

SX:

LDW (addr),SP

LDW (XY + d),SP

| **11** | **101** | **101** | **01** | **110** | **011** | **addr(low)** | | | **addr(high)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **00** | **111** | **110** | **d** |

Field Encoding:

Ф: 0 for IX, 1 for IY

**Example:**

LDW SP,IX

Before instruction execution:

After instruction execution:

SP:

IX:

SP: F F F 0

IX: F F F 0

The contents of the accumulator are multiplied by the source operand and the product is stored in the HL register. The contents of the accumulator and the source are unaffected. Both operands are treated as signed, twos-complement integers.

**MULT** [A,]src

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**Operation:**

HL A X src

The initial contents of the HL register are overwritten by the result. The Carry flag is set to 1 to indicate that the H register is required to represent the result; if the Carry flag is cleared to 0, the product can be correctly represented in eight bits and the H register merely holds sign-extension data.

**Flags: S:** Set if the result is negative; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise

**H:** Unaffected

**V:** Cleared '

**N:** Unaffected

**C:** Set if the product is less than - 27 or greater than or equal to 27; cleared otherwise

**Exceptions:** None

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **R:** MULT A,R | **11** | **101** | **101** | **11** | **1\*** | **000** |  |  |  |  |  |  |
| **RX:** MULT A,RX | **11** | **Ф11** | **101** | **11** | **101** | **101** | **11** | **rx** | **000** |  |  |  |
| **IM:** MULT A.n | **11** | **111** | **101** | **11** | **101** | **101** | **11** | **111** | **000** | **n** |  |  |
| **IR:** MULT A,(HL) | **11** | **101** | **101** | **11** | **110** | **000** |  |  |  |  |  |  |
| **DA:** MULT A,(addr) | **11** | **011** | **101** | **11** | **101** | **101** | **11** | **111** | **000** | **addr(low)** | **addr(hlgh)** |  |
| **X:** MULT A,(XX + dd) | **11** | **111** | **101** | **11** | **101** | **101** | **11 L\_\_J** | **XX** | **000** | **d(low)** | **d(hlgh)** |  |
| **SX:** MULT A,(XY + d) | **11** | **Ф11** | **101** | **11** | **101** | **101** | **11** | **110** | **000** | **d** |  |  |
| **RA:** MULT A,<addr> | **11** | **111** | **101** | **11** | **101** | **101** | **11** | **000** | **000** | **disp(low)** | **disp(high)** |  |
| **SR:** MULT A,(SP + dd) | **11** | **011** | **101** | **11** | **101** | **101** | **11** | **000** | **000** | **d(low)** | **d(high)** |  |
| **BX:** MULT A,(XXA + XXB) | **11** | **011** | **101** | **11** | **101** | **101** | **11** | **bx** | **000** |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Syntax

Instruction Format

Addressing Mode

Field Encodings: ф : rx: xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)

001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

MULT A,H '

**Example:**

Before instruction execution:

After instruction execution:

| **F E** | **szxhxvnc** |
| --- | --- |
| **1 2** | **0 0** |

AF: HL:

| **AF:** | **F E** | **10xhx0n0** |
| --- | --- | --- |
| **HL:** | **F F** | **D C** |

**Operation:**

**MULTU** [A,]src

HL \*- *A* x src

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

The contents of the accumulator are multiplied by the source operand and the product is stored in the HL register. The contents of the accumulator and the source are unaffected. Both operands are treated as unsigned, binary integers.

The initial contents of the HL register are overwritten by the result. The Carry flag is set to 1 to indicate that the H register is required to represent the result; if the Carry flag is cleared to 0, the product can be correctly represented in eight bits and the H register merely holds zero.

**Flags: S:** Cleared .... ....

**Z:** Set if the result is zero; cleared otherwise

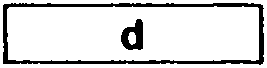
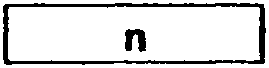
**H:** Unaffected

**V:** Cleared

**N:** Unaffected

**C:** Set if the product is greater than or equal to 28; cleared otherwise

**Exceptions:** None



Addressing  
Mode

Syntax

Instruction Format

R: RX: IM: IR: DA:

X: SX: RA: SR: BX:

MULTU

MULTU

MULTU

MULTU

MULTU

MULTU

MULTU

MULTU

MULTU

MULTU

A,n

A,(HL)

A,(addr)

A,<addr>

| **11** | **101** | **101** | **11** | **r** | **001** |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| **11** | **Ф11** | **101** | **11** | **101** | **101** |
| **11** | **111** | **101** | **11** | **101** | **101** |
|  | | | | | |
| **11** | **101** | **101** | **11** | **110** | **001** |

11 rx 001

11 111 001

11 011 101

11 101 101

11 111 101

11 101 101

addr(low)

11 111 001

11 xx 001

11 Ф11 101

11 111 101

11 011 101

11 011 101

11 101 101

11 101 101

11 101 101

11 101 101

11 000 001

11 000 001

11 bx 001

addr(high)

d(high)

d(low)

11 110 001

disp(low)

disp(high)

d(low)

d(high)

Field Encodings:

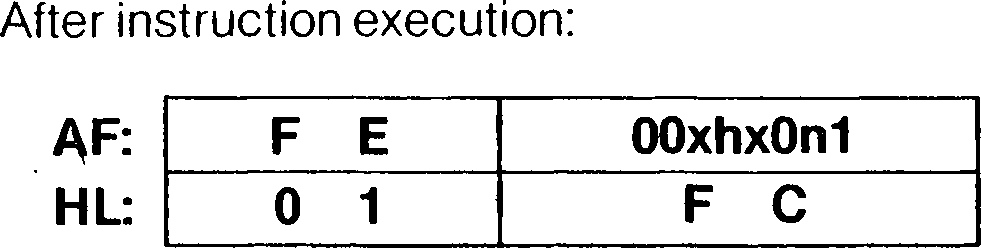
Ф: 0 for IX, 1 for IY

rx : 100 for high byte, 101 for low byte

xx: 001 for (IX + dd), 010 for (IY + dd). 011 for (HL + dd)

bx : 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

**Example:** MULTU A,H



| **AF:** | **F E** | **szxhxvnc** |
| --- | --- | --- |
| **HL:** | **0 2** | **F В** |

Before instruction execution:

**MULTUW** [HL,]src

**Operation:**

DEHL \*- HL x src

MULTUW

**Multiply Unsigned (Word)**

src = R, IM, DA, X, RA

The contents of the HL register are multiplied by the source operand and the product is stored in the DE and HL registers. The contents of the source are unaffected. Both operands are treated as unsigned, binary integers.

The initial contents of the HL register are overwritten by the result. The Carry flag is set to 1 to indicate that the DE register is required to represent the result; if the Carry flag is cleared to 0, the product can be represented correctly in 16 bits and the DE register merely holds zero.

**Flags: S:** Cleared

**Z:** Set if the result is zero; cleared otherwise

**H:** Unaffected .. . .,. .......... ■.

**' к ■ V:** Cleared ’ '' "

**N:** Unaffected '

**C:** Set if the product is greater than or equal to 216; cleared otherwise

**Example:**

MULTUW HL,DE

Exceptions: None

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Addressing Mode Syntax Instruction Format** | | | | | | | | | | | |
| **R:** MULTUW HL,RR  MULTUW HL,XY **IM:** MULTUW HL,nn  **DA:** MULTUW HL,(addr)  **X:** MULTUW HL,(XY + dd)  **RA:** MULTUW HL,<addr>  **IR:** MULTUW HL,(HL) | **11** | **101** | **101** | **11** | **rr** | **011** |  | | |  | |
| **11** | **Ф11** | **101** | **11** | **101** | **101** | **11** | **100** | **011** |  | |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **110** | **011** | **n(low)** | **n(high)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **010** | **011** | **addr(low)** | **addr(high)** |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **xy** | **011** | **d(low)** | **d(high)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **110** | **011** | **disp(low)** | **disp(high)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **000** | **011** |  | |

Field Encodings: <t>:

rr: xy:

0 for IX, 1 for IY

000 for BC, 010 for DE, 100 for HL, 110 for SP

000 for (IX + dd), 010 for (IY + dd)

Before instruction execution: After instruction execution:

|  |  |  |  |
| --- | --- | --- | --- |
| **F:** | **szxhxvnc** | **F:** | **OOxhxOnO** |
| **DE: 0 0** | **0 A** | **DE: 0 0** | **0 0** |
| **HL: 0 0** | **3 1** | **HL: 0 1** | **E A** |

**MULTW** [HLJsrc

src = R, IM, DA, X, RA

**Operation:**

DEHL HL X src

The contents of the HL register are multiplied by the source operand and the product is stored in the DE and HL registers. The contents of the source are unaffected. Both operands are treated as signed, twos-complement integers.

The initial contents of the HL register are overwritten by the result. The Carry flag is set to 1 to indicate that the DE register is required to represent the result; if the Carry flag is cleared to 0, the product can be correctly represented in 16 bits and the DE register merely holds sign-extension data.

**S:** Set if the result is negative; cleared otherwise - •••

**Flags:**

**Z:** Set if the result is zero; cleared otherwise

**H:** Unaffected

**V:** Cleared

**N:** Unaffected

**C:** Set if the product is less than -215 or greater than or equal to 215; cleared otherwise

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

| **R:** | MULTW HL,RR  MULTW HL.XY |
| --- | --- |
| **IM:** | MULTW HL.nn |
| **DA:** | MULTW HL,(addr) |
|  | MULTW HL,(XY + dd) |
| **RA:** | MULTW HL,<addr> |
| **IR:** | MULTW HL,(HL) |

| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **110** | **010** | **n(low)** | **n(high)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **010** | **010** | **addr(low)** | **addr(high)** |
| **11** | **in** | **101** | **11** | **101** | **101** | **11** | **xy** | **010** | **d(low)** | **d(high)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **110** | **010** | **disp(low)** | **disp(high)** |

11|101|101 1111 rr [010

11|Ф111101 11|1O1 101 1111100 1010

11 011 101 11 101 101 11 000 010

Field Encodings: ф : о for ix. 1 for iy

rr: 000 for BC, 010 for DE, 100 for HL, 110 for SP

xy : 000 for (IX + dd). 010 for (IY + dd)

**Example:**

MULTW HL,DE

|  |  |  |  |
| --- | --- | --- | --- |
|  | **szxhxvnc** | **F:** | **OOxhxOnO** |
| **0 0** | **0 A** | **DE: 0 0** | **0 0** |
| **0 0** | **3 1** | **HL: 0 1** | **E A** |

After instruction execution:

Before instruction execution:

DE:

HL:



**NEG** [A]

**Operation:**

**NEG**

**Negate Accumulator**

The contents of the accumulator are negated, that is, replaced by its twos-complement value. Note that 80h is replaced by itself, because in twos-complement representation the negative number with greatest magnitude has no positive counterpart; for this case, the Overflow flag is set to 1.

**Flags:** S: Set if the result is negative, cleared otherwise

**Z:** Set if the result is zero, cleared otherwise

**H:** Set if there was a borrow from the least significant bit of the high-order four bits of the result (bit 4); cleared otherwise

V: Set if the contents of the accumulator was not 80^ before the operation; cleared otherwise.

4, • • 1 I : у . : \* . . ■ N " Sot ; • •, r -X' ” •••\*'■,' 'W'4’ П ‘ ' - i.- Vi -V- 1 5

***\* \* \* s***

C: Set if the contents of the accumulator was not 00u before the operation; cleared otherwise.

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

NEG A

11 101 101

01 000 100

| **2 8** | **szxhxvnc** |
| --- | --- |

Before instruction execution:

AF:

AF: | D 8 | 10x0x010

**Example:** NEG A

After instruction execution:

**NEG** HL

**Operation:**

HL \*- - HL

The contents of the HL register are negated, that is, replaced by its twos-complement value. Note that 8000h is replaced by itself, because in twos-complement representation the negative number with greatest magnitude has no positive counterpart; for this case, the Overflow flag is set to 1.

**S:** Set if the result is negative, cleared otherwise

**Flags:**

**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there was a borrow from the least significant bit of the high-order four bits of the result (bit 12); cleared otherwise

**V:** Set if the contents of HL was 8000h before the operation; cleared otherwise

**N:** Set

C: Set if the contents of HL was not 000u before the operation; cleared otherwise.

**Exceptions:** None

Addressing  
Mode

Syntax

NEG HL

Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **001** | **100** |
| --- | --- | --- |

**Example:** NEG HL

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F:** |  | **szxhxvnc** | **F:** |  | **10x1x010** |
| **HL:** | **0 1** | **2 1** | **HL:** | **F E** | **D F** |

Before instruction execution:

After instruction execution:

NOP

**No Operation**

**NOP**

**Operation:** None

No operation.

**Flags:** No flags affected

**Exceptions:** None

Addressing  
Mode Syntax

Instruction Format

NOP

|  |  |  |
| --- | --- | --- |
| **00** | **000** | **000** |

**Example:** OTDR

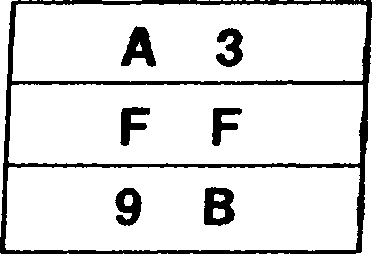
| **p** | | **szxhxvnc** |
| --- | --- | --- |
| **BC:** | **0 3** | **4 6** |
| **HL:** | **5 2** | **1 8** |

| **F:** | | **slxhsvlc** |
| --- | --- | --- |
| **BC:** | **I** | **4 6** |
| **HL:** | **5 2** | **1 5** |

I/O Page register:

Byte 9Bh written to I/O port 1 70346h, then byte FFh written to I/O port 1 70246h, then byte АЗн written to I/O port 170146ц.

Data memory:



5216:

5217:

5218:

OTIR

**Output, Increment and Repeat (Byte, Word)**

**OTIR OTIRW**

Repeat until В = 0: (C) (HL)

**Operation:**

В В — 1

HL \*- AUTOINCREMENT (by one if byte, by two if word)

This instruction is used for block output of strings of data. The string of data is loaded into the selected peripheral from memory at consecutive addresses, starting with the location addressed by the HL register and increasing. During the I/O transactions, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines Aq-A-is, and the contents of the I/O Page register are placed on address lines A16-A23. The byte or word of data from the memory location addressed by the HL register is loaded into the selected peripheral. The В register, used as a counter, is decremented by one. The HL register is then in­cremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next source for the output. If the result of decrementing В is zero, the instruction is terminated, otherwise the output sequence is repeated. Note that if the В register contains 0 at the start of the execution of this instruction, 256 bytes are output.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

**Flags: S:** Unaffected

**Z:** Set **H:** Unaffected **V:** Unaffected **N:** Set **C:** Unaffected

Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

**Exceptions:**

Addressing  
Mode

Syntax

OTIR OTIRW

Instruction Format

|  |  |  |
| --- | --- | --- |
| **11** | **101** | **101** |

|  |  |  |
| --- | --- | --- |
| **10** | **110** | **011** |

|  |  |  |
| --- | --- | --- |
| **11** | **101** | **101** |

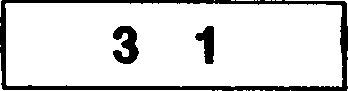
|  |  |  |
| --- | --- | --- |
| **10** | **010** | **011** |

|  |  |  |
| --- | --- | --- |
| **F:** |  | **szxhxvnc** |
| **BC:** | **о к»** | **4 4** |
| **HL:** | **5 0** | **0 4** |

|  |  |
| --- | --- |
|  | **slxhxvlc** |
| **0 0** | **4 4** |
| **5 0** | **0 8** |

BC:

HL:

Word 3A90h written to I/O port 31 0244h, then word B867h written to I/O port 310144H.

I/O Page register:

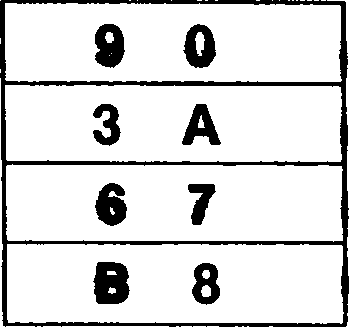
Data memory:

5004:

5005:

5006:

5007:



Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

**OUT** (O.src

src = R, RX, DA, X, RA, SR, BX

**Operation:**

(C) «-src

The byte of data from the source is loaded into the selected peripheral. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines Ag-A-is, and the contents of the I/O Page register are placed on address lines A16-A23- The byte of data from the source is then loaded into the selected peripheral.

**Flags:** No flags affected

**Exceptions:** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode ; Syntax . . Instruction Format

R: RX: DA:

X: RA: SR: BX:

OUT (C),R

OUT (C),RX

OUT (C),(addr)

OUT (C),(XX + dd)

OUT (C),<addr>

OUT (C),(SP + dd)

OUT (C).(XXA + XXB)

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **r** | **001** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **11** | **011** | **101** | **11** | **101** | **101** | **01** | **111** | **001** | **addr(low)** | **addr(high)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **111** | **101** | **11** | **101** | **101** | **01** | **XX** | **001** | **d(low)** | **d(high)** |
| **11** | **111** | **101** | **11** | **101** | **101** | **01** | **000** | **001** | **disp(low)** | **dispfhigh)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **01** | **000** | **001** | **d(low)** | **d(high)** |

11 Ф11 101 11 101 101 01 rx 001

| **11** | **011** | **101** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **bx** | **001** |
| --- | --- | --- |

Field Encodings: ф :

rx:

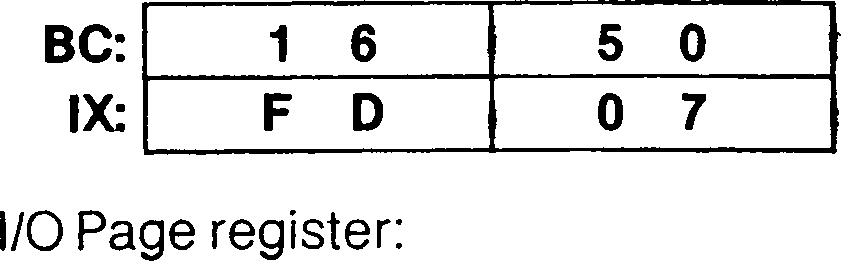
xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX 4- dd), 010 for (IY 4- dd), 011 for (HL + dd)

001 for (HL 4- IX), 010 for (HL 4- IY), 011 for (IX 4- IY)



**Example:**

OUT (C),IXH

Before instruction execution:

After instruction execution:

Byte FDh written to I/O port 321650H

**OUT**

**Output Accumulator**

**OUT** (n),A

**Operation:**

(n)\*-A

The contents of the accumulator are loaded into the selected peripheral. During the I/O transaction, the 8-bit peripheral address from the instruction is placed on the low byte of the address bus, the contents of the accumulator are placed on address lines Ag-A-is, and the contents of the I/O Page register are placed on address lines A16-A23. Then the contents of the accumulator are written into the selected port.

**Flags:** No flags affected

**Exceptions:** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

Syntax

Instruction Format

| **11** | **010** | **011** |
| --- | --- | --- |

OUT (n),A

**Example:** OUT (55H),A

Before instruction execution:

After instruction execution:

Byte 42h written to I/O port 114255H

I/O Page register:

OUTD

**Output and Decrement (Byte, Word)**

**OUTD**

**OUTDW**

(C) \*- (HL)

**Operation:**

B\*-B - 1

HL AUTODECREMENT HL (by one if byte, by two if word)

This instruction is used for block output of strings of data. During the I/O transaction, the . peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines A8-A-| 5, and the contents of the I/O < Page register are placed on address lines А16-А2з. The byte or word of data from the memory location addressed by the HL register is loaded into the selected peripheral. The В register, used as a counter, is decremented by one. The HL register is decremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next source for the output.

**Flags: S:** Unaffected

**Z:** Set if the result of decrementing В is zero; cleared otherwise

**H:** Unaffected

**V:** Unaffected

**N:** Set

**C:** Unaffected

**Exceptions:** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

Syntax

OUTD

OUTDW

Instruction Format

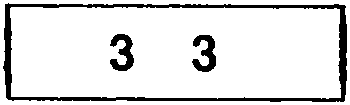
|  |  |  |
| --- | --- | --- |
| **11** | **101** | **101** |

|  |  |  |
| --- | --- | --- |
| **10** | **101** | **011** |

|  |  |  |
| --- | --- | --- |
| **11** | **101** | **101** |

|  |  |  |
| --- | --- | --- |
| **10** | **001** | **011** |

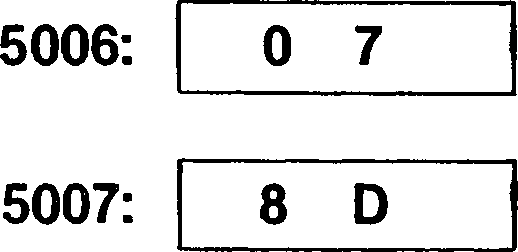
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **F:** | | | **szxhxvnc** | **F:** | | **sOxhxvlc** |
| **BC:** | **1** | **5** | **6 4** | **BC:** | **1 4** | **6 4** |
| **HL:** | **5** | **0** | **0 6** | **HL:** | **5 0** | **0 4** |

Word 8D07h written to

I/O Page register:

I/O port 331564H

Data memory:



Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

OUTI

**Output and Increment (Byte, Word)**

**OUTI OUTIW**

(C) \*- (HL)

**Operation:**

В В — 1

HL \*- AUTOINCREMENT HL (by one if byte, by two if word)

This instruction is used for block output of strings of data. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines Aq-Ais, and the contents of the I/O Page register are placed on address lines A-16-A23. The byte or word of data from the memory location addressed by the HL register is loaded into the selected peripheral. The В register, used as a counter, is decremented by one. The HL register is then incre­mented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next source for the output. r.. . .. . . .. .

**Flags: S:** Unaffected

**Z:** Set if the result of decrementing В is zero; cleared otherwise

**H:** Unaffected

**V:** Unaffected

**N:** Set

**C:** Unaffected

**Exceptions:** Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

Syntax

OUTI

OUTIW

Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **100** | **011** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **10** | **000** | **011** |
| --- | --- | --- |

**Example:**

OUTI

***. , > >■***

Byte 7Bh written to I/O port 331564H

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **F:** | | **szxhxvnc** | R | | | **sOxhxvlc** |
| **BC:** | **1 5** | **6 4** | **BC:** | **1** | **4** | **6 4** |
| **HL:** | **5 0** | **0 2** | **HL:** | **5** | **0** | **0 3** |

Before instruction execution:

After instruction execution:

I/O Page register:

Data memory:

**5002: 7 В**

OUT[W]

**Output HL**

**OUT[W]** (C),HL

**Operation:**

(C) - HL

The contents of the HL register are loaded into the selected peripheral. During the I/O transaction, the 8-bit peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines Aq-A-is, and the contents of the I/O Page register are placed on address lines A16-A23. Then the con­tents of the HL register are written into the selected port. For 8-bit data buses, only the contents of the H register are transferred during a single bus transaction.

**Flags:**

No flags affected

**Exceptions:**

Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing  
Mode

Syntax

Instruction Format

OUTW (C),HL

11 101 101

10 111 111

**Example:** OUTW (C),HL

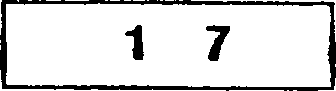
After instruction execution:

| **BC:** | **2** | **6** | **5** | **0** |
| --- | --- | --- | --- | --- |
| **HL:** | **3** | **A** | **8** | **4** |

Before instruction execution:

Word 843Ah written to I/O port 172650ц

I/O Page register:



\* J

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used. .

E3

**PCACHE**

**Purge Cache**

**PCACHE**

**Operation:** All cache entries invalidated

This instruction is used to invalidate all entries in the cache.

**Flags:**

No flags affected

**Exceptions:** None

Addressing  
Mode Syntax

Instruction Format

PCACHE

11 101 101 01 100 101

**POP**

**POP**

**POP** dst

dst = BC, DE, HL, AF, IX, IY, IR, DA, RA

dst \*- (SP)

**Operation:**

SP — SP + 2

The content of the memory location addressed by the Stack Pointer (SP) are loaded into the destination. For register destinations, the byte at the memory location specified by the contents of the SP is loaded into the low byte of the destination register (or Flag register for AF) and the byte at the memory location one greater than the contents of the SP is loaded into the high byte of the destination register. The SP is then incremented by two. If the destination is a memory location, the destination and the top of the stack must be non-overlapping.

**Flags:** No flags affected (unless dst = AF)

**Exceptions:** None

Addressing  
Mode Syntax

**R:** POP RR

POP XY

**IR:** POP (HL)

**DA:** POP (addr)

**RA:** POP <addr>

Instruction Format

| **11** | **rr** | **001** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **11** | **100** | **001** |
| --- | --- | --- |

| **11** | **000** | **001** |
| --- | --- | --- |

| **11** | **011** | **101** | **11** | **010** | **001** | **addr(low)** |
| --- | --- | --- | --- | --- | --- | --- |
| **11** | **011** | **101** | **11** | **110** | **001** | **disp(low)** |

11 011 101 11 000 001

addr(high) disp(high)

Field Encodings: ф : о for ix, 1 for iy

rr: 000 for BC. 010 for DE, 100 for HL, 110 for AF

**Example:**

POP BC . ,

Before instruction execution:

**BC: 2 3 ° ~e~**

**SP: I F E 3 T**

Data memory:

**FE32: 2 3**

**FE33: 0 9**

After instruction execution:

BC:

SP:

Data memory:

FE32:

FE33;

**PUSH** src

**Operation:**

SP — SP - 2 (SP) \*- src

**PUSH**

**Push**

src = BC, DE, HL, AF, IX, IY, IM, IR, DA, RA

The Stack Pointer (SP) is decremented by two and the source is loaded into the location addressed by the updated SP; the low byte of the source (or Flag register for AF) is load­ed into the addressed memory location and the upper byte of the source is loaded into the addressed memory location incremented by one. The contents of the source are unaffected. If the source is a memory location, the source and the new top of the stack must be non-overlapping.

**Flags:** No flags affected

**Exceptions:** System Stack Overflow Warning

Addressing  
Mode

Syntax

Instruction Format

| **R:** | PUSH RR | |
| --- | --- | --- |
| PUSH | XY |
| **IM:** | PUSH | nn |
| **IR;** | PUSH | (HL) |
| **DA:** | PUSH | (addr) |
| **RA:** | PUSH | <addr> |

| **11** | **rr** | **101** |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **Ф11** | **101** | **11** | **100** | **101** |  |  |
| **11** | **111** | **101** | **11** | **110** | **101** | **n(low)** | **n(high)** |
| **11** | **011** | **101** | **11** | **000** | **101** |  |  |
| **11** | **011** | **101** | **11** | **010** | **101** | **addr(low)** | **addr(high)** |
| **11** | **011** | **101** | **11** | **110** | **101** | **disp(low)** | **disp(high)** |

Field Encodings: ф : о for ix, 1 tor iy

rr: 000 for BC, 010 for DE, 100 for HL, 110 for AF

PUSH BC

**Example:**

Before instruction execution:

After instruction execution:

BC: 0 9 2 3

SP: F E 3 4

BC:

SP:

Data memory:

Data memory:

FE32: 0 0

FE33: 0 0

FE32:

FE33:

RES

**Reset Bit**

**RES** b.dst

dst = R, IR, SX

**Operation:** dst(b) \*- 0

The specified bit b within the destination operand is cleared to 0. The other bits in the destination are unaffected. The bit number b must be between 0 and 7.

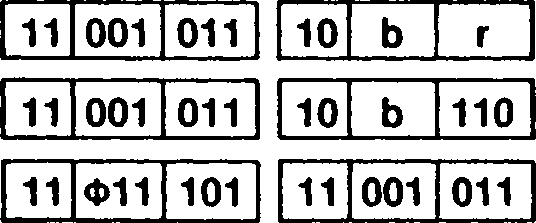
**Flags:**

No flags affected

**Exceptions:** None

Addressing  
Mode

Syntax Instruction Format



R:

IR:

SX:

RES b,R

RES b,(HL)

RES b,(XY + d)

10 b

110

Field Encoding: ф : о for ix, 1 for iy

**Example:**

RES 1,A

Before instruction execution:

After instruction execution:

A: 00010110

A:

00010100

RET

**Return**

**RET** [cc]

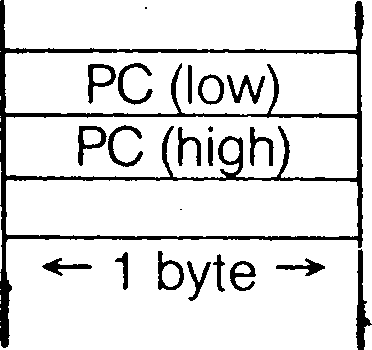
If the cc is satisfied then: PC \*- (SP)

**Operation:**

SP SP + 2

This instruction is used to return to a previously executing procedure at the end of a procedure entered by a Call instruction. For a conditional return, one of the Zero, Carry, Sign, or Parity/Overflow flags is checked to see if its setting matches the condition code “cc” encoded in the instruction; if the condition is not satisfied, the instruction following the Return instruction is executed, otherwise a value is popped from the stack and loaded into the Program Counter (PC), thereby specifying the location of the next instruction to be executed. For an unconditional return, the return is always taken and a condition code is not specified.

The following figure illustrates the format of the PC on the stack for the Return instruction:



SP before

SP after

low address

high address

**Flags:** No flags affected

**Exceptions:** None

Addressing Mode Syntax Instruction Format

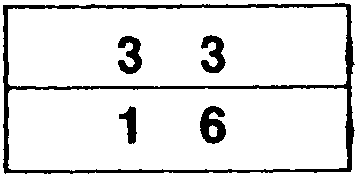
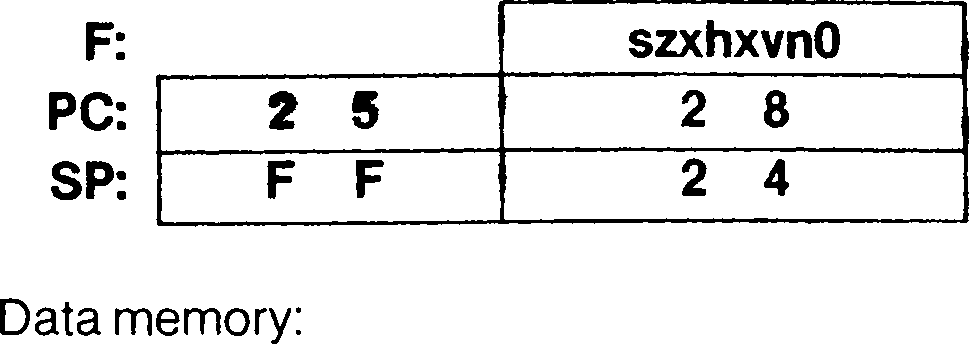
RET cc

11 cc 000

RET

11 001 001

Field Encodings: cc: ooo for nz, 001 for z, 010 for nc, 011 for c, 100 for po or nv, 101 for pe or v, 110 for P or NS, 111 for M or S



**Example:**

RET NC

Before instruction execution:

FF24:

| **F:** | | | **szxhxvnO** |
| --- | --- | --- | --- |
| **PC:** | **1** | **6** | **3 3** |
| **SP:** | **F** | **F** | **2 6** |

After instruction execution:

FF25:

| **FF24:** | **3** | **3** |
| --- | --- | --- |
| **FF25:** | **1** | **6** |

Data memory:

RETI

**Return from Interrupt**

**RETI**



**Operation:**

This instruction is used to return to a previously executing procedure at the end of a pro­cedure entered by an interrupt while in interrupt mode 0, 1, or 2. The contents of the location addressed by the Stack Pointer (SP) are popped into the Program Counter (PC).

The following figure illustrates the format of the PC on the stack for the Return from In­terrupt instruction:

SP before -\*

SP after -\*

PC (low)

PC (high)

1 byte

low address

high address

A special sequence of bus transactions is performed when this instruction is encountered in order to control Z80 family peripherals; see Chapter 12.

**Flags:** No flags affected

**Exceptions:**

Privileged Instruction

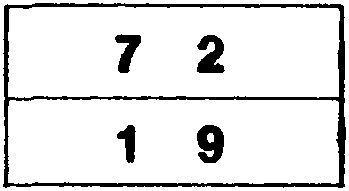
Addressing  
Mode Syntax

Instruction Format

RETI

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **001** | **101** |
| --- | --- | --- |



Data memory:

FFC6:

FFC7:

**Example:** RETI . .... ■.

• V \* •. • 1 ♦ • \* «\* • \* t \* \*

Before instruction execution: After instruction execution:

Data memory:

FFC6: 7 2

FFC7: 1 9

RETIL

**Return from Interrupt Long**

**RETIL**

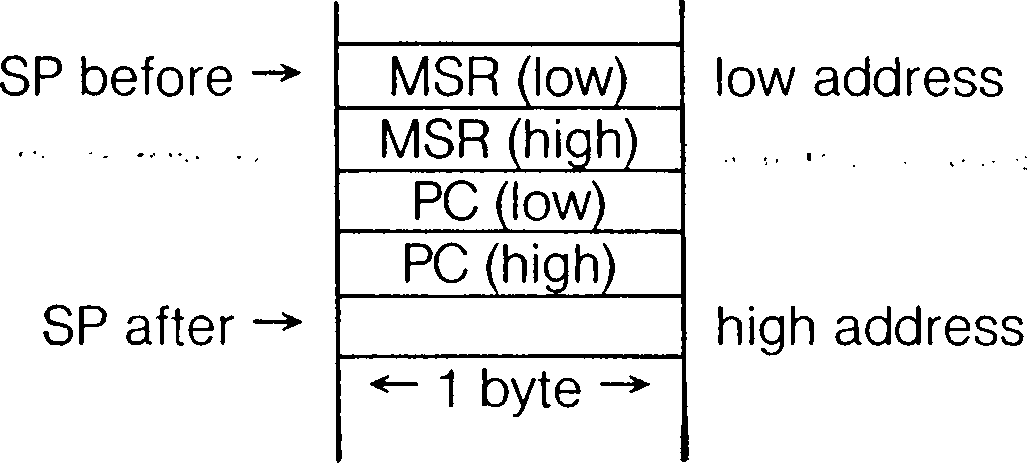
**Operation:**

PS \*- (SP)

SP \*- SP + 4

This instruction is used to return to a previously executing procedure at the end of a pro­cedure entered by an interrupt while in interrupt mode 3 or a trap. The contents of the location addressed by the Stack Pointer (SP) are popped into the Program Counter (PC) and Master Status register (MSR).

The following figure illustrates the format of the program status (PC and MSR) on the system stack for the Return from Interrupt Long instruction:



**Flags:** No flags affected

**Exceptions:**

Privileged Instruction

Addressing  
Mode Syntax

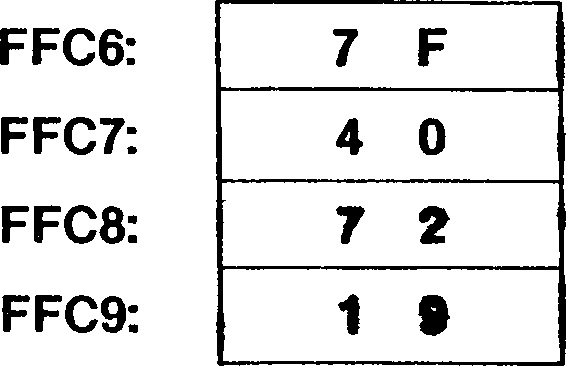
Instruction Format

RETIL

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **010** | **101** |
| --- | --- | --- |

**Example:** RETIL



Data memory:

Data memory:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PC:** | **8 4** | **1 0** | **PC:** | **1 9** | *7 2* |
| **SP:** |  | **C 6** | **SP:** | **F F** | **C A** |
| **MSR:** | **© ©** | **0 0** | **MSR:** | **4 0** | **7 F** |

Before instruction execution:

After instruction execution:

RETN

**Return from Nonmaskable Interrupt**

**RETN**

PC (SP)

**Operation:**

SP \*- SP + 2

MSR(0-7) \*- IFF(0-7)

This instruction is used to return to a previously executing procedure at the end of a pro­cedure entered by a nonmaskable interrupt while in interrupt mode 0,1, or 2. The con­tents of the location addressed by the Stack Pointer (SP) are popped into the Program Counter (PC). The previous setting of the interrupt masks in the Master Status register are restored.

The following figure illustrates the format of the PC on the stack for the Return from Non­maskable Interrupt instruction:

SP before

SP after

PC (low)

PC (high)

\*- 1 byte

low address

high address

**Flags:** No flags affected

**Exceptions:**

Privileged Instruction

Addressing  
Mode

Syntax

Instruction Format

RETN

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **000** | **101** |
| --- | --- | --- |

**Example:**

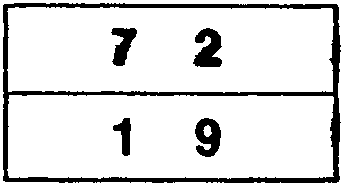
RETN

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PC:** | **8** | **4** | **1** | **0** | **PC:** | **1 9** | | **7** | **2** |
| **SP:** | **F** | **F** | **c** | **6** | **SP:** | **F** | **F** | **C** | **8** |
| **MSR:** | **4** | **0** | **0** | **0** | **MSR:** | **4** | **0** | **7** | **F** |

Before instruction execution:

After instruction execution:

Shadow Interrupt register:

Data memory:

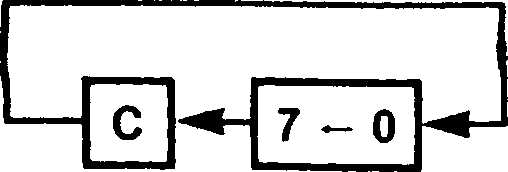
Data memory:

FFC6:

FFC7:

FFC6:

FFC7:



**RL** dst

dst = R. IR. SX

**Operation:**

tmp dst

dst(O) \*-• C

C dst(7)

dst(n + 1) \*- tmp(n) for n = 0 to 6

dst

**RL**

**Rotate Left**

The contents of the destination operand are concatenated with the Carry flag and together they are rotated left one bit position. Bit 7 of the destination operand is moved to the Carry flag and the Carry flag is moved to bit 0 of the destination.

**Flags: S:** Set if the most significant bit of the result is set; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise **H:** Cleared

**P:** Set if the parity of the result is even; cleared otherwise

**N:** Cleared

**C:** Set if the bit rotated from bit 7 was a 1; cleared otherwise

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

R:

IR:

SX:

RL R

RL (HL)

RL (XY + d)

| **11** | **001** | **011** |
| --- | --- | --- |

| **00** | **010** | **r** |
| --- | --- | --- |

| **11** | **001** | **011** |
| --- | --- | --- |

| **00** | **010** | **110** |
| --- | --- | --- |

| **11** | **Ф11** | **101** | **11** | **001** | **011** | d | **00** | **010** | **110** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

11 001 011 00 010 110

Field Encoding: ф : о for ix, 1 tor iy

**Example:** RL D

Before instruction execution:

After instruction execution:

F: szxhxpnO

D: 10001111

F: 00x0x101

D: 00011110

**RLA**

Rotate Left Accumulator

**RLA**

tmp \*- A A(0) \*- C C - A(7)

**Operation:**

**Flags:**

A(n + 1) tmp(n) for n = 0 to 6

The contents of the accumulator are concatenated with the Carry flag and together they are rotated left one bit position. Bit 7 of the accumulator is moved to the Carry flag and the Carry flag is moved to bit 0 of the destination.

Unaffected

**Z:** Unaffected

**H:** Cleared

**P:** Unaffected

**N:** Cleared

**C:** Set if the bit rotated from bit 7 was a 1; cleared otherwise

**Exceptions:**

None

Addressing  
Mode

Syntax

Instruction Format

RLA

00 010 111

**Example:**

RLA

Before instruction execution:

After instruction execution:

AF:

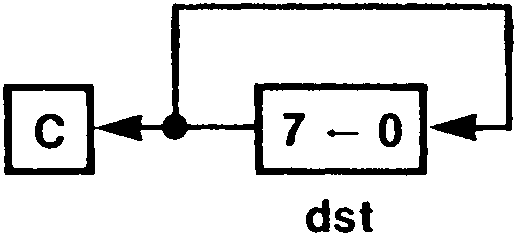
01110110

szxhxpn!

AF:

11101101

szxOxpOO



**RLC** dst

dst = R, IR, SX

**Operation:**

tmp \*- dst

C \*- dst(7)

dst(O) tmp(7)

dst(n + 1) tmp(n) for n = 0 to 6

RLC

**Rotate Left Circular**

The contents of the destination operand are rotated left one bit position. Bit 7 of the destination operand is moved to the bit 0 position and also replaces the Carry flag.

**Flags: S:** Set if the most significant bit of the result is set; cleared otherwise

**- Z:** Set if the result is zero; cleared otherwise . •

**H:** Cleared

**P:** Set if the parity of the result is even; cleared otherwise

**N:** Cleared

**C:** Set if the bit rotated from bit 7 was a 1; cleared otherwise

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

R: IR: SX:

RLC R

RLC (HL)

RLC (XY + d)

| **11** | **001** | **011** |
| --- | --- | --- |

| **11** | **001** | **011** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **00** | **000** | **r** |
| --- | --- | --- |

| **00** | **000** | **110** |
| --- | --- | --- |

| **11** | **001** | **011** |
| --- | --- | --- |

| **00** | **000** | **110** |
| --- | --- | --- |

**Field Encoding:** ф : о tor ix, 1 for iy

**Example:** RLC В

Before instruction execution:

After instruction execution:

**F: szxhxpnc**

**B: 10001000**

**F:**

**B:**

**00x0x101  
00010001**

**RLCA**

**Rotate Left Circular (Accumulator)**

**RLCA**

tmp \*- A

**Operation:**

C \*- A(7)

A(0) tmp(7) ’

A(n + 1) \*- tmp(n) for n = 0 to 6

7-0

The contents of the accumulator are rotated left one bit position. Bit 7 of the accumulator is moved to the bit 0 position and also replaces the Carry flag.

Unaffected

**Flags:**

**Z:** Unaffected

**H:** Cleared

**P:** Unaffected

**N:** Cleared

**C:** Set if the bit rotated from bit 7 was a 1; cleared otherwise

**Exceptions:**

None

Addressing  
Mode

Syntax

Instruction Format

00 000 111

RLCA

**Example:**

RLCA

Before instruction execution:

After instruction execution: ••l- •

10001000

szxhxpnc

00010001

szx0xp01

**RLD**

**Operation:**

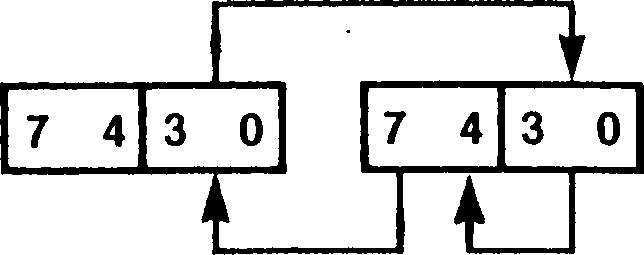
tmp(0:3) \*- A(0:3) A(0:3) \*- dst(4:7) dst(4:7) <- dst(0:3) dst(O:3) \*- tmp(0:3)

A

dst

RLD

**Rotate Left Digit**



The low digit of the accumulator is logically concatenated to the destination byte whose mem­ory address is in the HL register. The resulting three-digit quantity is rotated to the left by one BCD digit (four bits). The lower digit of the source is moved to the upper digit of the source; the upper digit of the source is moved to the lower digit of the accumulator, and the lower digit of the accumulator is moved to the lower digit of the source. The upper digit of the accumulator is unaffected. In multiple-digit BCD arithmetic, this instruction can be used to shift to the left a string of BCD digits, thus multiplying it by a power of ten. The accumulator serves to transfer digits between successive bytes of the string. This is analogous to the use of the Carry flag in multiple-precision shifting using the RL instruction.

**S:** Set if the accumulator is negative after the operation; cleared otherwise

**Flags:**

**Z:** Set if the accumulator is zero after the operation; cleared otherwise

**H:** Cleared

**P:** Set if the parity of the accumulator is even after the operation; cleared otherwise

**N:** Cleared .

**C:** Unaffected

**Exceptions:** None

Addressing  
Mode Syntax

Instruction Format

RLD

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **101** | **111** |
| --- | --- | --- |

**Example:** RLD

After instruction execution:

| **AF:** | **3 7** | **szxhxpnc** |
| --- | --- | --- |
| **HL:** | **5 0 0 0** |  |

Before instruction execution:

Data memory:

AF:

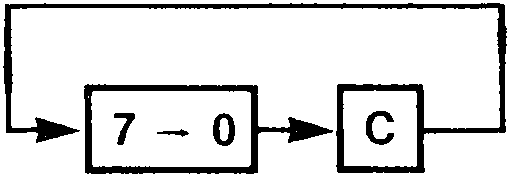
HL:

Data memory:

00x0x10c

5000: 0 4

5000: 4 7



RR

**Rotate Right**

**RR** dst

dst = R, IR, SX

**Operation:**

tmp \*- dst

dst(7) \*- C

C dst(O) ■

dst(n) \*- tmp(n + 1) for n = 0 to 6

dst

The contents of the destination operand are concatenated with the Carry flag and together they are rotated right one bit position. Bit 0 of the destination operand is moved to the Carry flag and the Carry flag is moved to bit 7 of the destination.

**S:** Set if the most significant bit of the result is set; cleared otherwise

**Flags:**

**Z:** Set if the result is zero; cleared otherwise

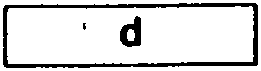
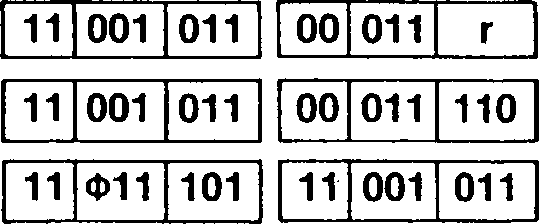
**H:** Cleared

**P:** Set if the parity of the result is even; cleared otherwise

**N:** Cleared

**C:** Set if the bit rotated from bit 0 was a 1; cleared otherwise

**Exceptions:** None



Addressing  
Mode

Syntax

Instruction Format

**R:** RR R

**IR:** RR (HL)

**SX:**RR (XY + d)

00 011 110

**Field Encoding:** ф : о for ix, 1 for iy

**Example:** RR В

Before instruction execution:

After instruction execution:

**szxhxpnO**

**B: 11011101**

**00x0x001 01101110**

RRA

**Rotate Right (Accumulator) \_**

**RRA**

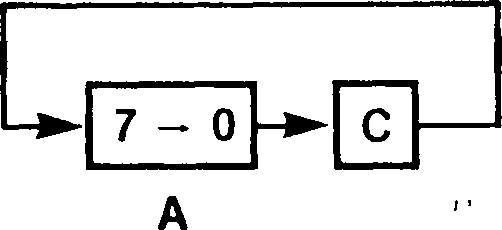
tmp \*- dst

**Operation:**

A(7) \*- C

C A(0)

A(n) \*- tmp(n + 1) for n = 0 to 6



The contents of the accumulator are concatenated with the Carry flag and together they are rotated right one bit position. Bit 0 of the accumulator is moved to the Carry flag and the Carry flag is moved to bit 7 of the accumulator.

**S:** Unaffected

**Flags:**

**Z:** Unaffected

**H:** Cleared

**P:** Unaffected '

**N:** Cleared

**C:** Set if the bit rotated from bit 0 was a 1; cleared otherwise

**Exceptions:** None

**Addressing  
Mode**

**Syntax**

**Instruction Format**

RRA

| **00** | **011** | **111** |
| --- | --- | --- |

| **AF:** | **11100001** | **szxhxpnO** |
| --- | --- | --- |

Before instruction execution:

**AF: 01110000 szx0xp01**

**Example:** RRA

After instruction execution:

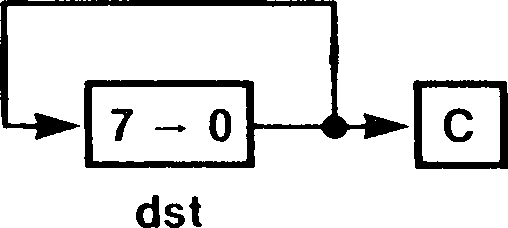
RRC

**Rotate Right Circular**

**RRC** dst dst = R, IR, SX

**Operation:** tmp dst

C \*- dst(O) dst(7) \*- tmp(O) dst(n) \*- tmp(n + 1) for n = 0 to 6



The contents of the destination operand are rotated right one bit position. Bit 0 of the destination operand is moved to the bit 7 position and also replaces the Carry flag.

**Flags: S:** Set if the most significant bit of the result is set; cleared otherwise

**Z:** Set if the result is zero; cleared otherwise **H:** Cleared

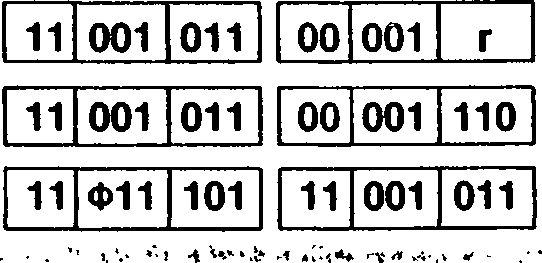
**P:** Set if the parity of the result is even; cleared otherwise

**N:** Cleared

**C:** Set if the bit rotated from bit 0 was a 1; cleared otherwise

**Exceptions:** None

i



**Addressing  
Mode Syntax**

**R:** RRC R

**IR:** RRC (HL)

**SX:** RRC (XY + d)

. < , , . , J' rl-. • /А <T- -ч,- I\*. » , .,<•< >■' . >• A ,

**Instruction Format**

00

001 110

Field Encoding: ф : о for ix, 1 for iy

**Example:** RRC A

AF:

|  |  |
| --- | --- |
| **00110001** | **szxhxpnc** |

Before instruction execution:

After instruction execution:

AF: 10011000 10x0x001

**RRCA**

Rotate Right Circular (Accumulator)

**RRCA**

tmp A

**Operation:**

C A(0)

A(7) \*- temp(O)

A(n) \*- tmp(n + 1) for n = 0 to 6

7-0

dst

The contents of the accumulator are rotated right one bit position. Bit 0 of the accumulator is moved to the bit 7 position and also replaces the Carry flag.

Unaffected

**Flags:**

**Z:** Unaffected ■ ■■■•■ - . > . . ,

**H:** Cleared

**P:** Unaffected

**N:** Cleared

**C:** Set if the bit rotated from bit 0 was a 1; cleared otherwise

**Exceptions:**

None

Addressing  
Mode

Syntax

Instruction Format

00 001 111

RRCA

**Example:**

RRCA

Before instruction execution:

After instruction execution:

AF:

00010001

szxhxpnc

AF:

10001000

szx0xp01

RRD

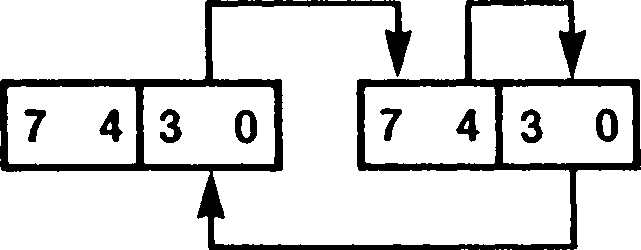
**Rotate Right Digit**

**RRD**

**Operation:**

tmp(0:3) A(0:3)

A(0:3) \*- dst(0:3) dst(0:3) \*- dst(4:7) dst(4:7) \*- tmp(0:3)



A

dst

.. .The low digit of the accumulator is logically concatenated to the destination byte whose mem­ory address is in the HL register. The resulting three-digit quantity is rotated to the right by one BCD digit (four bits). The lower digit of the source is moved to the upper digit of the source; the upper digit of the source is moved to the lower digit of the accumulator, and the lower digit of the accumulator is moved to the lower digit of the source. The upper digit of the accumulator is

/ unaffected. In multiple-digit BCD arithmetic, this instruction can be used to shift to the right a

string of BCD digits, thus multiplying it by a power of ten. The accumulator serves to transfer digits between successive bytes of the string. This is analogous to the use of the Carry flag in multiple-precision shifting using the RR instruction. I

**Flags: S:** Set if the accumulator is negative; cleared otherwise

**Z:** Set if the accumulator is zero after the operation; cleared otherwise **H:** Cleared

**P:** Set if the parity of the accumulator is even after the operation; cleared otherwise **N:** Cleared **C:** Unaffected •

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

RRD

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **100** | **111** |
| --- | --- | --- |

**Example:** RRD

Before instruction execution:

|  |  |  |
| --- | --- | --- |
| **AF:** | **0 6** | **szxhxpnc** |
| **H:** | **5 0** | **0 0** |

Data memory:

After instruction execution:

|  |  |  |
| --- | --- | --- |
| **AF:** | **0 2** | **00x0x00c** |
| **H:** | **СЛ о** | **0 0** |

Data memory:

5000:

5000: 6 3

**rioT**

**Restart**

**RST** address

SP SP - 2

**Operation:**

(SP) \*- PC

PC \*- address

The current Program Counter (PC) is pushed onto the stack and the PC is loaded with a constant address encoded in the instruction. Execution then begins at this address. The restart instruction allows for a call to one of eight fixed locations as shown in the table below. The table also indicates the encoding of the address used in the instruction en­coding. (The address is in hexadecimal, the encoding in binary.)

**Address t encoding**

00H 000

08H - • ■ ' 001 '

10H 010

18H 011

20H 100

28H 101

30H 110

38H 111

**Flags:**

No flags affected

**Exceptions:**

None

Addressing  
Mode

Syntax

Instruction Format

RST address

11 t 111

Field Encoding:

See table above

**Example:** RST 18H

| **PC:** | **4** | **6** | **2** | **0** |
| --- | --- | --- | --- | --- |
| **SP:** | **F** | **F** | **C** | **4** |

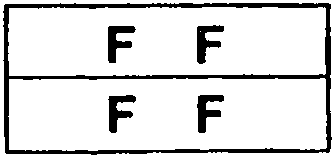
Before instruction execution:

Data memory:

| **PC:** | **0** | **0** | **1 8** |
| --- | --- | --- | --- |
| **SP:** | **F F** | | **C 2** |

After instruction execution:

Data memory:



FFC3:

FFC4:

**FFC3: 2 0**

**FFC4: 4 6**



SBC

**Subtract with Carry (Byte)**

**SBC** [A,]src

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**Operation:**

- src -

The source operand together with the Carry flag is subtracted from the accumulator and the difference is stored in the accumulator. The contents of the source are not affected. Twos-complement subtraction is performed.

**S:** Set if the result is negative; cleared otherwise

**Flags:**

**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there is a borrow from bit 4 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, if the operands are of the opposite signs and the result is the same sign as the source; cleared otherwise

**N:** Set '

**C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise

**Exceptions:** None

**Addressing  
Mode**

**Syntax**

**Instruction Format**

**R:**

SBC

10 011

RX:

SBC

A,RX

11 Ф11 101 10 011 rx

SBC

A,n

| **DA:** SBC A,(addr) | **11** | **011** | **101** | **10** | **011** | **111** | **addr(low)** | **addr(high)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X:** SBC A,(XX + dd) | **11** | **111** | **101** | **10** | **011** | **XX** | **d(low)** | **d(high)** |
| **SX:** SBC A,(XY + d) | **11** | **Ф11** | **101** | **10** | **011** | **110** | **d** |  |
| **RA:** SBC A,<addr> . | **11** | **111** | **101** | **10** | **011** | **000** | **dispflow)** | **dispfhigh)** |
| **SR:** SBC A,(SP 4- dd) | **11** | **011** | **101** | **10** | **011** | **000** | **d(low)** | **d(high)** |
| **BX:** SBC A,(XXA + XXB) | **11** | **011** | **101** | **10** | **011** | **bx** |  |  |
| - |  |  |  |  |  |  |  |  |

IM:

11 011 110

IR:

SBC

A,(HL)

10 011 110

J

Field Encodings: ф :

rx:

xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX 4- dd), 010 for (IY + dd), 011 for (HL + dd) 001 for (HL 4- IX), 010 for (HL 4- IY), 011 for (IX 4- IY)

**Example:**

SBC A,(HL)

Before instruction execution:

After instruction execution:

| **AF:** | **4 8** | **szxhxvnl** |
| --- | --- | --- |
| **HL:** | **2 4** | **5 4** |

| **AF:** | **2 F** | | **00x1x010** |
| --- | --- | --- | --- |
| **HL:** | **2** | **4** | **5 4** |

Data memory:

Data memory:

2454: 1 8

2454: 1 8

**SBC** dst,src

**Operation:**

dst \*\*- dst - src - C

SBC

**Subtract with Carry (Word)**

dst = HL

src = BC, DE, HL, SP or

dst = IX

src = BC, DE, IX, SP or

dst = IY

src = BC, DE, IY, SP

The source operand together with the Carry flag is subtracted from the destination and the result is stored in the destination. The contents of the source are not affected. Twos- complement subtraction is performed.

**S:** Set if the result is negative, cleared otherwise ..... ..

**Flags:**

**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there is a borrow from bit 12 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, the operands are of different signs and the result is of the same sign as the source; cleared otherwise

**N:** Set

**C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise.

**Exceptions:** None

Addressing  
Mode

Syntax

SBC HL,RR

SBC XY,RR

Instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **rr** | **010** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **01** | **rr** | **010** |
| --- | --- | --- |

Field Encodings: <t>: о for ix, 1 for iy

rr: 000 for BC, 010 for DE, 100 for subtract register from itself, 110 for SP

**Example:**

| **F:** | | **szxhxvnl** |
| --- | --- | --- |
| **DE:** | **0 0** | **1 1** |
| **HL:** | **0 1** | **0 0** |

SBC HL,DE

Before instruction execution:

| **F:** | | | **00x0x010** |
| --- | --- | --- | --- |
| **DE:** | **0** | **0** | **1 1** |
| **HL:** | **0** | **0** | **E E** |

After instruction execution:

sc

System Call

**SC** nn

**Operation:**

SP\*-SP- 4

(SP) PS

SP \*- SP - 2

(SP) \*- nn

PS \*- System Call Program Status

***t***

This instruction is used for controlled access to operating system software in a manner similar to a trap or interrupt. The current program status is pushed onto the system stack followed by a 16-bit constant embedded in the instruction. The program status con­sists of the Master Status register (MSR) and the updated Program Counter (PC), which points to the first instruction byte following the SC instruction. Next the 16-bit constant in the System Call instruction is pushed onto the system stack. The system Stack Pointer is always used regardless of whether system or user mode is in effect. The new program status is loaded from the Interrupt/Trap Vector Table entry associated with the SC in­struction. CPU control is passed to the procedure whose address is the PC value con­tained in the new program status.

The following figure illustrates the format of the saved program status on the system stack:

SP after

n (low)

low address

n (high)

MSR (low)

MSR (high)

PC (low)

PC (high)

SP before

high address

1 byte

**Flags:**

No flags affected

**Exceptions:**

System Call Trap, System Stack Overflow Warning

Addressing  
Mode

Syntax

Instruction Format

SC nn

11 101 101 01 110 001

n(low)

n(high)

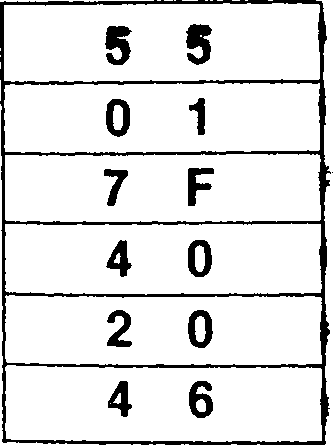
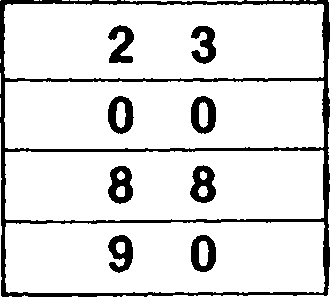
**Example:** SC 0155H

| **PC:**  **SP:** | **4 6** | **2 0** |
| --- | --- | --- |
| **F F** | **C 9** |
| **MSR:** | **4 0** | **7 F** |

Interrupt/Trap Vector Table Pointer:

| **PC:** | **9 0** | **8 8** |
| --- | --- | --- |
| **SP:** | **F F** | **C 3** |
| **MSR:** | **0 0** | **2 3** |

Data memory:



Physical memory:

365250:

365251:

365252:

365253:

FFC3:

FFC4:

FFC5:

FFC6:

FFC7:

FFC8:

Note: The physical memory addresses are 24-bit addresses emitted by the MMU. The data memory addresses are the 16-bit addresses from the CPU.

Set Carry Flag

**SCF**

**Operation:**

The Carry flag is set to 1.

**Flags:**

**S:** Unaffected

**Z:** Unaffected **H:** Cleared **V:** Unaffected **N:** Cleared **C:** Set

**Exceptions:**

None

Addressing

Mode

Syntax

Instruction Format

SCF

00 110 111

**Example:**

SCF

Before instruction execution:

After instruction execution:

szxOxvO!

szxhxvnc

**SET** b,dst

dst = R, IR, SX

SET

**Set Bit**

**Operation:**

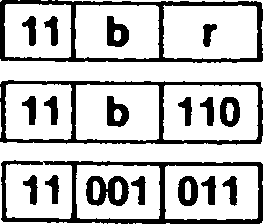
dst(b) \*- 1

The specified bit b within the destination operand is set to 1. The other bits in the destination are unaffected. The bit to be set is specified by a 3-bit field in the instruction; this field contains the binary encoding for the bit number to be set. The bit number must be between 0 and 7. .

**Flags:**

No flags affected

**Exceptions:** None



Addressing  
Mode

R:

IR:

SX:

Syntax

SET b.R

SET b,(HL)

SET b,(XY + d)

Instruction Format

| **11** | **001** | **011** |
| --- | --- | --- |

| **11** | **001** | **011** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

11 b 110

**Field Encoding:** ф : о for ix, 1 for iy

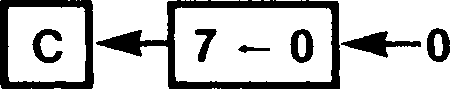
**Example:** SET 1 ,A

Before instruction execution:

After instruction execution:

**A: 00010100**

**A: 00010110**



SLA

**Shift Left Arithmetic**

**SLA** dst

dst = R, IR, SX

**Operation:**

tmp \*- dst

C \*- dst(7)

dst(O) \*- 0

dst(n + 1) \*- tmp(n) for n = 0 to 6

dst

The contents of the destination operand are shifted left one bit position. Bit 7 of the destination operand is moved to the Carry flag and zero is shifted into bit 0 of the destination.

**Flags: S:** Set if the most significant bit of the result is set; cleared otherwise

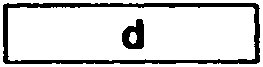
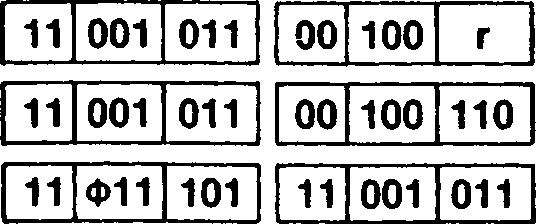
**Z:** Set if the result is zero; cleared otherwise **H:** Cleared

**P:** Set if the parity of the result is even; cleared otherwise

**N:** Cleared

**C:** Set if the bit shifted from bit 7 was a 1; cleared otherwise

**Exceptions:** None



**Addressing  
Mode**

**Syntax**

**Instruction Format**

**R:**

**IR:**

**SX:**

SLA R

SLA (HL)

SLA (XY + d)

00 100 110

**Field Encoding:** ф : о for ix, 1 for iy

**Example:**

SLA L

Before instruction execution:

After instruction execution:

**szxhxpnc 10110001**

**00x0x001 01100010**

SRA

**Shift Right Arithmetic**

**SRA** dst

dst = R, IR, SX

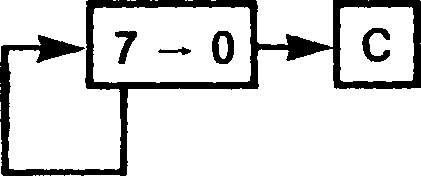
tmp \*- dst

**Operation:**

C \*- dst(O)

dst(7) \*- tmp(7)

dst(n) \*- tmp(n + 1) for n = 0 to 6



dst

The contents of the destination operand are shifted right one bit position. Bit 0 of the destination operand is moved to the Carry flag and bit 7 remains unchanged.

**Flags: S:** Set if the result is negative; cleared otherwise -

**Z:** Set if the result is zero; cleared otherwise

**H:** Cleared

**P:** Set if the parity of the result is even; cleared otherwise

**N:** Cleared

**C:** Set if the bit shifted from bit 0 was 1; cleared otherwise

**Exceptions:** None

**Addressing  
Mode**

**Syntax**

**Instruction Format**

**R:**

**IR:**

**SX:**

SRA R

SRA (HL)

SRA (XY + d)

| **11** | **001** | **011** | **00** | **101** | **Г I** |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| **11** | **001** | **011** | **00** | **101** | **110** |
|  | | | | | |
| **11** | **Ф11** | **101** | **11** | **001** | **011** |

| **00** | **101** | **110** |
| --- | --- | --- |

**Field Encoding:** ф : о for ix, 1 for iy

**Example:**

SRA (IX 4- 3)

Before instruction execution:

After instruction execution:



**F: szxhxpnc**

**IX: Г~0 О О**

**10x0x000  
0 0**

Data memory:

**1003:**

Address calculation:

**10111000**

Data memory:

**1003:**

**11011100**

**1000**

**3**

**1003**

dst = R, IR, SX

SRL

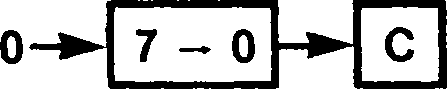
**Shift Right Logical**

**SRL** dst

tmp \*- dst

**Operation:**

C — dst(O) dst(7) \*- 0 •

dst(n) tmp(n + 1) for n = 0 to 6

dst

The contents of the destination operand are shifted right one bit position. Bit 0 of the destination operand is moved to the Carry flag and zero is shifted into bit 7 of the destination.

**Flags: S:** Cleared

**Z:** Set if the result is zero; cleared otherwise

**H:** Cleared

**P:** Set if the parity of the result is even; cleared otherwise

**N:** Cleared

**C:** Set if the bit shifted from bit 0 was 1; cleared otherwise

**Exceptions:** None

**Addressing  
Mode**

**Syntax**

**Instruction Format**

**R:**

**IR:**

**SX:**

SRL R

SRL (HL)

SRL (XY + d)

| **11** | **001** | **011** |
| --- | --- | --- |

| **11** | **001** | **011** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **00** | **111** | **r** |
| --- | --- | --- |

| **00** | **111** | **110** |
| --- | --- | --- |

| **11** | **001** | **011** |
| --- | --- | --- |

| **00** | **111** | **110** |
| --- | --- | --- |

**Field Encoding:** ф : о for ix, 1 for iy

**Example:**

**. ■**

SRL В

Before instruction execution:

After instruction execution:



**szxhxpnc  
10001111**

**00x0x101  
01000111**

**SUB** [A,]src

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**Operation:**

A *A -* src

SUB

**Subtract**

The source operand is subtracted from the accumulator and the difference is stored in the accumulator. The contents of the source are unaffected. Twos-complement subtrac­tion is performed.

. fc

**Flags: S:** Set if the result is negative; cleared otherwise

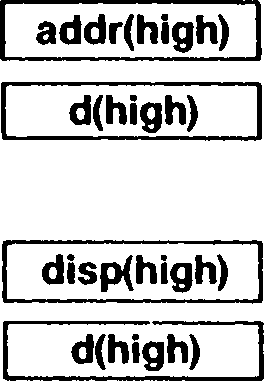
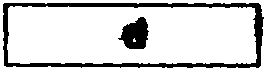
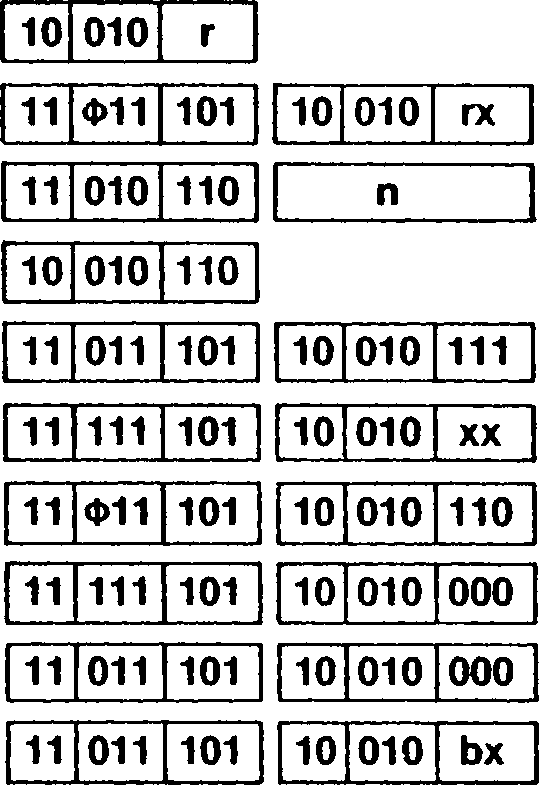
**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there is a borrow from bit 4 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, if the operands are of the opposite signs and the result is the same sign as the source; cleared otherwise

**N:** Set

**., C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise



| **RX:** | SUB | A,RX | |
| --- | --- | --- | --- |
| **IM:** | SUB | A,n |  |
| **IR:** | SUB | A,(HL) |  |
| **DA:** | SUB | A,(addr) |  |
| **X:** | SUB | A,(XX + | dd) |
| **SX:** | SUB | A,(XY 4- | d) |
| **RA:** | SUB | A,<addr> | |
| **SR:** | SUB | A,(SP + | dd) |
| **BX:** | .. .. SUB | A,(XXA - | +- XXB) |

R: SUB A.R

addr(low)

d(low)

dispflow)

d(low)

**Field Encodings:** ф :

rx:

xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX + dd), 010 for (IY + dd), 011 for (HL 4- dd) 001 for (HL 4- IX), 010 for (HL 4- IY), 011 for (IX 4- IY)

**Example:**

| **AF:** | **4 8** | **szxhxvnc** |
| --- | --- | --- |
| **HL:** | **2 4** | **5 4** |

SUB A,(HL)

Before instruction execution:

Data memory:

| **AF:** | **9 0** | **00x0x010** |
| --- | --- | --- |
| **HL:** | **2 4** | **5 4** |

After instruction execution:

Data memory:

2454: 1 8

2454: 1 8

|  |  |  |
| --- | --- | --- |
| **Exceptions:** | None | |
| **Addressing Mode** | **Syntax** | **Instruction Format** |

SUBW

**Subtract (Word)**

**SUBW** [HL,]src

src = R, IM, DA, X, RA

**Operation:**

HL HL - src

The source operand is subtracted from the HL register and the difference is stored in the HL register. The contents of the source are unaffected. Twos-complement subtrac­tion is performed.

**S:** Set if the result is negative; cleared otherwise

**Flags:**

**Z:** Set if the result is zero; cleared otherwise

**H:** Set if there is a borrow from bit 12 of the result; cleared otherwise

**V:** Set if arithmetic overflow occurs, that is, if the operands are of the opposite signs

' ■ and the result is the same sign as the source; cleared otherwise

**N:** Set

**C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise.

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

| **R:** | SUBW  SUBW | HL,RR HL.XY |
| --- | --- | --- |
| **IM:** | SUBW | HL,nn |
| **DA:** | SUBW | HL,(addr) |
| **X:** | SUBW | HL,(XY + < |
| **RA:** | SUBW | HL,<addr> |
| **IR:** | SUBW | HL,(HL) |

4 -W- a” 1 *f • »* -• J ’

| **11** | **101** | **101** |
| --- | --- | --- |

| **11** | **Ф11** | **101** |
| --- | --- | --- |

| **11** | **rr** | **110** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **11** | **101** | **110** |
| --- | --- | --- |

| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **111** | **110** | **n(low)** | **n(high)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **011** | **110** | **addr(low)** | **addr(high)** |
| **11** | **111** | **101** | **11** | **101** | **101** | **11** | **xy** | **110** | **d(low)** | **d(high)** |
| **11** | **011** | **101** | **11** | **101** | **101** | **11** | **111** | **110** | **disp(low)** | **disp(high)** |

11 Ф11 101 11 101 101 11 101 110

11 011 101 11 101 101 11 001 110

| **11** | **011** | **101** |
| --- | --- | --- |

| **11** | **101** | **101** |
| --- | --- | --- |

| **11** | **001** | **110** |
| --- | --- | --- |

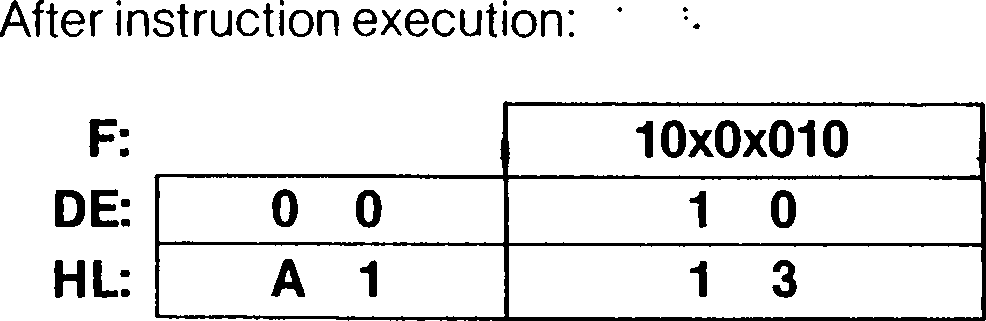
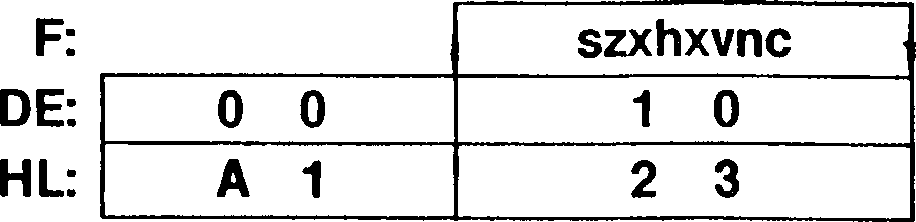
Field Encodings: ф :

rr: xy:

0 for IX. 1 for IY

001 for BC, 011 for DE, 101 for HL, 111 for SP

001 for (IX + dd), 011 for (IY + dd)



**Example:**

SUBW HL,DE

Before instruction execution:

**TSET** dst

dst = R, IR, SX

**Operation:**

S \*- dst(7)  
dst FFH

TSET

**Test and Set**

Bit 7 within the destination operand is tested, and the Sign flag is set to 1 if the specified bit is 1, otherwise the Sign flag is cleared to 0. The contents of the destination are then set to all 1s. For memory operands, the operand is always fetched from the external memory; on the Z-BUS interface, the status lines indicate a Test and Set operation dur­ing the memory read transaction.

Between the data read and subsequent write transactions, bus request is not granted. The data is read from memory, even if it is also present in the cache.

**Flags: S:** Set if bit 7 is 1; cleared otherwise

**Z:** Unaffected < ■'

**H:** Unaffected **P:** Unaffected **N:** Unaffected **C:** Unaffected

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

**R:** TSET R

**IR:** TSET (HL)

**SX:** TSET (XY + d)

| **11** | **001** | **011** |
| --- | --- | --- |

| **00** | **110** | **r** |
| --- | --- | --- |

| **11** | **001** | **011** |
| --- | --- | --- |

| **00** | **110** | **110** |
| --- | --- | --- |

| **11** | **Ф11** | **101** | **11** | **001** | **011** | **d** | **00** | **110** | **110** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

11 001 011 00 110 110

**Field Encoding:** ф : о for ix, 1 for iy

**Example: ...** TSET (HL)



After instruction execution:

|  | **szxhxpnc** |
| --- | --- |
| **0 3** | **8 2** |

Before instruction execution:

**Ozxhxpnc  
8 2**

Data memory:

Data memory:

**0382: 00010111**

**0382: 11111111**

**TSTI**

**Test Input**

**TSTI** (C)

**Operation:**

F test (C)

**Flags:**

**Exceptions:**

**Addressing  
Mode**

**Example:**

During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the В register are placed on address lines A8-A-I5, and the contents of the I/O Page register are placed on address lines Ai6-A23- The byte of data from the selected peripheral is tested and the CPU flags set according­ly. No CPU register or memory location is modified.

**S: Z: H:**

**P: N: C:**

Set if the tested byte is negative; cleared otherwise

Set if the tested byte is zero; cleared otherwise

Cleared

Set if the parity of the tested byte is even; cleared otherwise

Cleared

Unaffected

Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

**Syntax**

**Instruction Format**

TSTI (C)

TSTI (C)

11 101 101 01 110 000

Before instruction execution:

After instruction execution:

szxhxpnc

10x0x10c

BC:

I/O Page register:

Byte 93H available at I/O port 125046h-

**XOR**

**Exclusive OR**

src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

**XOR** [A,]src

**Operation:**

*A* \*- A XOR src

A logical EXCLUSIVE OR operation is performed between the corresponding bits of the source operand and the accumulator and the result is stored in the accumulator. A 1 bit is stored wherever the corresponding bits in the two operands are different; otherwise a 0 bit is stored. The contents of the source are unaffected.

**S:** Set if the most significant bit of the result is set; cleared otherwise

**Flags:**

**Z:** Set if all bits of the result are zero; cleared otherwise

**H:** Cleared

**P:** Set if the parity of the result is even; cleared otherwise

**N:** Cleared

**C:** Cleared .. . ..

**Exceptions:** None

Addressing  
Mode

Syntax

Instruction Format

| **R:** | XOR A.R |
| --- | --- |
| **RX:** | XOR A.RX |
| **IM:** | XOR A,n |
| **IR:** | XOR A,(HL) |
| **DA:** | XOR A,(addr) |
| **X:** | XOR A,(XX +d |
| **SX:** | XOR A,(XY + < |
| **RA:** | XOR A,<addr> |
| **SR:** | XOR A,(SP + < |
| **BX:** | .. XOR A,(XXA + |

10 101 r

11 Ф11 101

10 101

| **11** | **011** | **101** | **10** | **101** | **111** | **addr(low)** | **addr(high)** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **111** | **101** | **10** | **101** | **XX** | **d(low)** | **d(high)** |
| **11** | **Ф11** | **101** | **10** | **101** | **110** | **d** |  |
| **11** | **111** | **101** | **10** | **101** | **000** | **disp(low)** | **dispfhigh)** |
| **11** | **011** | **101** | **10** | **101** | **000** | **d(low)** | **d(high)** |

11 101 110

n

10 101 110

11 011 101

10 101 bx

\*.

Field Encodings: ф :

rx:

xx: bx:

0 for IX, 1 for IY

100 for high byte, 101 for low byte

001 for (IX 4- dd), 010 for (IY 4- dd), 011 for (HL 4- dd)

001 for (HL 4- IX), 010 for (HL 4- IY), 011 for (IX 4- IY)

**Example:**

XOR A,(HL)

Before instruction execution:

After instruction execution:

| **4 8** | **szxhxpnc** |
| --- | --- |
| **2 4** | **5 4** |

| **AF:** | **5 0** | | **. 00x0x100** |
| --- | --- | --- | --- |
| **HL:** | **2** | **4** | **5 4** |

Data memory:

2454: 1 8

Data memory:

2454: 1 8

**Operation:**

EPU \*- template

If the EPU Enable bit in the Trap Control register is set to 1, indicating an EPU is in the system, then the 4-byte template embedded in the instruction is fetched from memory and loaded into the EPU, thus indicating to the EPU the operation to be performed.

If the EPU Enable control bit in the Trap Control register is cleared to 0, an EPU trap is initiated. The trap causes the following information to be pushed onto the system stack (in the following order): Program Counter (PC) of the next instruction, Master Status register (MSR), and template address. The format of the system stack after the trap is in­dicated by the following figure:

new SP -\*

previous SP

template address (low)  
template address (high)  
MSR (low)  
MSR (high)

PC (low)  
PC (high)

1 byte

•'

low address

high address

The format for the EPU template for this instruction is indicated in the following figure:

10001110

\*\*\*\*01ID

\*\*0000

1 byte •

low address

high address

where ID is the two bit ID number specifying the EPU to process this instruction and \* indicates bits that encode the operation to be performed.

The template has no alignment restriction. The CPU fetches the template from external memory using two word transactions if the template is aligned on an even address, or a byte transaction followed by two word transactions if the template is unaligned.

**Flags:**

No flags affected

**Exceptions:** Extended Instruction

Addressing  
Mode

Operation

EPU Internal

Operation

instruction Format

| **11** | **101** | **101** |
| --- | --- | --- |

template 4

| **10** | **011** | **111** |
| --- | --- | --- |

template 1

template 2

template 3

The template is a 4-byte field.

**Operation:**

EPU \*- template

If the EPU Enable bit in the Trap Control register is set to 1, indicating an EPU is in the system, then the 4-byte template embedded in the instruction is fetched from memory and loaded into the EPU, thus indicating to the EPU the operation to be performed. Next data from the EPU is loaded into the accumulator.

If the EPU Enable control bit in the Trap Control register is cleared to 0, an EPU trap is initiated. The trap causes the following information to be pushed onto the system stack (in the following order): Program Counter (PC) of the next instruction, Master Status register (MSR), and template address. The format of the system stack after the trap is in­dicated by the following figure:

new SP -\*•

previous SP

template address (low)  
template address (high)  
MSR (low)  
MSR (high)

PC (low)  
PC (high)

\*- 1 byte

low address

high address

The format for the EPU template for this instruction is indicated in the following figure:

10001110

\*\*\*\*00ID

\*\*\*\*0000

\*\*\*\*0000

\*-1 byte •

low address

high address

where ID is the 2-bit ID number specifying the EPU to process this instruction and \* in­dicates bits that encode the operation to be performed.

The template has no alignment restriction. The CPU fetches the template from external memory using two word transactions if the template is aligned on an even address, or a byte transaction followed by two word transactions if the template is unaligned. The CPU places the data on ADg-AD-is into the accumulator.

**S:** Set if the byte loaded into the accumulator has a 1 in bit 7; cleared otherwise

**Flags:**

**Z:** Set if the byte loaded into the accumulator is zero; cleared otherwise

**H:** Cleared

**P:** Set if the parity of the byte loaded into the accumulator is even; cleared otherwise

**N:** Cleared

**C:** Unaffected

**Exceptions:** Extended Instruction

A EPU

|  |  |  |
| --- | --- | --- |
| **11** | **101** | **101** |

|  |  |  |
| --- | --- | --- |
| **10** | **010** | **111** |

template 1

template 2 template 3

template 4

The template is a 4-byte field.

src = IR, DA, X, RA, SR, BX

**Operation:**

EPU template

EPU \*- src

If the EPU Enable bit in the Trap Control register is set to 1, indicating an EPU is in the system, then the 4-byte template embedded in the instruction is fetched from memory and loaded into the EPU, thus indicating to the EPU the operation to be performed on the input operand. Next the data starting at the memory location determined by the source calculation is fetched from memory and loaded into the EPU; successive trans­fers are performed until the entire operand has been fetched. The number of bytes in the source operand is encoded in the fourth byte of the template. For PC Relative addressing mode, the address of the template is used instead of the address of the next instruction.

If the EPU Enable control bit in the Trap Control register is cleared to 0, an EPU trap is initiated. The trap causes the following information to be pushed onto the system stack (in the following order): Program Counter (PC) of the following instruction, Master Status register (MSR), operand logical address, and template logical address. The format of the system stack after the trap is indicated by the following figure:

new SP -\*■

previous SP -\*

template address (low)  
template address (high)  
operand address (low)  
operand address (high)  
MSR (low)  
MSR (high)

PC (low)  
PC (high)

1 byte -\*

low address

high address

. The format for the EPU template for this instruction is indicated in the following figure:

0p001110

\*\*\*\*01ID

low address

high address

1 byte

where p encodes whether the data resides in program memory (p = 1; Relative ad­dressing mode) or data memory; ID is the 2-bit ID number specifying the EPU to process this instruction, \* indicates bits that encode the operation to be performed, and n specifies the number of bytes of data to be transferred to the EPU.

Neither the template nor the operand has an alignment restriction. The CPU fetches the template from external memory using two word transactions if the template is aligned on an even address, or a byte transaction followed by two word transactions if the template is unaligned. Table 10-2 shows the sequences of transactions for the various cases of data transfers to the EPU.

**Flags:** No flags affected

Addressing  
Mode Operation

Instruction Format

|  |  |  |
| --- | --- | --- |
| **IR:** | EPU < | -(HL) |
| **DA:** | EPU < | - (addr) |
| **X:** | EPU « | — (XX + < |
| **RA:** | EPU - | - <addr> |
| **SR:** | EPU \* | - (SP + c |
| **BX:** | EPU \* |  |

Field Encodings: xx:

11 101 101 10 100 110

template 4

template 1

template 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **template 2** | | | **template 3** | | | **template 4** | |
| **11** | **101** | **101** | **10** | **XX** | **100** | **d(low)** | **d(high)** |
| **template 2** | | | **template 3** | | | **template 4** |  |
| **11** | **101** | **101** | **10** | **100** | **100** | **dispflow)** | **disp(high)** |
| **template 2** | | | **template 3** | | | **template 4** |  |
| **11** | **101** | **101** | **10** | **000** | **100** | **d(low)** | **d(high)** |
| **template 2** | | | **template 3** | | | **template 4** |  |
| **11** | **101** | **101** | **10** | **bx** | **100** | **template 1** | **template 2** |

10 100 111

addr(low)

11 101 101

template 1

template 1

template 1

template 3

template 4

101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd) 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

All templates are 4-byte fields.

template 3

template 1

dst = IR, DA, X, RA, SR, BX

**Operation:**

EPU \*- template dst EPU

If the EPU Enable bit in the Trap Control register is set to 1, indicating an EPU is in the system, then the 4-byte template embedded in the instruction is fetched from memory and loaded into the EPU, thus indicating to the EPU the operation to be performed. Next the data from the EPU is stored into memory starting at the location specified by the destination address; successive transfers are performed until the entire operand has been stored. The number of bytes in the source operand is encoded in the fourth byte of the template. For PC Relative addressing mode, the address of the template is used instead of the address of the next instruction.

If the EPU Enable control bit in the Trap Control register is cleared to 0, an EPU trap is initiated. The trap causes the following information to be pushed onto the system stack (in the following order): Program Counter (PC) of the next instruction, Master Status register (MSR), operand address, and template address. The format of the system stack after the trap is indicated by the following figure:

new SP

previous SP -\*

template address (low)  
template address (high)  
operand address (low)  
operand address (high)  
MSR (low)  
MSR (high)

PC (low)  
PC (high)

1 byte

low address

high address

The format for the EPU template for this instruction is indicated in the following figure:

0p001110 low address

000011 ID ★★★★★★★★

n — 1 high address

\*- 1 byte

where p encodes whether the data resides in program space (p = 1; Relative address­ing mode) or data memory; ID is the 2-bit ID number specifying the EPU to process this instruction, \* indicates bits that encode the operation to be performed, and n specifies the number of bytes of data to be transferred from the EPU.

Neither the template nor the operand has an alignment restriction. The CPU fetches the template from external memory using two word transactions if the template is aligned on an even address, or a byte transaction followed by two word transactions if the template is unaligned. Table 10-2 shows the sequences of transactions for the various cases of data transfers from the EPU.

**Flags:**

No flags affected

Instruction Format

| **11** | **101** | **101** | **10** | **101** | **110** | **template 1** | **template 2** | **template 3** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **template 4** | | | **•** | | | | | |
| **11** | **101** | **101** | **10** | **101** | **111** | **addr(low)** | **addr(high)** | **template 1** |
| **template 2** | | | **template 3** | | | **template 4** |  | |
| **11** | **101** | **101** | **10** | **XX** | **101** | **d(low)** | **d(high)** | **template 1** |
| **template 2** | | | **template 3** | | | **template 4** |  | |
| **11** | **101** | **101** | **10** | **100** | **101** | **dispflow)** | **disp(hlgh)** | **template 1** |
| **template 2** | | | **template 3** | | | **template 4** |  | |
| **11** | **101** | **101** | **10** | **000** | **101** | **d(iow)** | **d(high)** | **template 1** |
| **template 2** | | | **template 3** | | | **template 4** |  | |

| **11** | **101** | **101** | **10** | **bx** | **101** | **template 1** | **template 2** | **template 3** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |

template 4

Field Encodings: xx:

**Exceptions:** Extended Instruction

Addressing Mode Operation

**IR:** (HL) \*- EPU

**DA:** (addr) \*- EPU

**X:** (XX + dd) \*- EPU

**RA:** <addr> \*- EPU

**SR:** (SP + dd) \*- EPU

**BX:** (XXA + XXB) EPU

101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd)

001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

All templates are 4-byte fields.

Chapter 6.

Interrupts and Traps

6.1 INTRODUCTION

Exceptions are conditions that can alter the normal flow of program execution. The Z280 CPU supports three kinds of exceptions: interrupts, traps, and resets.

Interrupts are asynchronous events generated by a device external to the CPU; peripheral devices use interrupts to request service from the CPU. Traps are synchronous events generated internally in the CPU by particular conditions that can occur during the attempted execution of an instruction. Thus, the difference between traps and interrupts is their origin. A trap condition is always repro­ducible by re-executing the program that created the trap, whereas an interrupt is generally inde­pendent of the currently executing task.

A hardware reset overrides all other conditions, including interrupts and traps. It occurs when the RESET line is activated, and it causes certain

CPU control registers to be initialized. Resets are discussed in detail in Chapter 11.

There are seven maskable interrupts in the Z280 MPU architecture. Three of these interrupts are external inputs to the device (Interrupts A, 8, and C); the other four maskable interrupts are asserted by the on-chip peripherals. The seven Interrupt Request Enable bits in the Master Status register control which of the requested interrupts are accepted. Interrupt requests are grouped as listed in Table 6-1, with each group controlled by a separate Interrupt Request Enable bit. The list is presented in order of decreasing priority, with sources within a group listed in order of decreasing priority. .

The Enable Interrupt (El) instruction is used to selectively enable the maskable interrupts (by setting the appropriate bits in the MSR to 1) and the Disable Interrupt (DI) instruction is used to selectively disable interrupts (by clearing the appropriate bits in the MSR to 0). When an interrupt source has been disabled, the CPU ignores any requests from that source. Because maskable interrupt requests are not retained by the CPU, the request signal on a maskable interrupt line must be asserted until the CPU acknowledges the request.

6.2 INTERRUPTS

**Table 6-1. Grouping of Maskable Interrupt Requests**

**Members of Interrupt Group**

**Enable bit in MSR**

Maskable Interrupt A line **0**

Counter/Timer 0, DMA Channel 0 **1**

Maskable Interrupt В line **2**

Counter/Timer 1, UART Receiver, DMA Channel 1 **3**

Maskable Interrupt C line 4

UART Transmitter, DMA Channel 2 5

Counter/Timer 2, DMA Channel 3 6

Two kinds of interrupts are activated by four dif­ferent pins on the Z280 MPU. The nonmaskable interrupt (NM1) is an interrupt that cannot be disabled (masked) by software. Typically, NMI is reserved for high-priority external events that need immediate attention, such as an imminent power failure. Maskable interrupts are interrupts that can be disabled (masked) via software by clearing the appropriate bits in the Interrupt Request Enable field of the Master Status regis­ter.

When enabling interrupts with the EI instruction, all maskable interrupts are automatically disabled (whether previously enabled or not) for the duration of the execution of the EI instruction and the immediately following instruction.

Interrupts are always accepted between instruc­tions. The block move, block search, and block I/O instructions can be interrupted after any iteration.

The Z280 CPU has four modes fur handling exter­nally generated interrupts, selectable using the IM instruction. The first three modes extend the 7.80 CPU interrupt modes tu accommodate the 7280 MPU’s additional interrupt inputs in a compatible fashion. The fourth mode allows more flexibility in interrupt handling, providing support for nested interrupts and a sophisticated vectoring scheme. The on-chip peripherals always use this fourth interrupt mode, regardless of which mode is selected fur the external interrupts. The current interrupt mode in effect can be read from the Interrupt Status register.

Request Enable bits in the Master Status register to be cleared to 0, which puts the CPU in system mode with single-stepping disabled. Ihe previous condition of the MSR is nut saved. Ihe current value in the Program Counter is pushed onto the system-mode stack. for nonmaskable interrupts, the constant 0066|\_| is then loaded into the Pro­gram Counter; thus, 0066^ is the starting address of the nonmaskable interrupt service rou­tine. for maskable interrupts, the constant 0038^ is loaded into the Program Counter; 0038^ will be the starting address of the mask­able interrupt service routine. These logical addresses can be converted to physical addresses by the MMU.

* + 1. Interrupt Mode 0

Interrupt mode 0 is similar to the 8080 CPU interrupt response mode. For mode 0, an exter­nally generated interrupt (maskable or nonmask­able) causes the User/System bit and the Single­Step bit in the Master Status register to be cleared to 0, thereby placing the CPU in system mode with single-stepping disabled. All the Interrupt Request Enable bits in the MSR are also cleared to zero, which disables the maskable interrupts. The previous condition of the MSR is not saved.

for nonmaskable interrupts, the current value in the Program Counter is saved on the system stack, using the System Stack Pointer, and the constant 0066|\_| is loaded into the Program Counter. Loca­tion 0066^| in system program memory is, then, the starting logical address of the nonmaskable interrupt service routine; this logical address can, of course, be translated into a physical mem­

.... ory address by the MMU. -.4,-.-v ..... >

For maskable interrupts, the interrupting device

must place a Call or Restart instruction opcode on

the data bus during the interrupt acknowledge bus

transaction. The Z280 CPU reads this opcode and executes it; thus, the interrupting device, instead of memory, provides the first instruction of the service routine. Typically, a Restart instruction is used, since the Restart opcode is only one byte long, meaning that the interrupting peripheral needs to supply only one byte of infor­mation. Alternatively, a 3-byte call to any loca­tion can be executed.

* + 1. Interrupt Mode 1

In interrupt mode 1, the Z280 CPU automatically executes a Restart to a fixed location when an interrupt occurs. An externally generated inter­rupt (maskable or nonmaskable) causes the User/ System bit, the Single-Step bit, and all Interrupt

* + 1. Interrupt Mode 2

Interrupt mode 2 is a vectored interrupt response mode for maskable interrupts, wherein the inter­rupting device identifies the starting location of the service routine using an 8-bit vector read by the CPU during the interrupt acknowledge cycle.

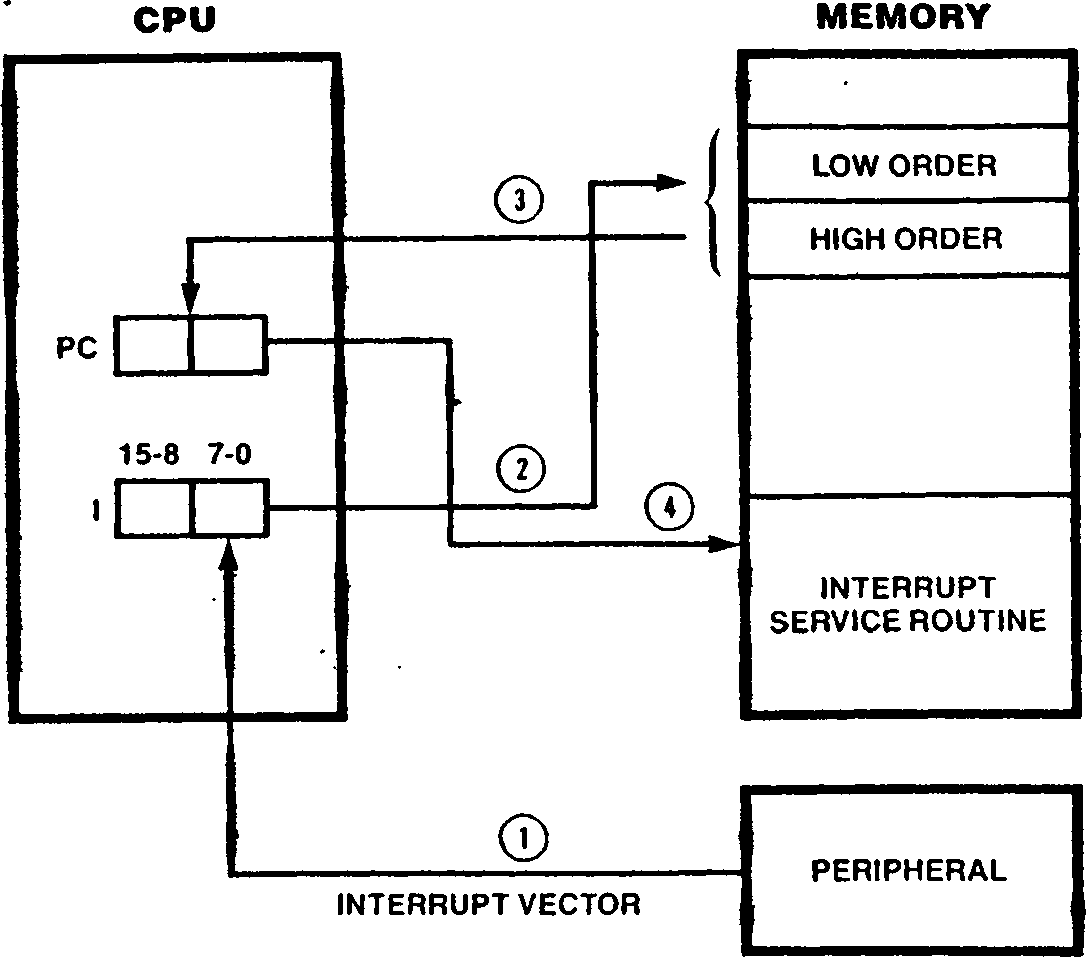
An externally generated interrupt (maskable or nonmaskable) causes the User//System bit, the Sin­gle-Step bit, and the Interrupt Enable Request bits in the Master Status register to be cleared to 0, which puts the CPU in system mode with single-stepping disabled. The previous condition of the MSR is not saved. The current value in the

Program Counter is pushed onto the system mode stack.

Гог nonmaskable interrupts, the constant 0066^ is then loaded into the Program Counter; thus, 0066^| is the starting address of the nonmaskable interrupt service routine. For maskable inter­rupts, the programmer must maintain a table in memory of the 16-bit starting addresses for every maskable interrupt service routine. This table can be located anywhere in the system mode data memory address space, starting on a 256-byte mem­ory boundary. When a maskable interrupt is accepted, a 16-bit pointer into this table is gen­erated in order to select the starting address of the appropriate service routine from the table entries. The peripheral generating the interrupt places an 8-bit vector on the data bus in response tu the interrupt acknowledge. This vector becomes the lower eight bits of the pointer into the table. The upper eight bits of the pointer are the contents of the I register. This pointer is treated as an address in the system data memo г > space that can be translated to a physical address by the MMU. The actual logical address of the service routine is found by referencing the word located at the address formed by concatenating the I register’s contents with the vector. Figure 6-1

sequence of interrupts, zeros.

A reset clears the I



VECTOR TABLE

NOTES:

? ’1. Interrupt vector generated by peripheral is read by CPU during interrupt ’ \*■"'•" '

acknowledge cycle.

2. Vector combined with I register contents form 16-bit memory address pointing to vector table.

3. TWo bytes are read sequentially from vector table. These two bytes are read into the PC.

4. Processor control is transferred to interrupt service routine and execution continues.

Interrupt mode 3 is interrupts from the erals, regardless of

Figure 6-1. Mode 2 Interrupt Processing

The Master Status register is not saved when proc­essing interrupts under interrupt modes 0, 1, and 2. If the Z280 CPU is running in the user mode when an interrupt occurs, the MSR is automatically changed to system mode when the interrupt is acknowledged, without recording the previous user mode of operation. Similarly, the single-step mode and the maskable interrupts are automatically disabled during interrupt processing, with no sav­ing of the previous status. Thus, to resume proc­essing of an interrupted user-mode program after the execution of an interrupt service routine, the operating system must change the Master Status register in order to switch back to user mode; the Return from Interrupt Long instruction can be used for this purpose.

In interrupt modes 0, 1, and 2, a nonmaskable interrupt automatically disables all maskable interrupts (as in the Z80 CPU). All of the Inter­rupt Request Enable bits (bits 0 through 6 in the MSR) are copied to a special register in the CPU called the Interrupt Shadow register. The Inter­rupt Request Enable bits are then cleared to all zeros. A Return from Nonmaskable Interrupt

Гог a Z80 Bus configuration of the Z280 MPU, only one interrupt line (either Interrupt A, Interrupt B, or Interrupt C) can be used if interrupt modes **0f** 1, or 2 and the Z8(l family peripherals are used; Z80 peripherals being serviced on multiple interrupt lines would all be affected by a Return from Interrupt (RE II) instruction.

* + 1. Interrupt Mode 3

Interrupt mode 3 exploits the advanced features of the Z28O MPU architecture. When an interrupt request is accepted (maskable or nonmaskable), the Master Status register, Program Counter, and a 16-bit ’’reason code” are automatically stored on the system-mode stack. Next, new values for the MSR and PC are fetched from a table in memor> called the Interrupt/Trap Vector Table, thereby determining the operating modes and starting address of the service routine (see section 6.5).

The reason code for externally generated inter­rupts is the contents of the data bus during the interrupt acknowledge, and is usually supplied by the interrupting device. For 8-bit data bus con­figurations of the Z280 MPU, the upper byte of the reason code is all zeros. For interrupts from the on-chip peripherals, the reason code is identical to the vector address in the Interrupt/Trap Vector Table, thereby identifying the interrupting device. The Interrupt/Trap Vector Table Pointer register in the CPU is used to reference the Interrupt/Trap Vector Table during mode 3 interrupt processing.

Interrupt mode 3 is the intended mode of operation when using the advanced features of the Z280 MPU architecture, such as system and user modes and sing le-stepping, since the Master Status register of the interrupted task is automatically saved and another loaded for the service routine. This allows each service routine to be executed in the appropriate mode without affecting the status of the interrupted task. Also, vector tables can be provided for both maskable and nonmaskable inter­rupts when in mode 3.

always used for processing Z280 MPU’s on-chip periph- which mode is selected for the external interrupt requests.

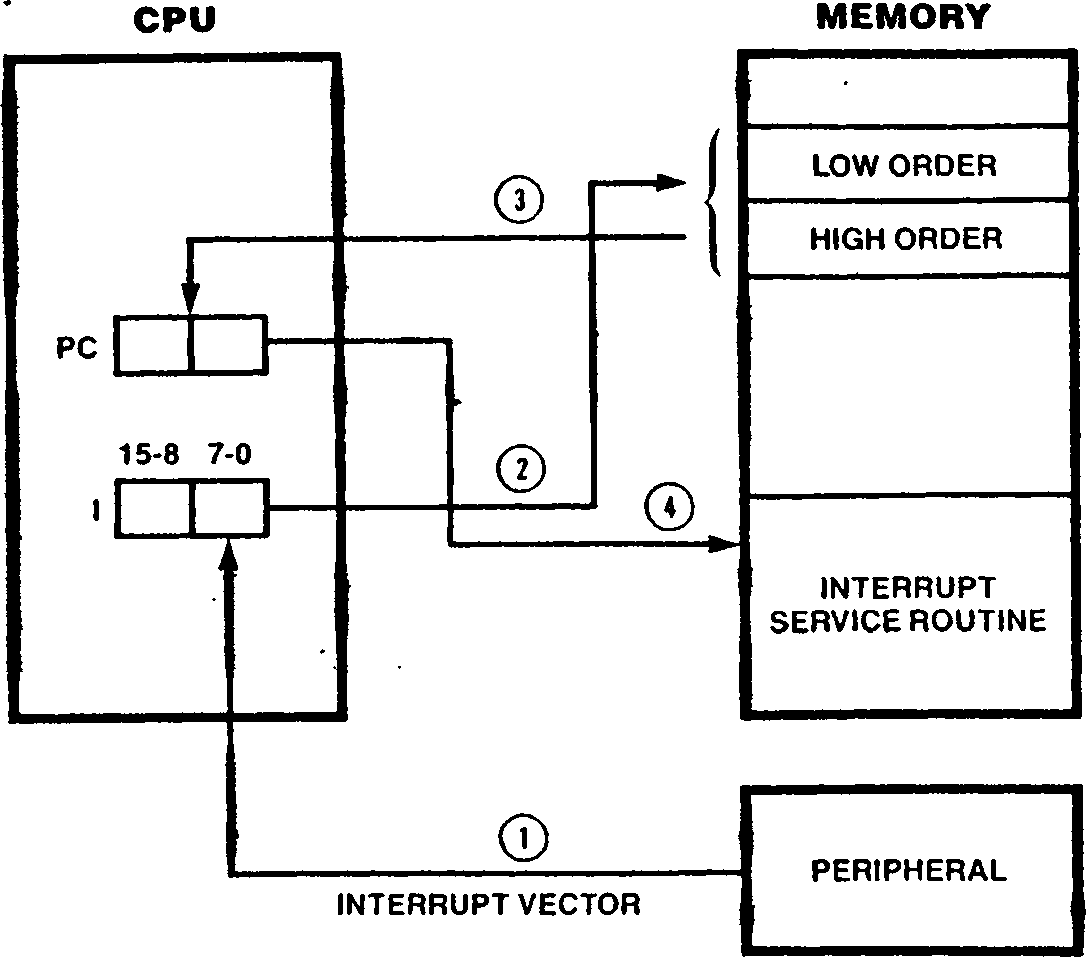
instruction

restores the previous settings of the

Interrupt Request Enable bits by copying the con­tents of the Interrupt Shadow register into bits 0 I able 6-2 summarizes interrupt processing for all four modes.

sequence of interrupts, zeros.

A reset clears the I



VECTOR TABLE

NOTES:

? ’1. Interrupt vector generated by peripheral is read by CPU during interrupt ’ \*■"'•" '

acknowledge cycle.

2. Vector combined with I register contents form 16-bit memory address pointing to vector table.

3. TWo bytes are read sequentially from vector table. These two bytes are read into the PC.

4. Processor control is transferred to interrupt service routine and execution continues.

Interrupt mode 3 is interrupts from the erals, regardless of

Figure 6-1. Mode 2 Interrupt Processing

The Master Status register is not saved when proc­essing interrupts under interrupt modes 0, 1, and 2. If the Z280 CPU is running in the user mode when an interrupt occurs, the MSR is automatically changed to system mode when the interrupt is acknowledged, without recording the previous user mode of operation. Similarly, the single-step mode and the maskable interrupts are automatically disabled during interrupt processing, with no sav­ing of the previous status. Thus, to resume proc­essing of an interrupted user-mode program after the execution of an interrupt service routine, the operating system must change the Master Status register in order to switch back to user mode; the Return from Interrupt Long instruction can be used for this purpose.

In interrupt modes 0, 1, and 2, a nonmaskable interrupt automatically disables all maskable interrupts (as in the Z80 CPU). All of the Inter­rupt Request Enable bits (bits 0 through 6 in the MSR) are copied to a special register in the CPU called the Interrupt Shadow register. The Inter­rupt Request Enable bits are then cleared to all zeros. A Return from Nonmaskable Interrupt

Гог a Z80 Bus configuration of the Z280 MPU, only one interrupt line (either Interrupt A, Interrupt B, or Interrupt C) can be used if interrupt modes **0f** 1, or 2 and the Z8(l family peripherals are used; Z80 peripherals being serviced on multiple interrupt lines would all be affected by a Return from Interrupt (RETI) instruction.

6.2.4 Interrupt Mode 3

Interrupt mode 3 exploits the advanced features of the Z28O MPU architecture. When an interrupt request is accepted (maskable or nonmaskable), the Master Status register, Program Counter, and a 16-bit ’’reason code” are automatically stored on the system-mode stack. Next, new values for the MSR and PC are fetched from a table in memor> called the Interrupt/Trap Vector Table, thereby determining the operating modes and starting address of the service routine (see section 6.5).

The reason code for externally generated inter­rupts is the contents of the data bus during the interrupt acknowledge, and is usually supplied by the interrupting device. For 8-bit data bus con­figurations of the Z280 MPU, the upper byte of the reason code is all zeros. For interrupts from the on-chip peripherals, the reason code is identical to the vector address in the Interrupt/Trap Vector Table, thereby identifying the interrupting device. The Interrupt/Trap Vector Table Pointer register in the CPU is used to reference the Interrupt/Trap Vector Table during mode 3 interrupt processing.

Interrupt mode 3 is the intended mode of operation when using the advanced features of the Z280 MPU architecture, such as system and user modes and sing le-stepping, since the Master Status register of the interrupted task is automatically saved and another loaded for the service routine. This allows each service routine to be executed in the appropriate mode without affecting the status of the interrupted task. Also, vector tables can be provided for both maskable and nonmaskable inter­rupts when in mode 3.

always used for processing Z280 MPU’s on-chip periph- which mode is selected for the external interrupt requests.

instruction

restores the previous settings of the

Interrupt Request Enable bits by copying the con­tents of the Interrupt Shadow register into bits 0 I able 6-2 summarizes interrupt processing for all four modes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TohiA e-2. Interrupt Modes** | | | | |
|  | | | **Effect on MSR**  System mode, Single-Step and interrupts disabled | **Effect on Pc** i  Set to 66ц ' "^\*\*\*\*\*^^ |
| **Interrupt Mode** | **Interrupt Type** | **Saved Status Information** |
| 0 | Nonmaskable | PC | 1  \* |
| 0 | Maskable | \* | /Г | Set to 66ц |
| 1 | Nonmaskable | PC | ***tl*** | Set to 38h |
| 1 | Maskable | PC | ***H*** | Set to 66ц |
| 2 | Nonmaskable | PC |  | Fetched from address formed by । |
| 9 | Maskable | PC | ***H*** |
|  |  |  | register and interrupt vector |
| 3 | Nonmaskable | MSR, PC, and | Fetched from Interrupt/ | Fetched from Interrupt/ |
|  | reason code | Trap Vector Table | Trap Vector Table |
| 3 | Maskable | MSR, PC, and reason code | ***H*** | • |
| \*: Depends on instruction returned by interrupting device during acknowledge cycle. | | | | |

**6.3 TRAPS**

The Z280 CPU architecture supports eight types of traps, all of which are generated internally in the MPU. The Privileged Instruction, System Call, Access Violation, and Division Exception traps cannot be disabled. I/O instructions can be spec­ified as privileged instructions in the Trap Con­trol register. The Extended Instruction, System Stack Overflow Warning, Single-Step, and Breakpoint-on-Halt traps can be selectively enabled or disabled in the Trap Control register and MSR.

Traps are processed by saving the current program status (PC and MSR) on the system stack and load­ing new program status from the Interrupt/Trap Vector Table, in a manner similar to interrupts using interrupt mode 3. The current interrupt mode has no effect on trap processing. Thus, the Interrupt/Trap Vector Table must be present in memory and the Interrupt/Trap Vector Table Pointer in the CPU must be initialized before executing any instruction that could generate a trap. Traps can occur only if executing Z280 MPU instructions that are not part of the Z80 CPU instruction set or if trap-generating features of the Z280 CPU (such as stack overflow warnings) have been explicitly enabled.

* + 1. **Extended Instruction Trap**

The Extended Instruction trap occurs when the Z280 CPU encounters an extended instruction while the EPU Enable bit in the Trap Control register is a zero. For instructions that transfer data betweer an EPU and memory, the following information is pushed onto the system stack when processing the Extended Instruction trap: the address of the next instruction, the MSR, the address of the memory operand, and the address of the template portion of the extended instruction (in that order). For Load Accumulator from EPU and EPU Internal Opera­tion instructions, the address of the next instruction, the MSR, and the address of the tem­plate in the extended instruction are saved. The PC and MSR values for the service routine are then loaded from the Interrupt/Trap Vector Table. The Interrupt/Trap Vector Table contains four dif­ferent entries for Extended Instruction traps, one for each type of extended instruction.

The Extended Instruction trap allows the progra™ to simulate (in software) the operation of an in a trap service routine when no EPUs are present in the system.

**£•3\*2 Privileged Instruction Trap**

The Privileged Instruction trap occurs Z280 CPU encounters a privileged instructiun in the user mode (the User/System bit is set to 1). i/o instructions can be priv\* instructions, depending on the contents Trap Control register. The following inf13^ a is saved on the system stack when proces^ Privileged Instruction trap: the address instruction causing the trap and the MSR order).

Ihe Privileged Instruction trap protects the ating sy3tem environment by preventing U8erT? programs from executing instructions that disrupt the system. 'Ould

**6 Systee Call Trap**

Tbe System Call trap occurs whenever a System Call instruction is executed. The following informs ti0n is saved on the system stack when processing a System Call trap: the address of the next instruction, the MSR, and the 16-bit immediate operand encoded in the System Call instruction (in that order).

The System Call trap provides a means by which a user mode program can request an operating system function, thereby allowing for an orderly transi- tion between the user and system modes.

* 1. **Л Access Violation Trap**

Warning bit in the Trap Control register is auto­matically cleared to 0 when this trap occurs in order to prevent repeated traps.

The System Stack Overflow Warning trap notifies the operating system of potential stack overflow problems.

* + 1. **Division Exception Tr^»**

The Division Exception trap occurs while executing a Divide instruction if the divisor is zero (divide by zero case) or the quotient cannot be represented in the destination precision (over­flow case); the CPU flags are set to distinguish between these two situations (see the descriptions for the Divide instructions in Chapter 3). The following information is saved on the system stack when processing a Division Exception trap: the address of the Divide instruction and the MSR (in that order).

The Access Violation trap occurs whenever the Z280 MPU’s on-chip MMU detects an illegal memory access. Specifically, this trap occurs when the ffflj’s translation mode is enabled and either the address to be translated implies using a page descriptor register whose Valid bit is zero or the access is a write to a page whose Write-Protect bit is set to 1. The following information is saved on the system stack when processing an Access Violation trap: the address of the instruc­tion causing the trap and the MSR (in that order). Information about the logical address that caused the fault is saved in the MMU (see Chapter 7).

* + 1. **Single-Step Trap**

Two control bits in the Master Status register are used to control Single-Step traps: the Single-Step bit (bit 8) and the Single-Step Pending bit (bit 9). The Single-Step trap occurs when the Single-Step Pending bit in the MSR is set to 1. To enter single-step mode, wherein a Single-Step trap is executed after each instruction, the Single-Step bit in the MSR is set to 1. At the beginning of instruction execution, the state of the Single-Step Pending bit is checked; if it is set, a Single-Step trap is executed. Then, the state of the Single-Step bit is copied into the

Single-Step Pending bit and the instruction is

The Access Violation trap facilitates the imple­mentation of virtual memory systems using the Valid bit in the page descriptor registers and allows information in memory to be write- protected.

executed. If the instruction generates another trap (such as a Privileged Instruction trap), that trap handling routine is executed before the Single-Step Pending bit is again checked and the Single-Step trap is processed. This execution sequence is illustrated in Figure 6-2. Note that once the Single-Step bit gets set, a Single-Step

**4\*3.5 System Stack Overflow Warning Trap**

lhe System Stack Overflow Warning trap can occur if the Stack Overflow Warning bit in the Trap Contr°l register is set to 1. If so, then for PU8h to the system stack, the 12 most signif- \*Cant bits of the Stack Pointer are compared to ?\* contents of the Stack Limit register and a

8,5 is generated if they match. The following

Qf»ation js saved on the system stack when (^e88in9 a System Stack Overflow Warning trap ho second System Stack Overflow Warning trap ’ Crated): the address of the next instruction the «SR (in that order). The Stack Overflow

trap does not occur until after the next instruction, because the Single-Step Pending bit is checked before being loaded with the state of the Single-Step bit. Single-Step traps are then executed after each instruction until the Single-Step bit in the MSR is cleared to 0.

The Single-Step Pending bit in the MSR is automat­ically cleared by a Division Exception, Access Instruction,

Violation,

Privileged

Breakpoint-on-Halt trap,

ao that the saved

a result of

value put on the stack as processing will have a 0 in bit position 9 each of those trap types, the address of

or

MSR trap

For the

CLEAR  
SINGLE-STEP  
PENDING BIT

EXCEPTION  
PROCESSED

YES

START

SINGLE-STEP TRAP

YES

NO

IS

SINGLE-STEP­

PENDINGBIT

SET

COPY SINGLE­STEP BIT INTO SINGLE-STEP­PENDING BIT

EXECUTE INSTRUCTION

The Single-Step trap facilitates the debugging of Z280 CPU code. The following text explains four methods for entering single-step operations.

1. PUSH a PC value for the instruction you wish to jump to.

PUSH an MSR value with the desired combination of ’ the Single-Step (SS) and Single-Step Pending

(SSP) bits.

. Execute and RETIL instruction.

1. Execute a LDCTL instruction with the desired combination of the SS and SSP bits.

NO

OTHER

EXCEPTION

Figure 6-2. Instruction Execution Sequence

c. Execute a System Call (SC) with an identifier that you reserve for a single-step entry.

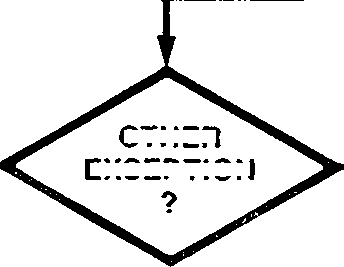
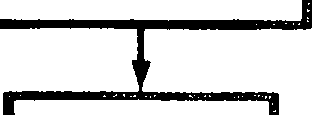
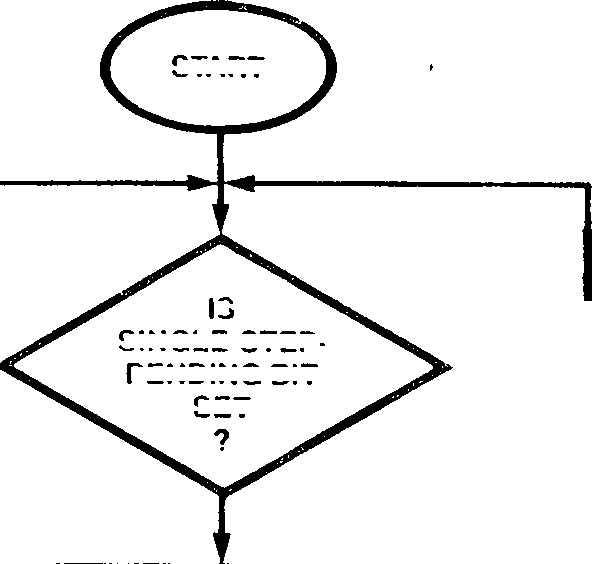
POP the identifier and branch to the remaining single-step code routine.

. POP the MSR.

Set the desired combinations of SS and SSP.

PUSH the new MSR.

Execute the RETIL instruction.



This method can be used only in the User Mode of

actual trapping instruction is saved on the stack (as opposed to the address of the next instruction). The trapping instruction can be re-executed upon returning from the trap service routine, in which case another Single-Step trap is not desired before instruction execution. Similarly, the Single-Step Pending bit is automatically cleared by a Single-Step trap, to ensure that only one Single-Step trap occurs per instruction.

When executing a Return from Interrupt Long (RETIL) instruction to return from an interrupt or trap service routine, the Single-Step Pending bit in the MSR for the interrupted program is the OR of the Single-Step Pending bit in the MSR of the service routine and the Single-Step Pending bit in the MSR value that was saved during trap proces­sing. Thus, if the service routine was being exe­cuted in single-step mode, a Single-Step trap occurs after execution of the RETIL instruction, before resumption of the interrupted program.

operation.

d. Use the "Breakpoint-on-HaIt" trap by substituting a HALT opcode for the first byte of an instruction where single-stepping is to start. The trap service routine should look something like this:

POP the MSR.

Set the desired combinations of SS and SSP.

PUSH the MSR.

Restore the instruction byte that the HALT opcode replaced.

Execute the RETIL instruction.

Both interrupt and trap routines can be single-stepped by setting the appropriate SS and SSP combination in the MSR entry in the Interrupt/Trap Vector Table.

Instructions that cause a trap but will be re-executed (ie: privileged, divide, page fault) automatically clear the SSP bit in the PUSHed MSR. This ensures that only one single-step trap will occur for these instructions.

The following information is saved on the system stack when processing a Single-Step trap: the address of the next instruction and the MSR (in that order).

**Trap Type**

**Table 6-3. Trap Types**

**Can be  
Disabled**

**Status Saved**

Extended Instruction Yes

Privileged Instruction No

[System Call No](#bookmark335)

Access Violation No

System Stack Overflow Yes

Division Exception No

Single-Step Yes

Address of next instruction

MSR value

Address of operand in memory (if applicable)

Address of EPU template

Address of instruction causing trap •

MSR value

Address of next instruction

MSR value

16-bit reason code from SC instruction

Address of instruction causing trap

MSR value

Address of next instruction

MSR value

Address of instruction causing trap

MSR value

Address of next instruction

MSR value

Breakpoint-on-Halt

Yes Address of Halt instruction

MSR value

Table 6-4. Interrupt Acknowledge Encoding for Z80 Bus Configuration

|  |  |  |
| --- | --- | --- |
| ad2 | **AD1** | **Interrupt Being Acknowledged** |
| **0** | **0** | Interrupt A |
| **0** | **1** | Nonmaskable Interrupt |
| **1** | **0** | Interrupt В |
| **1** | **1** | Interrupt C |

6.3.8 Breakpoint-on-Halt Trap

The Breakpoint-on-Halt trap occurs if a Halt instruction is encountered while the Breakpoint- on-Halt Enable bit in the MSR is set to 1. The following information is saved on the system stack when processing a Breakpoint-on-Halt trap: the address of the Halt instruction and the MSR (in that order).

The Breakpoint-on-Halt trap provides a breakpoint 1 facility that is useful in debugging environments

in which breakpoints on instruction boundaries are desired.

The trap types and the status saved during the processing of each trap are summarized in Table 6-3.

* 1. INTERRUPT AND TRAP HANDLING

The Z280 CPU response to an interrupt request or trap condition consists of up to five steps: acknowledging the external request (externally- generated interrupts only), saving current program status, loading new program status, executing the service routine, and returning to the interrupted program. Interrupts are accepted and processed between instructions, with the exception of the block move, search, and I/O instructions, which can be interrupted between any iteration. Traps are detected during instruction execution, with the exception of the Single-Step trap, as described previously. Thus, a trap condition is processed before handling any pending interrupts.

* + 1. Interrupt Acknowledge

An interrupt acknowledge bus transaction is required only for externally-generated inter­rupts. The main effect of the interrupt acknowl­edge is to establish communication between the requestor and the Z280 CPU.

For Z80 Bus configurations of the Z280 MPU, the type of interrupt being acknowledged is indicated on bus lines AD'i and AO2 while the Address Strobe is being asserted during the interrupt acknowledge cycle, as per Table 6-4.

For the Z80 Bus configurations of the Z280 MPU, no external acknowledge cycle is generated for nonmaskable interrupts in interrupt modes 0, 1, and 2, or for maskable interrupts in interrupt mode 1. For maskable interrupts in interrupt modes 0, 2, and 3, and for nonmaskable interrupts in mode 3, 8-bit data is read from the ADq-AD? bus lines during the acknowledge cycle; this data is used as dictated by the interrupt mode in effect, as described in section 6.2. For maskable interrupts in interrupt mode 0, successive bytes are read on ADQ-AD7 until a complete instruction has been fetched, via repetition of the acknowledge cycle.

For Z-BUS configurations of the Z280 MPU, any interrupt from an external source is acknowledged. The type of interrupt being acknowledged is indicated by the STq-STj status lines during the acknowledge cycle. A word of data is read from the address/data bus during the acknowledge cycle and used as dictated by the interrupt mode in effect. For interrupt modes 2 and 3, the lower byte of this data is used as the interrupt vector. For maskable interrupts in interrupt mode 0, successive bytes are read on ADg-ADy until a complete instruction has been fetched, via repetition of the acknowledge cycle.

Acknowledge cycles are always executed in system mode, regardless of the mode of the interrupted program. The MSR of the interrupted program is not affected by this change in mode. The CPU stays in system mode until the start of execution of the service routine. In interrupt modes 0, 1, and 2, the service routine starts in system mode; in interrupt mode 3, the MSR of the service rou­tine is determined by the contents of the Inter- rupt/Trap Vector Table.

Interrupt requests from the on-chip peripherals never generate an acknowledge cycle and are always processed using interrupt mode 3. Similarly, traps do not generate acknowledges.

* + 1. Status Saving

During exception processing, the status of the interrupted program is saved on the system stack. In interrupt mode 0, the Program Counter is auto­matically saved when processing nonmaskable inter­rupts; the instruction returned by the peripheral device will determine what status information is saved when processing maskable interrupts. For interrupts in interrupt mode 1 or 2, the Program Counter is automatically saved. For interrupts in interrupt mode 3, the Program Counter and MSR of the interrupted task are saved, followed by the "reason code" (Figure 6-3). For external inter­rupt requests, the reason code is the value read from the data bus during the interrupt acknowledge cycle; the upper byte of the reason code is all zeros for 8-bit data bus (Z80 Bus) configurations of the Z280 MPU. For interrupts from the on-chip peripherals, the reason code is the offset address in the Interrupt/Trap Vector Table that corresponds to the MSR value entry for that interrupt type.

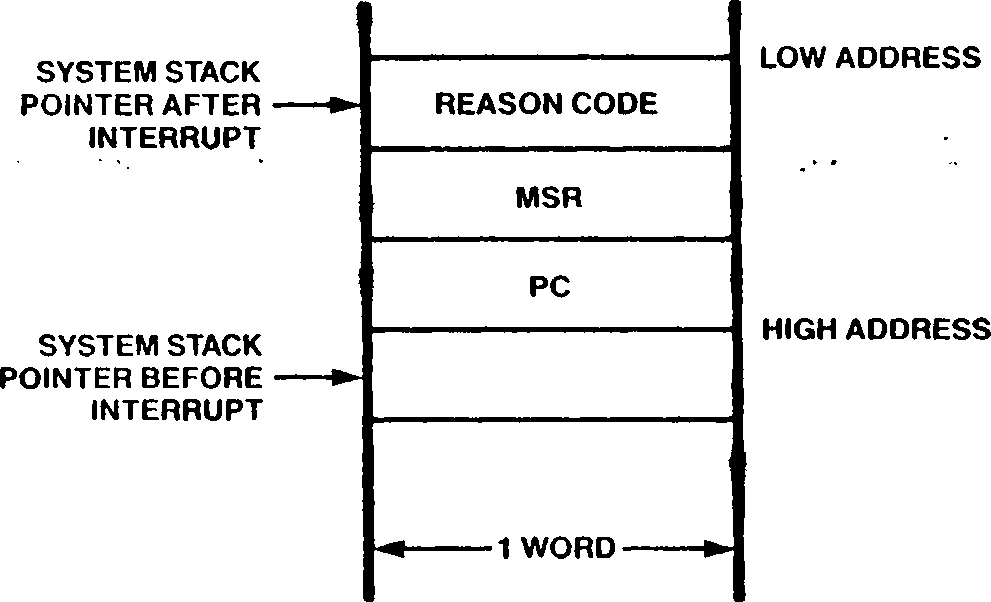


Figure 6-3. Format of Saved Status on System Stack Due to a Mode 3 Interrupt

The Program Counter value saved during interrupt processing is the address of the next instruction in the interrupted routine, except for interrupts during block move, block search, and block 1/0 instructions. The block instructions can be interrupted between any one iteration of their operation, in which case the PC value saved is the address of the block instruction itself.

The status saved as a result of a trap depends on the type of trap being executed, as noted in Figure 6-3. The PC and MSR values are always saved during trap processing, along with other trap-dependent information. '

If any memory write operation involved in saving status information during interrupt or trap proc­essing causes a memory access violation, a special "fatal condition" is entered, as described in sec­tion 6.6.

* + 1. Loading New Program Status

After saving the status of the interrupted pro­gram, new program status values (i.e., new values for the PC and MSR) are automatically loaded, in accordance with the interrupt mode and any data read during the acknowledge cycle. This new pro­gram status determines the operating modes and starting address of the service routine.

For externally generated interrupts in interrupt modes 0, 1, and 2, the Master Status register is automatically modified to specify system mode with the Single-Step trap and all maskable interrupts disabled. For externally generated interrupts in interrupt mode 3, all internally generated inter­rupts, and all traps, the new MSR value is loaded from the Interrupt/Trap Vector Table.

For externally generated maskable interrupts proc­essed using interrupt mode 0, the first instruc­tion of the service routine is supplied by the interrupting device. This must be a Call or

Restart instruction that loads the PC with the starting address of the service routine. For non­maskable interrupts in interrupt mode 0, the PC is set to 0066ц, and all maskable interrupts are automatically disabled.

In interrupt mode 1, the PC is set to 0038ц for externally generated maskable interrupts and to 0066ц for nonmaskable interrupts.

be translated to a physical address by the MMU when the CPU fetches the first instruction of the service routine.

* + 1. Executing the Service Routine

In interrupt mode 0, the interrupting device pro­vides the Restart or Call instruction that begins the service routine; this instruction saves the Program Counter value of the interrupted routine and provides the address of the service routine. In the other interrupt modes and for traps, the starting address of the service routine is deter­mined automatically during interrupt processing, as described in the preceding section. This pro­gram is now executed.

For externally generated interrupts in interrupt modes 0, 1, and 2, all maskable interrupts are automatically disabled; therefore the service rou­tine is protected from additional interrupts until the MSR is altered via a Load Control, Enable Interrupt, Return from Nonmaskable Interrupt, or Return from Interrupt Long instruction. Inter­rupts in mode 3 and all traps cause a new MSR to be loaded from the Interrupt/Trap Vector Table; the value of this MSR determines which interrupts are enabled during the service routine. Service routines that enable interrupts before exiting permit interrupts to be handled in a nested fash­ion.

For externally generated maskable interrupts in

interrupt mode 2, the PC is fetched from an Inter­rupt Vector table in system data memory; the logi­cal address of the fetched PC value is formed by concatenating the contents of the I register with the 8-bit vector returned by the interrupting device during the acknowledge cycle. For nonmask­able interrupts, the PC is set to 0066ц. :

For externally generated interrupts in interrupt mode 3, all internally generated interrupts, and all traps, the PC and MSR values for the service routine are fetched from the Interrupt/Trap Vector Table (see section 6.5). The new value for the MSR is at a fixed location in this table. Exter­nally generated interrupts can be vectored or nonvectored in interrupt mode 3, as determined by the contents of the Interrupt Status register. For nonvectored interrupts and all traps, the new PC value is at a fixed location in the Inter­rupt/Trap Vector Table; for vectored interrupts, the location of the new PC in the table is depen­dent on the 8-bit vector read during the acknowl­edge cycle.

The value loaded into the Program Counter during exception processing is a logical address that can

* + 1. Returning from a Service Routine

Three different instructions are available for returning from an interrupt or trap service rou­tine: Return from Nonmaskable Interrupt, Return from Interrupt, and Return from Interrupt Long. All three are privileged instructions, since they must retrieve values from the system stack.

The Return from Nonmaskable Interrupt (REIN) instruction is used to return from nonmaskable interrupts in interrupt modes 0, 1, and 2. This instruction pops the word on the top of the stack into the Program Counter, restoring the Program Counter value present before the interrupt, and loads the Interrupt Request Enable bits in the MSR with the contents of the Interrupt Shadow regis­ter.

The Return from Interrupt (RETI) instruction is used to return from externally generated maskable interrupts in interrupt modes 0, 1, and 2. This instruction pops the word on the top of the stack into the Program Counter, which restores the Pro­gram Counter value present before the Interrupt. The RETI instruction also causes a special bus

transaction that fetches this instruction from external memory (regardless of whether it is con­tained in the on-chip cache), with the appropriate bus control and status signals to indicate that an instruction fetch is occurring; this is used to reset the interrupt logic of the Z80 family peripherals.

The Return from Interrupt Long (RETIL) instruction is used to return from interrupts in interrupt mode 3 and all traps, since it causes both the MSR and PC values to be popped from the stack. If this instruction is used to return from an inter­rupt processed with another interrupt mode (e.g., if RET1L is used to return from a mode 2, instead of a mode 3, interrupt), an MSR value must be pushed onto the stack in the service routine prior to execution of the RET IL. For interrupts in interrupt mode 3 and all traps, the service — rout ine must pop the reason code or other trap-dependent information off the stack before executing RE I IL. Unlike RETI, RET IL causes no special bus activity and, therefore, cannot be used to automatically reset Z80 family periph­erals.

request is enabled (bit 1 in the MSR is set to 1), the interrupt is processed as follows: the current PC and MSR values are saved on the system stack; an identifier word with the value 14^ is saved on the system stack; a new value for the MSR is fetched from location 14^ in the Interrupt/Trap Vector Table; a new value for the PC is fetched from location 16^ in the Interrupt/Trap Vector Table; execution of the service routine is begun.

If an interrupt reguest is received from an external source on interrupt line A under interrupt mode 3 and that interrupt request is enabled (bit 0 in the MSR is set to 1), then interrupt processing proceeds as follows:

• An acknowledge cycle is executed, during which data is read from the external data bus.

« The current PC and MSR values are saved on the

• The data read from the bus during the

system stack

acknowledge cycle is saved on the system stack

as the identifier word.

• A new value for the MSR is fetched from location 08^ in the Interrupt/Trap Vector Table

6.5 INTERRUPT/TRAP VECTOR TABLE

The Interrupt/Trap

During interrupt processing under interrupt mode 3 and all trap processing, the PC and MSR values that determine the starting location and operating modes of the appropriate service routine are fetched from a table in memory called the Inter­rupt/Trap Vector Table. This table holds an MSR and PC value for the service routine for every possible type of interrupt and trap. The particu­lar values fetched from the table during exception processing are a function of the type of exception that occurred and, for vectored external inter­rupts, the vector returned by the peripheral dur­ing the acknowledge cycle. The format of the Interrupt/Trap Vector Table is given in Table 6-5. Each entry in the Interrupt/Trap Vector Table consists of two words—an MSR value followed by a PC value. If an external interrupt is vec­tored, as determined by the contents of the Inter­rupt Status register, the 8-bit vector returned by the peripheral is used as an index into a list of up to 128 possible PC values for the service routine; only even-valued vectors are supported by the Z280 CPU architecture. Thus, for a vectored interrupt, there is only one starting MSR value for all the possible service routines, but up to 128 potential PC values. The NMI and Interrupt A requests share the same vectors.

**e** A new value for the PC is fetched either from location 0A in the Interrupt/Trap Vector Table (if bit 13 of the Interrupt Status register is 0, indicating that Interrupt A is not vectored) or from the location in the Interrupt/Trap Vector Table found by adding the lower byte of the data read from the bus during the acknowledge cycle (the interrupt vector) to (if bit 13 of the Interrupt Status register is 1, indicating that Interrupt A is vectored).

• Execution of the service routine is begun.

Fur vectored interrupts, the interrupt vector returned during the acknowledge cycle must be even-valued in order to reference a valid PC value in the Interrupt/Trap Vector Table.

The Interrupt/Trap Vector Table Pointer register must be initialized to hold the must significant 12 bits of the starting physical address of the

Interrupt/Trap Vector Table.

Vector Table must start on a 4K byte boundary in physical memory (that is, a memory address whose 12 least significant bits are all zeros).

For example, suppose an interrupt is requested by

the on-chip counter/timer 0. If that interrupt

**вжав**

**яа**

Table 6-5. Interrupt/Trap Vector Table Format

Address in Table (Hexadecimal) Contents

00 Reserved

04 NMI vector

08 Interrupt line A vector

0C Interrupt line В vector

10 Interrupt line C vector

14 Counter/Timer 0 vector '

18 Counter/Timer 1 vector

1C Reserved

20 Counter/Timer 2 vector

24. DMA channel 0 vector

28 DMA channel 1 vector

2C DMA channel 2 vector

30 DMA channel 3 vector '

34 UART receiver vector

38 UART transmitter vector

3C Single-Step trap vector ■

"" ' ' 40 ' ' ’ Breakpoint-on-Halt trap vector " ' " '

44 Division Exception trap vector

48 Stack Overflow Warning trap vector

4C Access Violation trap vector

1. System Call trap vector .
2. Privileged Instruction trap vector
3. EPU \*- Memory Extended Instruction trap vector

5C Memory <- EPU Extended Instruction trap vector

1. A ♦- EPU Extended Instruction trap vector
2. EPU Internal Operation Extended Instruction trap vector

68-6C Reserved

70-16E 128 Program Counter values for NMI and interrupt line A vectors (MSR values from position 04 and

08 in this table, respectively)

170-26E 128 Program Counter values for interrupt line В (MSR value from position 0C in this table)

270-36E 128 Program counter values for interrupt line C (MSR value from position 10 in this table)

6.6 THE FATAL CONDITION

During interrupt and trap processing, the CPU automatically attempts to save status information about the interrupted program on the system stack. If the MMU is enabled, an access violation can occur during the status saving process if a write is attempted to an invalidated page or to a page that is write-protected. Detection of an access violation during the status saving process dition; the following steps are taken automati­cally when the fatal condition occurs: the current PC contents are written to the HL register, the current MSR contents are written to the DE regis­ter, all the Interrupt Request Enable bits in the MSR are cleared to 0, and the CPU enters a Halt state. This Halt state is identical to the Halt state caused by the execution of a Halt instruc­tion, with one exception: a Halt state induced by a fatal condition can be exited only by a reset.

causes the Z280 CPU to enter a special fatal con­

Chapter 7.

Memory Management Unit

7.2 MMU ARCHITECTURE

7.1 INTRODUCTION

The Z280 MPUs include an on-chip paged Memory Man­agement Unit (MMU), which allows the MPUs to address more than 64K bytes of physical memory. Memory management with the MMU involves two issues: memory allocation and memory protection. The allocation of memory is controlled by allowing the MMU to translate the 16-bit logical addresses from the Z280 CPU into the 24-bit physical addresses output by the MPU. Thus, a given programming task can be relocated to any area of physical memory, regardless of the logical addresses used by that task. During this translation process, the MMU also monitors the type of memory access being made; the WU can inhibit accesses or write-protect memory areas, thereby allowing memory to be protected from unwanted or unintended modes of use.

The Z280 MMU consists of two sets of 16 page descriptor registers, used to translate addresses and assign memory attributes on a page-by-page basis, and a Master Control register that governs MMU operation. There is one page descriptor reg­ister associated with each logical page of mem­ory. One set of 16 page descriptor registers is dedicated to system mode operation and the other set to user mode operation. The MMU registers are accessed using 1/0 instructions.

When translation is enabled for a particular mode (system or user), as determined by the contents of the MMU Master Control register, the MMU trans­lates memory addresses whenever the CPU is operat­ing in that mode, using the set of page descriptor registers dedicated to that mode. However, there are two exceptions to that rule:

The MMU partitions the 64K logical address space

have been modified

of the Z280 CPU into fixed-sized memory pages and maps those pages into the physical address space. Separate mapping facilities are available for the system and user modes of operation; translation can be performed in either one or in both modes. Optionally, the MMU provides for separating instruction fetches from data references, which allows the user to define up to four different logical address spaces: system mode program, sys­tem mode data, user mode program, and user mode data. If the program and data address spaces are separated, the MMU uses a page size of 8192 (8I<) bytes; if not, the page size is 4096 (4K) bytes.

The MMU is programmed via I/O references to its control registers. The MMU records which pages

and can inhibit the cache mech­anism to prevent the writing of data to the on-chip cache. Access Violation traps are gener­ated when an error condition is detected (such as an attempted write to a read-only page). Access violations cause the currently executing instruc­tion to be aborted, and allow that instruction to be restarted in a manner compatible with virtual memory requirements. Upon reset, the MMU is dis­abled, allowing logical addresses to pass through to physical memory without translation.

$ When the CPU is fetching program status infor­mation from the Interrupt/Trap Vector Table in response to an interrupt under interrupt mode 3

or a trap, the Interrupt Trap Vector Table Pointer register is used to determine the phys­

ical address of the program status information.

The Load in User Program (LDUP) and Load in User Data (LDUD) instructions are executed in

system mode but use the user mode page descrip­tor registers to translate the data operand’s address.

Memory addresses generated by the on-chip DMA channels are 24-bit physical addresses that are not translated by the MMU. Only memory addresses, and not 1/0 addresses, are translated by the MMU.

While an address is being translated, any attri­butes associated with the logical page containing that address are checked. The attributes for a page are determined by the contents of that page’s page descriptor register. Pages can be write- protected and/or made non-cacheable using these attributes. A non-cacheable page is one whose contents cannot be copied into the on-chip cache during program execution; thus, accesses to loca­tions in non-cacheable pages always use the exter­nal bus. This attribute is useful in multiproces­sor systems with shared memory areas, where each processor must be able to access the most current version of the information in the shared memory area, or in systems with memory-mapped I/O devices. The MMU also maintains a status bit for each page, which indicates if that page has been modified, t

Each page descriptor register contains a Valid bit, which indicates if that descriptor contains valid information. Attempts to access an address contained in a page with an invalid descriptor and attempts to write to an address in a page that is write-protected generate Access Violation traps. An Access Violation trap causes the currently exe­cuting instruction to be aborted, facilitating the development of virtual memory systems. A special 1/0 port in the MMU (Invalidation 1/0 port) is available for resetting the valid bits in a whole group of page descriptor registers with a single I/O instruction.

For system mode operation, user mode operation, or both, the MMU can be configured to separate instruction fetches from data fetches, therefore separating the program address space from the data address space. This allows a Z280 MPU program to contain up to 64K bytes of code and operate on up to 64K bytes of data. With the program/data sep­aration mode in effect, the 16 page descriptor registers for that mode are partitioned into two sets of eight descriptors: one set for instruction fetches and one set for data fetches. An instruc­tion fetch or data reference using the PC relative addressing mode is translated using the page descriptor registers associated with the program address space; data accesses using other addres­sing modes and accesses to the interrupt vector table under interrupt mode 2 use the page descrip­tor registers associated with the data address space. In this mode, pages аге 8K bytes long. Two control bits in the MMU Master Control regis­ter specify independently whether program/data separation is in effect for system mode and whether program/data separation is in effect for user mode.

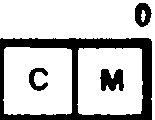
When translation is disabled for a particular mode (system or user), the MMU does not translate mem­ory addresses or perform attribute checking while the CPU is operating in that mode. For a memory access when the MMU is disabled, the logical mem­ory address passes through the MMU without trans­lation to physical address outputs Ag-A^ and physical address outputs Aj$-A23 are all zeros. When the MMU is disabled all memory is assumed to be both writeable and cacheable.

* 1. PAGE DESCRIPTOR REGISTERS

There are two sets of 16 page descriptor registers in the MMU, one set for system mode operation and one set for user mode operation. Each page descriptor register is 16 bits long, consisting of **a** 12-bit page frame address field and a 4-bit attribute field (Figure 7-1).,

**15 ’ I I I I I I I**

PAGE FRAME ADDRESS ■ ± 1. 1 1 I 1 -A —A..J

Figure 7-1. Page Descriptor Register

WP

The page frame address field contains the most significant 12 bits (if prugram/data separation is not in effect) or most significant 11 bits (if program/data separation is in effect) of the starting physical address for that page. The low- order bits of the page’s base physical address are assumed to be all zeros; thus, pages always start on 4K byte boundaries in physical memory without program/data separation, or 8K byte boundaries with program/data separation.

The least significant four bits of each page descriptor register are attribute and status bits for that page, as described below:

**Modified Bit (M).** This status bit is automati­cally set to 1 whenever a write is successfully performed to a logical address in the page; it can be cleared to 0 only by writing to the page descriptor register via a software command. If the Valid bit is 0, the contents of this bit are undefined.

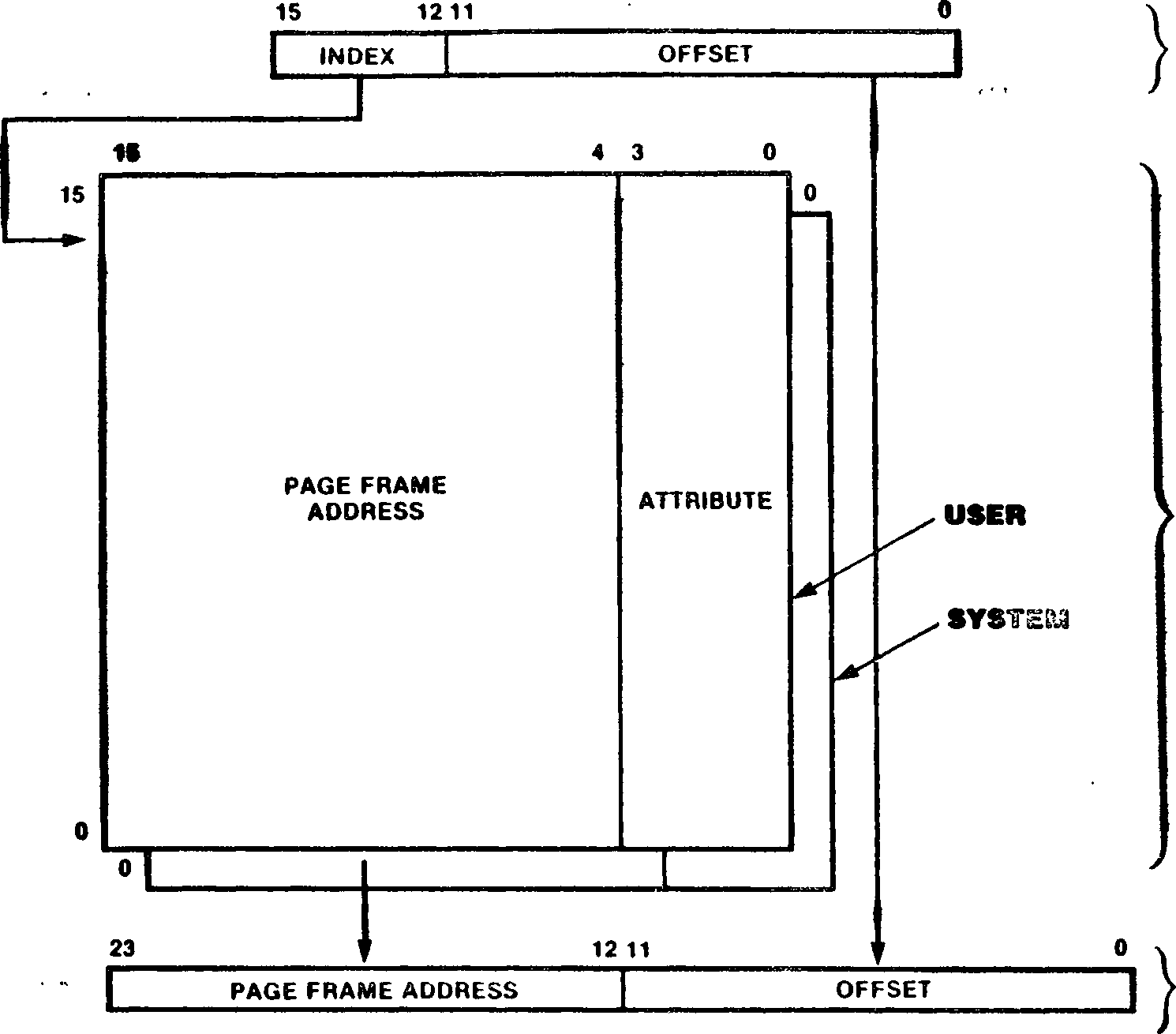
**Cacheable Bit (C).** When this bit is set to 1, information from the page can be stored in the on-chip cache memory. When this bit is cleared to **0t** the cache control mechanism is inhibited from retaining a copy of information from the page.

**Write-Protect Bit (WP).** When set to 1, write operations to addresses in the page generate an Access Violation trap and the write is inhibited. When this bit is cleared to 0, all valid accesses to the page are allowed.

**Valid Bit (V).** This bit is set to 1 to indicate that the page descriptor register contains valid information about the page. When cleared to 0, all accesses to addresses in the page are inhibited and generate Access Violation traps.

**Э№ЗЛС**

* 1. ADDRESS TRANSLATIONand a 12-bit offset field that forms the lower 12



LOGICAL

ADDRESS

DADE DESCRIPTOR REGISTERS

PHYSICAL ADDRESS

Figure 7-2. Address Translation without Program/Data Separation

If address translation is enabled, logical addresses are translated to physical addresses in one of two ways, depending on the program/data separation mode, as specified in the MMU Master Control register. The format of the page descrip­tor registers is independent of which mode is in effect.

* + 1. Address Translation Without Program/Data

Separation

When program/data separation is not in effect, the 16-bit logical address from the CPU is divided into two fields, a 4-bit index field used to select one of the 16 page descriptor registers,bits of the resulting physical address. The upper 12 bits of the physical address are provided by the page frame address field of the selected page descriptor register. The pages аге 4K bytes long. This translation mechanism is illustrated in Figure 7-2. Page descriptor register 0 is the descriptor for logical addresses 0000ц to OFTF^, page descriptor register 1 is the descriptor for logical addresses 1000ц to 1FFF|\_|, and so on. Thus, the index portion of the logical address selects the page descriptor register. The page frame address field of that page descriptor register then determines the actual starting address for that page in physical memory; the low-order 12 bits of the logical address specify the offset within that 4K byte

page.

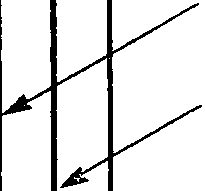
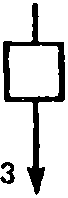
* + 1. Address Translation With Program/Data Separation

When prograiVdata separation is in effect, the 16-bit logical address from the CPU is divided into a 3-bit index and a 13-bit offset. A Pro- gram/Data address control signal from the CPU becomes the most significant bit of the 4-bit index that selects the appropriate page descriptor register; the three most significant bits of the logical address form the least significant bits of this index. The upper 11 bits of the page frame address field in the selected page descriptor reg­ister provide the upper 11 bits of the resulting physical address. The least significant 13 bits of the logical address form the low order 13 bits of the physical address, as illustrated in Figure 7-3. Page descriptor register 0 is the descriptor for logical addresses OOOOH-1FFFH in the data address space, Page descriptor register 1 is the descriptor for logical addresses 2OOOp|-3FFF|^ in the data address space, and so on through page descriptor register 7; page descriptor register 8 is the descriptor for logical addresses OOOO^-1FFF^ in the program address space, page descriptor register 9 is the descriptor for logi­cal addresses 2OOO^-3FFFh in the program address space, and so on. Thus, each page is 8K bytes long, where the starting address of the page in physical memory is determined by the page frame address field in the selected page descriptor reg­ister, and the 13 least significant bits of the logical address specify the offset within that 8K byte page. In this mode, the least significant bit of the page frame address field in each page descriptor register is not used; this bit is modi­fied by translation, and values read from it are unpredictable.

***•• • -t ‘ .*** ■ ч, • • ' •

PROGRAM/

DATA BIT



15

13 12

INDEX

OFFSET

16-BIT LOGICAL PROGRAM

OR DATA ADDRESS

INDEX

| **PROGRAM PAGE FRAME ADDRESS** | **0**  **0**  **0**  **0**  **0**  **0**  **0**  **0** | **ATTRIBUTE** |
| --- | --- | --- |
| **DATA PAGE FRAME**  **• ADDRESS** | **0 0**  **0**  **0**  **0**  **0**  **0**  **0** | **ATTRIBUTE** |

15

0

15

0

8

0

0

0

23

5 4 3

13 12

PROGRAM PAGE

DESCRIPTOR REGISTERS

PAGE FRAME ADDRESS

SYSTEM .

**USER**

DATA PAGE

DESCRIPTOR REGISTERS

OFFSET

24-BIT PHYSICAL PROGRAM OR DATA ADDRESS

Figure 7-3. Address Translation with Program/Data Separation

**PDR Pointer or  
PFI Field**

OF

10

Besides the two sets of 16 page descriptor regis­ters, the MMU contains a Master Control register and a Page Descriptor Register Pointer. The 16-bit Master Control register controls the opera­tion of the MMU; the 8-bit Page Descriptor Regis­ter Pointer is used to select a particular page descriptor register during 1/0 accesses to the descriptors.

The 16-bit MMU Master Control register is shown in figure 7-4. This register consists of four con­trol bits and a 5-bit status field; the fields in this register are described below:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **lUTE** | **UPD** | **1** | **1** | **STE** | **SPD** | **1** | **1** | **1** | **1** | **1** | **i I I I П**  **PFI I**  **1 L l I I** |

15

0

Figure 7-4. MMU Master Control Register

**User Mede Translate Enable (UTE).** When this bit is set to 1, logical memory addresses generated during user-mode operation are translated to phys­ical addresses with attribute checking. When this bit is cleared to 0, the logical addresses are passed through the MMU to the address outputs with zeros in the most significant bits and no attri­bute checking or modified bit setting is per­formed.

User Mode Proqram/Data Separation Enable (UPD).

When this bit is set to 1, instruction fetches and data accesses using the PC Relative addressing mode use user-mode Page Descriptor registers 8 through 15, and data references using other addressing modes use user-mode Page Descriptor registers 0 through 7; the page size is 8K bytes. When this bit is cleared to 0, both instruction and data fetches use user-mode Page Descriptor registers 0 through 15 and the page size is 4K bytes.

**Syste\* Mode Translate Enable (STE).** When this bit is set to 1, logical memory addresses generated during system-mode operation are translated to physical addresses with attribute checking. When this bit is cleared to 0, the logical addresses are passed through the MMU to the address outputs with zeros in the most significant bits and no attribute checking or modified bit setting is per­formed.

through 15, and data references using other addressing modes use system-mode Page Descriptor registers 0 through 7; the page size is 8K bytes. When this bit is cleared to 0, both instruction and data fetches use system-mode Page Descriptor registers 0 through 15 and the page size is 4K bytes.

**Page Fault Identifier (PFI) Field.** This 5-bit status field latches an identification number that indicates which Page Descriptor register was being accessed when an access violation was detected. The encoding used is given in Table 7-1.

The MMU Master Control register is programmed via a word output instruction to 1/0 port address FFxxFO^ (where ”x” indicates a "don’t care”) and is read via a word input instruction to that same port. A reset clears this register to all zeros, thereby disabling address translation and attri­bute checking in the MMU. Bits 5 through 9, 12, and 13 in this register are not used.

The Page Descriptor registers in the MMU are accessed using the Page Descriptor Register Pointer (PDR Pointer). The 8-bit PDR Pointer con­tains the address of one of the Page Descriptor registers; the encoding is given in TabTe 7-1. The permissible contents of the PDR Pointer are 00^ through 1Fp|. The PDR Pointer is accessed via byte 1/0 instructions to port address FFxxF1H.

**Table 7-1. Page Descriptor Register Addresses**

**Selected Page  
Descriptor Register**

00 User Page Descriptor 0

01 User Page Descriptor 1

User Page Descriptor 14

User Page Descriptor 15

System Page Descriptor 0

System Page Descriptor 1

1E System Page Descriptor 14

1F System Page Descriptor 15

System Mode Program/Data Separation Enable (SPD).

When this bit is set to 1, instruction fetches and  
data accesses using the PC Relative addressing  
mude use system-mode Page Descriptor registers 8

7.6 ACCESSING PAGE DESCRIPTOR REGISTERS

Data is read or written to the Page Descriptor registers via I/O instructions. Three different types of accesses are allowed, each of which is implemented with its own unigue I/O port address.

to that port, as delineated in Table 7-2. When writing to the invalidation port only the least significant four bits are sampled; the upper four bits are nut used. Reading port FFxxF2^ returns unpredictable data.

**Table 7-2. MMU Invalidation Port**

* + 1. Descriptor Select Port

Moves of one word of data to' or from a Page Descriptor register are accomplished through I/O port address FFxxF5^, the Descriptor Select Port. The Paqe Descriptor register accessed is the one addressed by the PDR Pointer; the PDR Pointer itself is unaffected. Any word I/O instruction can be used.

**Data Written to Port FFxxF2 Page Descriptor Registers**

**(Hexadecimal) Invalidated**

**01** System Page Descriptor Registers 0-7

02 System Page Descriptor Registers 8-15

03 System Page Descriptor Registers 0-15

04 User Page Descriptor Registers 0-7

08 User Page Descriptor Registers 8-15

0C User Page Descriptor Registers 0-15

* + 1. Block Move Port

**Register**

FFxxFOh FFxxF1h FFxxF5h FFxxF4|\_j FFxxF2h

Block moves of data into and out of Page Descrip­tor registers are accomplished by word accesses to I/O port address FFxxF4|\_j. Ihe Paqe Descriptor register accessed is the one addressed by the PDR Pointer. Any word 1/0 instruction can be used. After the access, the contents of the PDR Pointer are automatically incremented by one; thus, a sin­gle block I/O instruction can be used to access several successive Page Descriptor registers. For example, if the PDR Pointer is initialized tu 00, the execution of an INIRW instruction to I/O port FFxxF4|\_| causes data from successive Page Descriptor registers starting with user Paqe Descriptor register 0 to be loaded into memory.

For accesses to the Page Descriptor registers using the Descriptor Select port or the Block Move • port, the permissible contents of the PDR Pointer are the addresses for the Page Descriptors given in Table 7-1: 00|\_| to 1F^. Execution of an I/O instruction to ports FFxxF4^ or FFxxF5|^ when the contents of the PDR Pointer are outside of this permitted range will have unpredictable results.

* + 1. Invalidation Port

The Valid bits in the Paqe Descriptor registers can be cleared to 0 via byte writes to 1/0 port address FFxxFZ^^ thereby invalidating the con­tents of the Page Descriptor registers. Individ­ual Valid bits can subsequently be set by writing to individual Page Descriptor registers using the

Descriptor Select port or the Block Move port. The Page Descriptor registers invalidated by a write to port rFxxF2|\_| depend on the data written

The 1/0 port addresses for the MMU registers are listed in Table 7-3.

**Table 7-3. I/O Port Addresses for MMU Control Registers**

**Port**

**Address**

Master Control Register

Page Descriptor Register Pointer

Descriptor Select Port

Block Move Port  
Invalidation Port

Changing an MMU control register or Page Descriptor register does not cause a flush of the CPU instruction pipeline. While an instruction that changes an MMU register is executing, up tu two subsequent instructions can be pre-fetched into the CPU pipeline; execution of these subsequent instructions must have benign results. In other words, when changing an MMU register, up to two subsequent instructions can be fetched before the change to the MMU register is guaranteed to take effect. (However, no data accesses are pre-fetched.) Iherefure, when initially enabling the MMU for address translation, the instruction that enables the MMU and the next two instructions must be in a paqe whose logical addresses are identical to physical addresses (so that it is immaterial exactly when the MMU begins the translation process for those instruction fetches). When altering a page descriptor register while translation is enabled, neither of the next two instructions should reside in the page associated with the Page Descriptor register being changed.

Detection of a page fault (due to an attempted access to an invalidated page) or a write-protect violation (due to an attempted write to a write- protected page) causes the currently executing instruction to be immediately aborted and generates an Access Violation trap. The starting address of the instruction that caused the violation and the current MSR value are automatically saved on the system stack when processing an Access Violation trap. Furthermore, the MMU latches the address of the referenced Page Descriptor register in the PF I Field of the MMU Master Control register whenever a violation occurs•

For most Instructions, the CPU registers are not modified daring the execution of aborted instruc­tions; i.e., their contents are the same as before the aborted instruction began. The exceptions ace the block move, block search, and block I/O instructions; when aborted, the CPU registers are the same as just before the iteration of the instruction in which the violation occurred. In either case, no modification of CPU registers is necessary before restarting the aborted instruc­tion.

The instruction abort mechanism of the Z280 MPU facilitates the implementation of virtual memory in Z280-based systems. In a virtual memory sys­tem, a cleared Valid bit in the Page Descriptor register can be used to indicate when a memory page is not currently mapped into main memory. If an access is attempted to such a page, the instruction is aborted and the Access Violation trap service routine is invoked. The service rou­tine can determine which Page Descriptor register is involved by reading the PF I field of the MMU Master Control register, swap the appropriate page from the secondary storage device into main mem­ory, adjust the appropriate Page Descriptor regis­ters, and then restart the aborted instruction. The aborted instruction is automatically restarted by using the Return from Interrupt Long instruc­tion to retrieve the original PC and MSR values from the system stack. No adjustments to other CPU registers are required. During the swapping process, the modified status bit in the page descriptor register can be used to determine if a page has been modified since the last time it was copied to a secondary storage device.

Chapter 8.

On-Chip Memory

ОИИМИИВИМШНШИ дазяЕдм—ив: вЖ

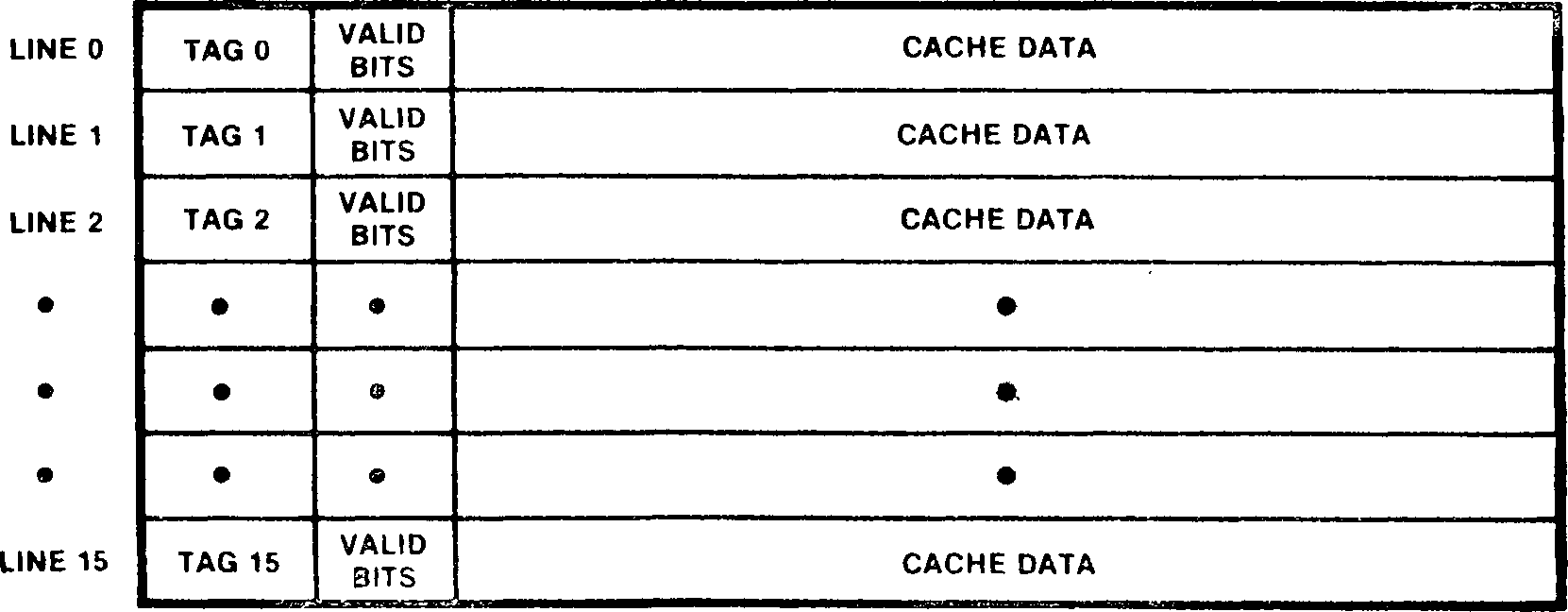
8.1 INTRODUCTION

The Z280 MPU has 256 bytes of on-chip memory. This on-chip memory can operate in either of two modes, as determined by the contents of the Cache Control register (see Chapter 3). In one mode, the on-chip memory is dedicated to fixed physical memory locations; the memory addresses that are mapped into the on-chip memory are determined under program control. In the other mode, the on-chip memory acts as a cache for either instruc­tions, data, or both. When acting as a cache, the set of memory locations mapped into the on-chip memory at a given time is determined by the action of the executing program; the memory locations that were most recently accessed are stored in the cache. Memory accesses to locations mapped into the on-chip memory do not generate external bus transactions and, therefore, are faster than accesses to external memory; thus, use of the on-chip memory leads to faster, more efficient program execution. On reset, the on-chip memory is automatically enabled for use as a cache for instructions only.

semiconductor memory accessed via bus transac­tions.) for each memory access, control logic in the MPU checks if the memory location involved is currently stored in the cache. If so, the access is made to the cache, usually without generating an external bus transaction; if not, the access is made to main memory and the contents of the cache may be updated.

Z280 MPU cache organization is illustrated in Figure 8-1. The cache is arranged as 16 lines of 16 bytes each. Each line of the cache can hold a copy of sixteen consecutive bytes of memory in physical memory locations whose 20 most signifi­cant address bits are identical. Thus, for exam­ple, one line of the cache could hold the data from physical memory locations 153820|\_| to 15382F The 20 bits of physical address asso­ciated with one line of 16 bytes in the cache is called the tag address for that line. Each line of the cache also has 16 valid bits associated with it; each byte in the line is associated with one valid bit. The valid bit is used to indicate if the corresponding byte in the cache holds a valid copy of the memory contents at the asso­ciated physical memory location.

* 1. CACHE MEMORY MODE



20 BITS 16 BITS 16x8 BITS

Tag n = the 20 Address bits associated with line n

Valid bits = 16 bits that indicate which bytes in the cache contain valid data Cache data = 16 bytes

Figure 8-1. Cache Organization

If the М/С bit in the Cache Control register is cleared to 0, then the 256 bytes of on-chip memory are treated as a cache. Cache memories are small, high-speed memory buffers situated between the processor and main memory. (Main memory is the

Lines in the cache are allocated using a Least-

Recently Used (LRU) algorithm. If a read access is made to a physical memory address not currently stored in the cache (a cache ’’miss”), and the MMU does not assert cache inhibit, the line in the

| **Operation** | **Hit/Miss** | **Cache Instruction** | **Cache Dat** | **Cache Activity** | | **Bus Transaction** | **Cache/Memory Supplies Information** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **a Contents** | **LRU** |
|  |  | **MMU Cache Inhibit** | | **-\* Cacheable Transaction** | |  |  |
| Instruction Read | Hit | Disabled | Don’t care | Updated | No change | Yes | Memory |
|  |  | Enabled | Don’t care | No change | Updated | None | Cache |
|  | Miss | Disabled | Don’t care | Updated\* | No change | Yes | Memory |
|  |  | Enabled | Don’t care | Updated | Updated | Yes | Memory |
| Data Read | Hit | Don’t care | Disabled | Updated | No change | Yes | Memory |
| (non Test & Set) |  | Don’t care  •' • • 1 ‘ | Enabled | No change . | .. Updated | None | ■■ Cache e |
|  | Miss | Don’t care | Disabled | Updated\* | No change | Yes | Memory |
|  |  | Don’t care | Enabled | Updated | Updated | Yes | Memory |
| Data Read | Don’t care | Don’t care | Don’t care | Updated\* | No change | Yes | Memory |
| (Test & Set) |  |  |  |  |  |  |  |
| Data Write | Hit | Don’t care | Disabled | Updated | No change | Yes |  |
|  |  | Don’t care | Enabled | Updated | Updated | Yes ' | -■ ■ ■ ■ |
|  | Miss | Don’t care | Disabled | No change | No change | Yes | ■ ■ |
|  |  | Don’t care | Enabled | No change | No change | Yes |  |
| EPU-to-Memory | Don’t care | Don’t care | Don’t care | Updated\* | No change | Yes | EPU |
| Memory-to-EPU | Don’t care | Don’t care | Don’t care | No change | No change | Yes | Memory |
| EPU Template | Don’t care | Don’t care | Don’t care | No change | No change | Yes | Memory |
| RETI Opcode | Don’t care | Don’t care | Don’t care | No change | No change | Yes | Memory |
|  |  | MMU Cache **Inhibit -♦** | | **Noncacheable Transaction** | |  |  |
| Don’t care | Don’t care | Don’t care | Don’t care | Updated\* | No change | Yes | Memory |
| ‘Updated if a cache line contains the accessed location, otherwise unaffected. | | | | | | | |

Table 8-1. CPU Accesses to On-Chip Memory as Cache

cache that has been least recently accessed is selected to hold the newly read data. All bytes in the selected line are marked invalid except for the bytes containing the newly accessed data. A cache miss on a data write does not cause a line to be allocated to the memory location accessed.

On a cache miss during a memory read, one or two bytes (depending on the bus size) are fetched from main memory and written to the cache. The cache does not prefetch beyond the currently requested byte or word, with one exception; if burst mode operations are specified in the Cache Control reg­ister, burst mode transactions are used when fetching instructions.

The cache can be configured to hold only instruc­tions, only data, or both instructions and data, as determined by the contents of the Cache Control register. If the cache contains data, writes to data at locations in the cache also generate external bus transactions to update the appro­priate memory locations; thus, external memory is always guaranteed to contain valid information.

Tables 3-1 and 8-2 summarize cache operation. Whether or not a given memory operation accesses the cache depends on a number of factors: the type of access being made (program read, data read, or data write), whether the cache is enabled for that type of access, the type of instruction being executed, whether the MMU asserts cache inhibit, and whether the CPU or a DMA device ini­tiates the transaction. The Cache Control regis­ter determines if the cache is used for instruc­tion fetches or data accesses or both. Execution of the Test and Set (TSET) instruction, Return from Interrupt (REII) instruction, and the extended instructions force external bus transac­tions, regardless of the contents of the cache, as described below. If the MMU is enabled, the access can be cacheable or noncacheable, as deter­mined by the contents of the page descriptor reg­ister in use. If the MMU is not enabled, all transactions are considered to be cacheable. Both the CPU and on-chip DMA channels can access the cache. For DMA operations, only data read and data write transactions can occur. The state of the Cache Data Disable control bit in the Cache

Control register is ignored during DMA transactions; therefore, an on-chip DMA device always updates the cache contents during DMA write operations to memory locations that are currently mapped into the cache.

For read operations, a cache ’’hit” is a reference to a location with a valid entry in the cache, and a cache ’’miss” is a reference to a location that has no valid entry in the cache. In the general case (and assuming the transaction is cacheable), read operations that are cache hits cause the data to be read from the cache without generating an external bus transaction. Read operations that are cache misses cause the data to be read from the external memory via an external bus cycle and update the cache contents. Updating the cache contents may involve replacing the least-recently used line of the cache with a new line that contains the read location. For write operations, a cache hit is a write to a location in the cache, even if the destination byte is marked as invalid in the cache, and a cache miss is a write to a location that is not in the cache. Write operations that are cache hits cause both the cache and external memory to be updated, and write operations that are cache misses have no effect on the cache. Memory write operations always gener­ate external bus transactions.

Exceptions to the above rules include the Test and Set, Return from Interrupt, and extended instruc­tions. Data read operations during execution of a Test and Set instruction always read the data from the main memory with an external bus transaction, regardless of whether or not the location read is valid in the cache. This ensures that the most recent value for a semaphore is read from external memory in the case that the semaphore is in shared memory in a multiprocessor system; another proces­sor may have changed the semaphore after it was last read into the MPU’s cache.

If an RE ГI opcode is fetched from the cache, the instruction fetch cycles are repeated with external bus transactions; this ensures that Z80 family peripherals connected to the Z280 MPU with an interrupt reguest daisy chain can detect the RE ГI opcode fetch (a reguirement for the proper operation of the Z80 family peripherals).

If extended instructions are resident in the cache, the EPU template portion of those instructions is always read using external bus transactions. This ensures that an Extended Processing Unit (EPU) that is monitoring the external bus can detect and read the template during those instruction fetch cycles. If the extended instruction results in a transfer of data between the EPU and memory, all the involved data transactions occur on the external bus. Cache hits during EPU-to-memory write transactions result in the updating of cache contents as well as external memory.

For memory reads, the LRU algorithm logic is updated to reflect that the associated cache line is the most-recently accessed line if the read was an instruction fetch in a cache enabled for instructions or a data fetch in a cache enabled for data. For data writes, the LRU algorithm logic is updated only for a cache hit in a cache that is enabled for data.

When the on-chip DMA controllers transfer data to memory, cache contents are modified if the write is to a location mapped into the cache, but the LRU algorithm is unaffected. EPU-to-memory transactions have the same effect. The cache is not affected by the activity of external DMA controllers.

On reset, all the valid bits in the cache are cleared to 0, marking all cache entries as invalid, and the on-chip memory is configured as a cache for instructions only.

| **Operation** | **Hit/Miss** | **Instruction** | **Cache Data** | **Contents** | **LRU** | **Transaction** | **Information** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Read | Hit | Don’t care | Don’t care | Updated | No change | Yes | Memory |
|  | Miss | Don’t care | Don’t care | Updated\* | No change | Yes | Memory |
| Write | Hit | Don’t care | Don't care | Updated | No change | Yes | — |
|  | Miss | Don’t care | Don’t care | No change | No change | Yes | — |

Table 8-2. On-Chip DMA Accesses (Both Flowthrough and Flyby) Effect on On-Chip Memory as Cache

Cache/Memory

Memory Cache Cache Activity Bus Supplies

\* Updated if a cache line contains the accessed location, otherwise unaffected.

* 1. FIXED-ADDRESS MODE

When the М/С bit in the Cache Control register is set to 1, the on-chip memory is treated as fixed physical memory locations. Accesses to these mem­ory locations never generate external bus transac­tions and, therefore, are faster than memory accesses that use the external bus (Table 3-3).

the on-chip memory. This is accomplished by ena­bling the on-chip memory as a cache for data only, reading data from 16 physical memory locations that are in different cache lines, and then set­ting the М/С bit in the Cache Control register to 1 to enable the fixed-address mode for the on-chip memory. Altering the М/С bit in the Cache Control register does not affect the contents of

the on-chip memory, including the tag addresses.

In this mode, the on-chip memory is still organ­

locations in each line.

in this mode.

and no dis-

the CPU is acces­

ized as 16 lines of 16 bytes each, with a 20-bit tag address that specifies the 16 physical memory

All locations are assumed

to contain valid information, whether or not they have been initialized; the individual valid bits associated with each byte in the line are ignored

The Cache Data Disable and Cache

Instruction Disable bits in the Cache Control reg­ister are also ignored in this mode, tinction is made as to whether sing instructions or data.

Before entering this mode, the user must initial­ize the tag addresses for all 16 lines of on-chip memory. The values for these tags determine the 256 physical memory addresses that are mapped into Note that each line of the on-chip memory must be assigned a unique tag address before entering this mode so that no unpredictable addresses are mapped into the on-chip memory. If instructions are to be fetched from the on-chip memory while in this mode. Return from Interrupt (RE II) instructions and the templates within extended instructions should never be resident in the on-chip memory; in each case, the operation of devices external to the MPU depends on these instructions being fetched with external bus transactions, as men­tioned in section 8.2. Data to be transferred to or from an EPU cannot be resident in on-chip mem­ory either, since this data must be transferred to the EPU over the external bus.

Table 8-3. DMA/CPU Accesses to On-Chip Memory as Fixed Memory Location

Cache/Memory

Memory Cache Cache Activity Bus Supplies

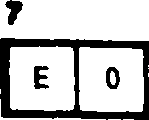
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Operation** | **Hit/Miss** | **Instruction** | **Cache Data** | **Contents** | **LRU** | **Transaction** | **information** |
| Read | Hit | Don’t care | Don’t care | No change | No change | No | Cache |
|  | Miss | Don’t care | Don’t care | No change | No change | Yes | Memory |
| Write | Hit | Don’t care | Don’t care | Updated | No change | No | ~~■ —~~ |
|  | Miss | Don’t care | Don’t care | No change | No change | Yes |  |

Глава 9.

Встроенные периферийные устройства

9.1 INTRODUCTION

9.3 REFRESH CONTROLLER



Этот регистр позволяет определять частоту появления полей в этом регистре

MPU Z280 оснащен рядом периферийных устройств на кристалле в дополнение к центральному процессору, MMU и кэш-памяти. Эти периферийные устройства включают генератор тактовых импульсов, контроллер динамического обновления оперативной памяти, четыре контроллера прямого доступа к памяти (DMA), три счетчика/таймера и универсальный асинхронный приемник/передатчик (UART).

Каналы DMA, счетчики/таймеры и UART являются программируемыми пользователем устройствами, которые могут быть сконфигурированы для работы в нескольких различных режимах. Доступ к этим устройствам осуществляется с помощью команд I/O; однако при обращении центрального процессора к встроенным периферийным устройствам не генерируется никаких внешних транзакций шины I/O. Эти устройства могут генерировать запросы на прерывание для MPU Z280, как описано ниже и в главе 6. Прерывания от этих встроенных периферийных устройств всегда обрабатываются с использованием режима прерывания 3, независимо от того, какой режим прерывания используется для генерируемых извне прерываний.

Встроенный контроллер обновления памяти в Z280 MPU доступен для генерации операций обновления памяти в системах, использующих динамические ОЗУ. Работой этого механизма управляет регистр частоты обновления, который расположен в адресном пространстве ввода-вывода Z280 MPU. Если включено, обновление памяти выполняется со скоростью, указанной содержимым этого регистра.

Формат 8-разрядного регистра частоты обновления показан на рисунке 9-1. Механизм обновления и транзакции обновления описаны ниже.1 » । • ■

**RATE I**

**1 1 I .1 I**

Figure 9-1. Refresh Rate Register

9.2 CLOCK OSCILLATOR

MPU Z280 оснащен встроенным тактовым генератором, который может быть подключен непосредственно к кристаллу или любому другому подходящему источнику тактовой частоты. Частота тактовой частоты процессора составляет половину частоты внешнего источника тактовой частоты или кристалла. Тактовая частота процессора может быть дополнительно разделена на коэффициент 1, 2 или 4, чтобы обеспечить синхронизацию шины, как указано в содержимом регистра синхронизации шины и инициализации (см. главу 3. Тактовый сигнал синхронизации шины выводится MPU для использования остальной частью системы.

Встроенный тактовый генератор, усилитель с высоким коэффициентом усиления, включается либо подключением кристалла к контактам Clock/Crystal Input (XTAL1) и Crystal Output (XTALO), либо подключением тактового входа к контакту Clock/Crystal Input. Кристалл должен быть параллельного резонансного основного типа.

**Refresh Enable (E) bit.** When this bit is set to 1, the refresh mechanism is enabled. When this bit is cleared to 0, the refresh mechanism is disabled and refresh transactions are not generated.

**Refresh Rate field.** The contents of this 6-bit field determine the frequency of refresh transactions if the Refresh Enable bit is set to 1. A value of n (0 < n < 63) in this field specifies a refresh rate of once every 4n processor clock cycles; a value of 0 in this field indicates a refresh rate of every 256 processor clock cycles.

The Refresh Rate register is accessed via byte I/O operations to I/O port address ЕЕххЕ8ц (where x means ’’don’t care”). Bit 6 of this register is not used. On reset, the Refresh Rate register is initialized to 88^, thereby enabling memory refresh at a rate of 32 processor clock cycles per refresh. This register can be read at any time to

rrfci. «ixif wuarvwwr.iuj

determine if refresh is enabled and the current refresh rate.

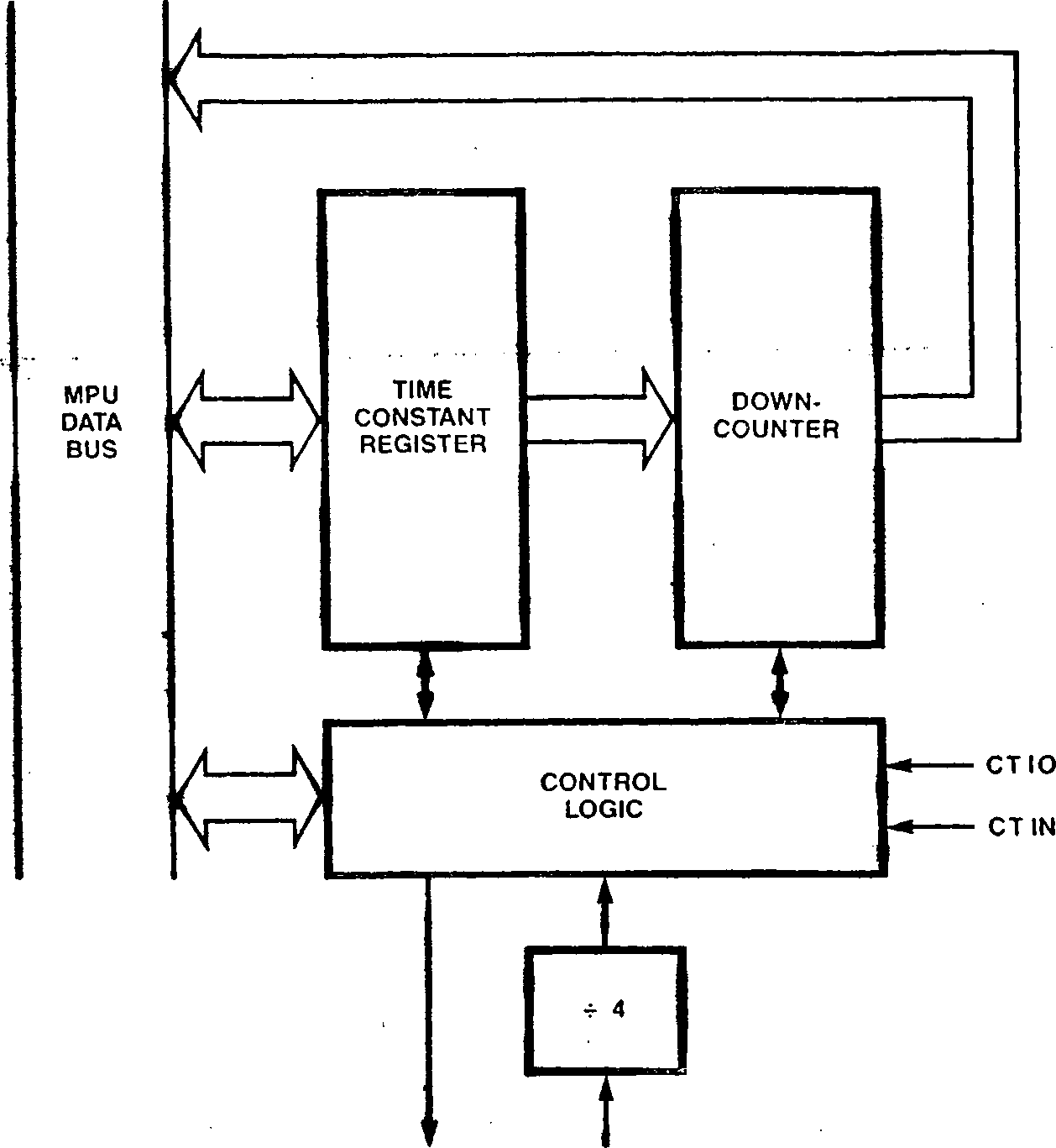
A 10—bit refresh address is output on address lines AQ-A9 during a refresh transaction. This refresh address is incremented by one for Z80 bus (8-bit data bus) configuration and by two for Z-BUS (16-bit data bus) configuration of the Z280 MPU between refresh transactions. The refresh address is not accessible by the programmer and is not affected by a reset.

During instruction execution, the actual refresh transactions are generated as soon as possible after the refresh period has elapsed. Generally, the refresh transaction is executed after the last clock cycle of the bus transaction in progress at the time that the refresh period elapsed. If the CPU receives an interrupt request during that same bus transaction, the refresh transaction is inserted before processing the interrupt. When the Z280 MPU does not have control of the bus due to a bus request, refresh transactions cannot be executed; while the MPU is in this state, internal circuitry records the number of refresh periods that have elapsed (that is, the number of ’’missed” refresh transactions). When the Z280 MPU regains control of the bus, the refresh mechanism automatically issues the missed refresh cycles. Similarly, if the refresh period elapses while the MPU is in a wait state (due to WAIT being asserted) during a bus transaction, the number of missed refresh transactions is recorded internally, and those refresh cycles are issued after WAIT is deactivated and the bus transaction is completed. The internal circuitry can record up to 256 such missed refresh operations.

Pseudo-static memories and some peripheral devices (such as the Z8000 family of peripherals) require a minimum transaction rate on the bus for correct operation. If the refresh mechanism is disabled by clearing the Refresh Enable bit in the Refresh Rate register, the rate field in this register is used to determine the minimum transaction rate on the bus. In this mode, if the refresh timer reaches 0 and no external bus transaction has occurred since the last time the refresh timer elapsed, then a refresh transaction will be generated. Thus, in a system that does not require memory refresh transactions, the Refresh Rate field in the Refresh Rate register must be initialized to an appropriate value even if memory refresh operations are disabled.

9e4 COUNTER/TIMERS

Ihe Z280 MPU’s three on-chip 16-bit counter/timers can be configured to satisfy a broad range of counting and timing applications, including event counting, interval timing, watchdog timing, and clock generation. Each counter/timer is composed of a 16-bit downcounter, a 16-bit time constant register, and two 8-bit control and status registers (the Counter/Timer Configuration register and the Counter/Timer Command/Status register). The three independent devices are referred to as counter/timer 0 (C/T 0), counter/ timer 1 (C/T 1), and counter/timer 2 (C/T 2). Figure 9-2 is a block diagram of a Z280 MPU counter/timer.



INTERRUPT CPU

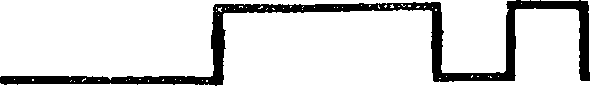
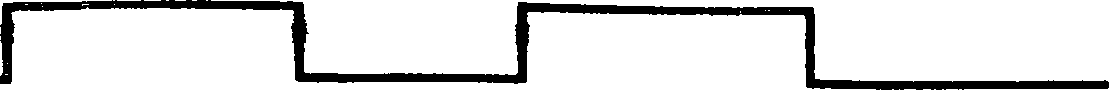
TO CPU CLOCK

Figure 9-2. Counter/Timer Block Diagram

C/T 0 and C/T 1 can be programmably linked to form a 32-bit counter/timer.

Two external connections are available for each counter/timer: a Counter/Timer I/O pin (C/T I/O) that can act as a gale or trigger input or a counter/timer output, and a Counter/Timer Input pin (C/T IN) that can serve as a count, gate, trigger, or gate/trigger input. The contents of the Counter/Timer Configuration register determine the pin functions for a given application.

The counter/timers can operate in counter mode or in timer mode. In counter mode, the downcounter decrements the count on the occurrence of an external event; specifically, the counter is clocked by a rising edge on the Counter/Timer Input pin. In timer mode, the downcounter is clocked by an internal signal—the CPU clock divided by four.



Gate and trigger inputs to the downcounter can be used to control counter/timer activity. Both hardware and software gate and trigger signals are available. Either retriggerable or nonretrigger- able modes can be specified.

In either mode, the maximum count frequency is the CPU

clock divided by four.

9e4.2 Gates and Triggers

low-to-hiqh transitions.

If the appropriate

either counter mode or

GATE

INPUT

COUNTER OR TIMER

Figure 9-3. Counter Operation with Gate Only

COUNT/TIME

REGISTER DECREMENTED

The counter/timer\*s "terminal count" condition is

when the downcounter holds a count of 0. This terminal count condition can be used to generate an interrupt request to the CPU. Counter/timecs can generate a counter/timer output signal when the terminal count is reached. Upon reaching terminal count, a counter/timer can be programmed either to discontinue counting (single-cycle mode) or to reload the initial time constant value and continue counting (continuous mode). -4\*

9.4.1 Countег/Timer Operating Modes

The counter/timers have two basic operating modes, distinquished by the clocking signal to the downcounter: counter mode and timer mode. The current mode for counter/timer operation is determined by the contents of the Counter/Timer Configuration register.

In counter mode operation, the counter/timer monitors an external input line and records low-to-high transitions on that line. The Counter/Timer Input pin is used as the counter’s input signal; if the appropriate enabling conditions are met, a low-to-hiqh transition on that pin will cause the contents of the down­counter to be decremented by one. The decrement operation in the downcounter is actually performed on the first rising edge of the scaled processor clock (CPU clock divided by 4) after the low-to-high transition on the C/T IN signal.

Typically, counter mode is used in event-counting types of applications.

In timer mode operation, the counter/timer monitors the internal CPU clock scaled by four for enabling conditions are met, such a transition causes the contents of the downcounter to be decremented by one. No external inputs are required in the timer mode of operation. Timer mode is used in applications such as delay interval timing, watchdoq timing, and clock generation.

Gate and trigger inputs are used to control counter/timer activity in timer node.

Gate signals are used in applications where counting or timing is to occur only during certain specified intervals; the counter/timer will count or time only while the gating condition is met. for applications where an external pin is configured as a gate input, counting or timing operations are performed only while the gate input is high. A software gate bit (one bit of the Counter/Timer Command/Status register) is used as a filter for the gate input; while the software gate bit is cleared to 0, the gating condition is nut met regardless of the state of the gating line. In other words, the gating condition is a logical AND of the hardware and software gates; both the gate input must be high and the software gate bit must be set to 1 fur the counter/ timer to be operating. If no external pins are configured as a gating signal, then the software gate bit must be set to 1 to satisfy the gating condition.

Figure 9-3 illustrates the gating facility in an application where the counter/timer is in counter mode with both the gate and the count signals coming from external pins. This example assumes that the software gate bit has been set to 1. The contents of the downcounter are decremented on a low-to-high transition of the count input only if the gate input is high.

If trigger mode is selected, a countdown seguence for a counter/timer begins only after a triggering condition occurs; a counting or timing operation can begin only after a low-to-hiqh transition is detected on the trigger. If an external input is used as a trigger, that line is monitored by the counter/timer. Alternatively, a software trigger bit (one bit in the Counter/Timer Command/Status register) can be set to 1 from a previously cleared value to activate the counter/timer. The

triqqer condition is a logical OR of the hardware and software triggers; that is, either a hardware or software trigger will activate an enabled counter/timer.

figure 9-4 illustrates trigger operation in an application where the counter/timer is in the counter mode with both the triqqer and count inputs provided by external pins. This example assumes that the software trigger bit does not make a low to high transition. The contents of the downcounter are decremented on a low-to-high transition of the count input only after a low-to-hiqh transition on the trigger input has been detected.

Either a retriggerable or nonretriggerable operation can be specified. In the retriggerable mode, the occurrence of a trigger condition causes the counter/timer to reload its initial time constant value regardless of the current contents of the downcounter. This mode is used in applications such as watchdog timers. In the nonretriggerable mode, after the first trigger condition starts counter/timer activity, subsequent trigger conditions are ignored. Nonretriggerable mode is used in applications such as delay counters that measure a fixed delay from a given event.

Gate and trigger operations can be combined in a single counter/timer. Separate gate and trigger inputs (either hardware or software) can be specified, or one external input can be used as both a gate and a trigger. In the latter case, a low-to-high transition on the input acts as a triqqer that starts counter/timer activity, and then counting or timing continues only as long as the input signal remains high. Again, either retriggerable or nonretriggerable modes are available. figure 9-5 illustrates counter/timer

operation in an application where counter mode is selected, one input is a count input, and the other input is used as both the trigger and gate.

* + 1. Terminal Count Condition

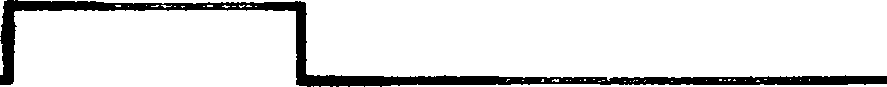
During operation, the counter/timer counts down from a preset time constant value. The time constant value can range from 0 to 65535. The terminal count condition is reached with the transition from a count of 1 in the duwncounter to **a** count of 0. The counter/timers can be programmed to interrupt the CPU and/or generate a counter/timer output signal when the terminal count is reached.

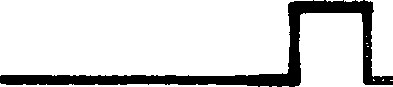
Another set of operating modes determines counter/timer activity upon reaching the terminal count. Whether in counter or timer mode, a counter/timer can be configured for single-cycle mode or continuous mode. In single-cycle mode, the counter/timer halts operation upon reaching terminal count; a new trigger is reguired to reload the time constant and initiate another countdown sequence. In continuous mode, the counter/timer is automatically reloaded with the time constant upon reaching terminal count; the downcounter is reloaded on the next count input after reaching terminal count. for example, a counter/timer in continuous mode with a 3 in its Time Constant register will be reloaded on every fourth count input.

An interrupt enable bit in the Counter/Timer Configuration register determines if an interrupt request is generated at the terminal count. This request will be processed by the CPU if the appro­priate Interrupt Request Enable bit in the CPU’s Master Status register is set to 1 (see Chapter 6).

**TRIGGER**

INPUT





*»J ‘*

COUNTER

Figure 9-4. Counter Operation with Trigger Only

COUNT/TIME

REGISTER

**DECREMENTED**

СОУ^Т/TIME

REGISTER

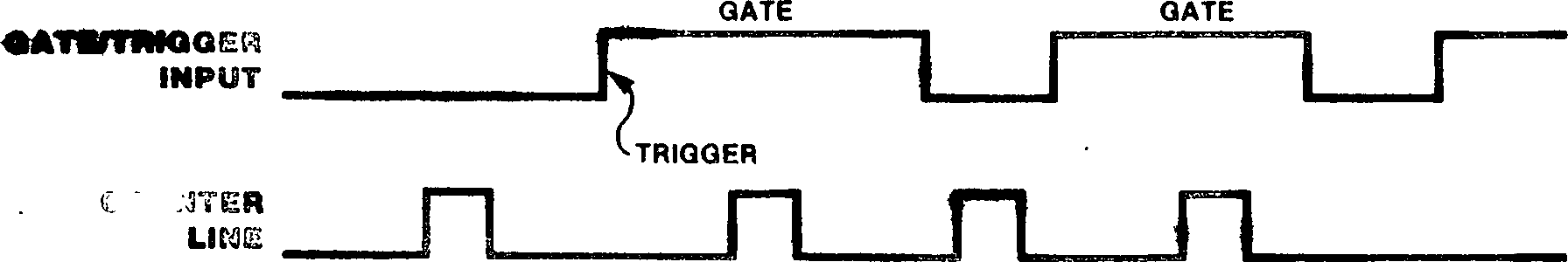


Figure 9-5. Counter Operation with Gate and Trigger

Each counter/timer

C/S RE

IE CTC

IPA

| **EO** | **IPA Field** | | **Pin Functionality** | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **C/T** | **G** | **T** | **Counter/Timer I/O** | **Counter/Timer Input** | **Mode** |
| 0 | 0 | 0 | 0 | Unused | Unused | Timer |
| 0 | 0 | 0 | **1** | Unused | Trigger | Timer |
| 0 | 0 | 1 | **0** | Gate | Unused | Timer |
| 0 | 0 | 1 | 1 | Gate | Trigger | Timer |
| 0 | 1 | 0 | **0** | Unused | Input | Counter |
| 0 | **1** | 0 | 1 | Trigger | Input | Counter |
| 0 | 1 | 1 | 0 | Gate | Input | Counter |
| 0 | 1 | 1 | 1 | Gate/Trigger | Input | Counter |
| 1 | 0 | 0 | 0 | Output | Unused | Timer |
| 1 | 0 | 0 | **1** | z Output | Trigger | Timer |
| 1 | 0 | 1 | 0 | Output | Gate | Timer |
| 1 | 0 | 1 | **1** | Output | Gate/Trigger | Timer |
| 1 | **1** | 0 | 0 | Output | Input | Counter |
| 1 | 1 | 0 | **1** | Unused | Unused | Reserved |
| 1 | 1 | 1 | 0 | Unused | Unused | Reserved |
| 1 | 1 | 1 | **1** | Unused | Unused | Reserved |

Table 9-1. Encoding of the IPA Field in the Counter/Timer Configuration Register

The CTIO pin can be configured as a counter/timer output signal. Reaching the terminal count condition causes a low-to-high transition on the Cl 19 pin; this signal remains high as long as the downcounter holds a value of zero (that is, until **a** non-zero time constant is loaded into the downcounter due to a trigger condition).

* + 1. Counter/Tinter Registers

has two 8-bit command and status registers and two 16-bit count registers. The 8-bit Counter/Timer Configuration and Counter/Timer Command/Status registers determine the counter/timer ’ s operating modes and provide status information about the current operation.

**if** C/T 0 and C/T I are linked to form a 32-bit counter/timer, the functionality of these registers is affected, as described in section 9.4.5. The 16-bit Time Constant register holds the initialization value for the counter/timer, and the 16-bit Count-Time register contains the value of the current count in progress.

* + - 1. Counter/Timer Configuration Register

The Counter/limer Configuration register, shown in Figure 9-6, specifies the counter/timer’s mode of operation. Ihe five fields in this register are described below.

• CTC Is present on counter/timer 0 only.

Figure 9-6. Counter/Timer Configuration Register

**Continuous/Single Cycle (C/S).** While this bit is set to 1, the duwncounter is automatically reloaded with the contents of the Time Constant register on the next c?ount input signal after terminal count is reached, and the counting or timing operation continues. While this bit is cleared to 0, no automatic reloading occurs when terminal count is reached.

**Retrigger Enable (RE).** While this bit is set to 1, the value of the Time Constant register is loaded into the downcounter whenever a trigger input is received (retriggerable mode). While this bit is 0, trigger conditions do not cause reloading of the downcounter.

**Interrupt Enable (IE).** While this bit is set to 1, the counter/timer generates an interrupt reguest to the Z280 CPU upon reaching terminal count. While this bit is cleared to 0, no interrupt reguests can be generated by the counter/timer.,,,

**Counter/Timer Cascade (CTC).** For C/T 0, this is the enable bit for linking to C/T I in order to form a 32-bit counter/timer (see section 9.4.5). The state of this bit has no effect in C/T I and C/T 2.

**Input Pin Assignments (IPA).** The contents of this 4-bit field determine the operating mode of the counter/timer (counter or timer mode) and the functionality of the external pins associated with that counter/timer. The four bits in this field are associated with enabling the generation of an output pulse (E0), selecting the counter or timer mode (C/T), enabling the gating facility (G), and enabling the triggering facility (I). Table 9-1 shows the encoding of this field.

IГ a reserved encoding of the specified fur any counter/timer, operation is unpredictable.

The Counter/Timer Configuration cleared to all zeros by a reset.

9.4.4. 2 Counter/Timer Command/Status Register

IPA field is  
counter/timer

registers are

The Counter/Timer Command/Status register provides for software control of counter/timer operation and reflects the current status of the counter/ timer. Three control bits and three status bits are included in the Command/Status register. The format for this register is illustrated in Figure 9-7. **7** 0

I EN GT TG 1 1 CIP CC CORI

Figure 9-7. Counter/Timer Command/Status Register

**Enable (EN).** While this bit is set to 1, the counter/timer is enabled; operation begins on the first rising edge of the processor clock following the setting of this bit from a previously cleared state. Writing a 1 to this bit when its previous value was a 1 has no effect. While this bit is cleared to 0, the counter/timer is disabled and performs no counting or timing operations. While in the disabled state, the contents of the Time Constant register are continuously loaded into the downcounter.

**Software Gate (GT).** While the counter/timer is enabled (the EN bit is a 1), downcounter operation begins on the rising edge of the first scaled processor clock following the setting of this bit from a previously cleared value. Writing **a** 1 to this bit when the previous value was a 1 has no effect. While this bit is cleared to 0, the counting or timing seguence is halted.

**Software Trigger (TR).** While the counter/timer is enabled (the EN bit is a 1), the trigger condition is generated on the rising edge of the first scaled processor clock following the setting of this bit from a previously cleared value. If a previous trigger condition has not occurred, the contents of the Time Constant register are loaded into the downcounter and the counting or timing seguence begins. If a hardware or software trigger has already occurred and the Retrigger Enable bit is set to 1, the counter/timer will be retriggered. If a trigger has already occurred, the Retrigger Enable bit is cleared to 0, and a counting or timing operation is in progress (that is, the downcounter holds a count other than 0), then setting the TR bit has no effect on counter/timer operation. Clearing this bit to 0 also has no effect on counter/timer operation.

Count in Progress (CIP). This status bit indicates if a counting or timing operation is in progress. While this bit is a 1, the counter/timer has a time constant loaded and the downcounter holds a non-zero value. While this bit is a 0, the counter/timer is not operating. The state of this bit is determined by control logic in the counter/timer and cannot be altered by a write operation to this register.

**End-of-Count Condition Has Been Reached (CC).** This status bit is set to 1 by control logic in the counter/timer when the end-of-count condition is reached (that is, the downcounter has been decremented to zero in the single-cycle mode or the downcounter has been reloaded in the

continuous mode). While this bit is a 0, the downcounter has not been decremented to 0 since the last time that this bit was cleared by software. This bit can be read or written under

program control. \* > • • • •

**Count Overrun (COR).** This status bit is set to 1 by control logic in the counter/timer if the end-of-count condition is reached while the CC bit is already set to 1, thereby indicating a count over-run condition. If this bit is a 0, the end-of-count condition has not been reached while the CC bit is a 1 since the last time the CC bit was cleared by software. This bit can be read or written under program control.

The Counter/Timer Command/Status register is cleared to all zeros by a reset. Bits 3 and 4 of this register are not used, and should always be written with zeros (however, when bits 3 and 4 are read back, they will be 1s regardless of whether they were written with zeros or ones).

9.4.4.3 Time Constant and Count-Time Registers

The 16-bit Time Constant register holds the value to be loaded into the downcounter when counter/ timer operation begins. The downcounter is loaded with the contents of the Time Constant register when the counter/timer is initially triggered to begin counter/timer operation, each time the end-of-count condition is reached if the continuous mode is selected, and at the occurrence of each trigger condition if retriggerable mode is selected. By loading the Time Constant register, the user can specify counts ranging from 1 to 65536. The contents of the Time Constant register are continuously loaded into the downcounter while the counter/timer is disabled (the EN bit is 0).

The 16-bit Count-Time register holds the current value in the downcounter and can be read at any time without affecting counter/timer operation. Writes to this register have no effect.

Both the Time Constant and Count-Time registers hold unpredictable values after a reset.

fable 9-2 lists the 1/0 port addresses associated with each of the counter/timers\* registers• The Counter/Timer Configuration register and Counter/ Timer Command/Status register are accessed with byte I/O instructions and, with the exception of the read-only CIP bit, can be read or written. The Time Constant and Count-Time registers are accessed with word 1/0 instructions. The Time

If the IE bit in the more

the counter/timers.

significant counter/timer (С/Г 1) is set to 1, an interrupt reguest is generated when the 32-bit counter reaches end-of-count, using the interrupt request signal from C/T 1; if the IF. bit in the less significant counter/timer (C/F 0) is set to 1, an interrupt request is generated when the lower 16 bits of the 32-bit downcounter reach 0

Constant register can be read or written; the

(in other words, when C/T 0 reaches end-of-count),

Count-Time register is read-only.

using the interrupt request signal from C/T 0. If

Table 9-2, I/O Addresses of Counter/Timer Registers

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **C/T0** | **Counter/Timer C/T1** | **C/T2** |
| Configuration | FExxEO | FExxE8 | FExxF8 |
| Command/Status | FExxEI | FExxE9 | FExxF9 |
| Time Constant | FExxE2 | FExxEA | FExxFA |
| Count-Time | FExxE3 | FExxEB | FExxFB |
| All addresses are in hexadecimal. | |  |  |
| “x” means "don’t care" | • |  |  |

9.4.5 Linking Counter/Tiraers

the 0E bit in C/T 0 is set, the C/T 1/0 signal associated with C/T 0 goes high whenever the lower half of the 32-bit down-counter holds a 0 (in other words, when C/F O’s downcounter holds a 0).

Similarly, the Command/Status register in the more significant counter/timer (C/T I) contains the control and status bits for the linked 32-bit counter/timer. However, the status bits in the less significant counter/timer (C/T 0) hold valid status for the lower-half of the 32-bit counter/timer (that is, the status of C/T 0 itself).

9.4.6 Counter/Tieer Sequence of Events

Under software control, two Z280 MPU counter/ timers can be linked to form a 32-bit counter/ timer. C/T 0 can be linked with C/T 1. This linking function is controlled by the CTC bit in the Counter/Timer Configuration register in C/T 0. While the СГС bit in C/T O’s Configuration register is set to 1, C/T 0 and C/T 1 are linked together.

Before starting a counting or timing sequence, the counter/timer must be configured for the par- \* ticular application by loading its Configuration register. Next, the starting value for the downcounter is specified by loading the Time Constant register; initial values ranging from 0 to 65535 can be specified for the downcounter. Lastly, the enable (EN) bit in the Command/Status register is set to 1 to enable counter/timer

the C/S, RE, register of (C/T I). Any IPA field of

Linking the two counter/timers together affects the functionality of the counter/timers\* registers. If C/T 0 and C/T I are linked to form a 32-bit counter, C/T 1’s Time Constant register holds the upper 16 bits and C/T O’s Time Constant register holds the lower 16 bits of the 32-bit count to be loaded into the downcounter when a counter/timer operation begins. Similarly, C/T 1’s Count-Time register holds the upper 16 bits and C/T O’s Count-Time register holds the lower 16 bits of the current count.

The effect of linking counter/timers on the Con­figuration and Command/Status registers is summarized in Table 9-3. The configuration of the 32-bit counter/timer is determined by the state of and IPA fields in the Configuration the more significant counter/timer external connections specified in the the C/T 1 Configuration register use the pins associated with C/T I. The controls in the Configuration register for C/T 0 are ignored, with the exception of the CTC, IE, and E0 bits. The CTC bit in C/T 0 is used to specify linking ofoperation.

While the EN bit is cleared to 0, the counter/ timer cannot be triggered, interrupt requests from the counter/timer cannot be generated, and the downcounter holds the value in the Time Constant register. However, clearing the EN bit does not clear any pending interrupt requests—it only prevents new interrupt requests from being generated.

Once the EN bit is set to 1, the countdown sequence begins when the counter/timer is triggered, causing the contents of the Time Constant register to be loaded into the down counter. The downcounter is loaded on the rising edge of the external trigger input (if an external trigger was specified in the Configuration register) or by writing a 1 into the TG bit of the Command/Status register. The EN and TG bits can both be set to 1 during the same write operation to the Command/Status register to both enable and trigger a counter/timer (assuming that the TG bit was a zero previously, so that a low-to-high

Table 9-3. Configuration and Command/Status Registers for Linked Counter/Timers

|  |  |  |
| --- | --- | --- |
| **Bit** | **Active/lgnored** | **Comments** |
| **C/T 1 Configuration Register** | |  |
| C/S | Active | Specifies continuous or single-cycle mode for 32-bit counter/timer. |
| RE | Active | Specifies retriggerable or nonretriggerable mode for 32-bit counter/timer. |
| IE | Active | Interrupt enable for 32-bit counter/timer. |
| CTC | Ignored |  |
| EO ■ | Active | Enable output for 32-bit counter/timer; C/T 1 's output pin is used. |
| C/T | Active | Specifies counter or timer mode for 32-bit counter/timer. |
| G | Active | Enable gate input for 32-bit counter/timer; C/T 1 ’s input pin is used. |
| **у** | Active | Enable trigger input for 32-bit counter/timer; C/T 1 ’s input pin is used. |

|  |  |
| --- | --- |
| **C/T 0 Configuration Register** | |
| C/S | Ignored |
| RE | Ignored |
| IE | Active |
| CTC | Active |
| EO | Active |
| C/T | Ignored |
| **G** | Ignored |
|  | Ignored |

Interrupt enable for lower half of 32-bit counter/timer.

Set to 1 to link counter/timers. ......

Enable output for lower half of 32-bit counter/timer (C/T 0 only).

EN

GT

TG CIP

EN Ignored

GT Ignored

TG Ignored

CIP^ , Active

CC Active

COR Active

C/T1 Command/Status Register

Active Enable control for 32-bit counter/timer.

Active Software gate for 32-bit counter/timer.

Active Software trigger for 32-bit counter/timer.

Active Count-in-Progress status bit for 32-bit counter/timer.

Active End-of-Count Has Been Reached status bit for 32-bit counter/timer.

COR Active Count Overrun status bit for 32-bit counter/timer.

C/T 0 Command/Status Register

Count-in-Progress status bit for lower half of 32-bit counter/timer.

End-of-Count Has Been Reached status bit for lower half of 32-bit counter/timer.

Count Overrun status bit for lower half of 32-bit counter/timer.

transition on the trigger is detected). The trigger condition is a logical DR of the external trigger input (if specified) and the TG bit. (

I

Once triqqered, the rate at which the downcounter counts is determined by the mode of the counter/ timer. In the timer mode, the downcounter is clocked internally by a signal that is one-fourth the frequency of the CPU clock (one-eighth the frequency of the external clock source). In the counter mode, the downcountec is clocked by a rising edge on the count input signal (this edge is internally synchronized with the scaled CPU clock).

In counter mode, the first low-to-high transition on the count input should occur a minimum of four internal CPU clock cycles after the trigger event. Count inputs occurring within four CPU clock cycles of the trigger may or may not be recognized by the downcounter.

Once the downcounter is loaded, the countdown sequence continues towards the terminal count condition as long as the counter/timer\* s gate input is high. The gate input to the counter/ timer is the logical AND of the external gate input (if an external gate was specified in the Configuration register) and the GI bit in the

Cumin and/St atus register. If the gate input goes low, the countdown halts, and then resumes when the gate input goes high again. The gate function does not affect the trigger function.

The reaction to triggers during the countdown operation depends on the state of the RE bit in the Configuration register. If RE is a 0? retriggers are ignored and the countdown sequence continues normally. 1f RE is a 1, each occurrence of a trigger condition causes the downcounter to be reloaded from the Time Constant register and All four DMA channels, referred to as DMAO, DMA1, DMA2, and DMA3, are capable of controlling ’’flowthrough” type data transfers, wherein data Is temporarily stored in the DMA device between reading from the source and writing to the destination. Two of the channels, DMAO and DMA1, also support ’’flyby” mode data transfers, wherein the data is read from the source and written to the destination during a single bus transaction. Otherwise, the four DMA controllers are identical, although they have different priorities with respect to interrupt and bus requests.

the countdown sequence starts over again.

The current state of the downcounter can be determined by polling the status bits in the Command/Status register and by reading the current count from the Count-Time register. Reading these registers does not affect the current countdown sequence.

The state of the C/!T bit in the Configuration - register controls the operation of the counter/ timer upon reaching terminal count. If the C/S bit is a 1, specifying the continuous mode of operation, the downcounter is reloaded from the Time Constant register on the next count input after reaching terminal count, and a new countdown sequence begins. The Time Constant register can be programmably altered during counter/timer operation without affecting the current countdown sequence. If the С/S' bit is 0, specifying single-cycle operation, the downcounter halts upon reaching terminal count until the next occurrence of a trigger condition reloads the downcounter.

If the IE bit in the Configuration register is a 1, an interrupt request is generated upon reaching the terminal count. If a counter/timer output signal is specified in the IPA field of the Configuration register, reaching terminal count causes a low-to-high transition on the output signal; this signal then remains high until the downcounter is reloaded with a non-zero value due to a trigger condition or disabling of the counter/timer with a non-zero value in the Time Constant register. Note that the counter/timer output line can be forced high by disabling the counter/timer with all zeros loaded into the Time Constant register.

9Л DMA CHANNELS

The Z2R0 MPU has four on-chip Direct Memory Access (DMA) transfer controllers for high-bandwidth data transmissions within a Z280-based system. Each DMA channel is capable of controlling high speed memory-to-memory, memory-to-peripheral, peripheral-to-memory, or peripheral-tо-peripheral data transfers.

Two external signals provide the interface between the DMA channels and external memory or peripheral devices. The READY (RDY) input is used by an external device to request activity by a DMA channel. The DMA SIROBE (DMASTB) output is used to signal the 1/0 port when a flyby transaction is in progress; DMASIB is available only for DMAO and DMA 1.

• \ I\*. •\* • \*. » • \* \*. *I 4 \** . »• .» Г» ' \* 5 M *«.♦ t Л* \* t • •

Two 24-bit addresses are generated by the DMA for each flowthrough transaction, and one 24-bit address for each flyby transaction. These addresses can be physical memory addresses or I/O port addresses. The addresses are automatically generated for each transaction, and can be fixed, incrementing, or decrementing. Two readable registers, the Source Address register and Destination Address register, hold the current address of the source and destination ports.

During a DMA-controlled transaction, the DMA channel assumes control of the system’s address and data buses. The on-chip DMA channels behave as if they were external bus requestors with respect to acquiring, using, and relinquishing the bus. The DMA channels are arranged in a priority daisy chain with the external Bus Request input signal being the ’’next lowest bus requestor” on the chain. Data can be transferred as bytes or words, using the same memory and I/O timing as the CPU for bus transactions (as determined by the contents of the Bus Timing and Initialization register).

Two DMA devices can be programmably linked, where one DMA channel is used to program the second DMA channel. DMA3 can be linked to DMA1 and DMA2 can be linked to DMAO in this manner. DMAO can also be programmably linked to the on-chip UART’s receiver, and DMA1 can be linked to the on-chip UART’s transmitter.

The DMA Master Control register specifies the

general configuration of all four DMA channels, including the linking of DMA channels to the UART. Each DMA channel has its own Transaction

Descriptor register that determines the operating

modes fur that channel, Source Address and

9.5.2 DMA Transfer Modes

Destination Address registers that hold the addresses for the DMA transfers, and a Count register that controls the number of transfers to be performed. All DMA registers are accessed via 1/0 instructions.

9.5.1 Types of DMA Operations

The Z280 MPU’s on-chip DMA channels are capable of two basic types of operations: flowthrough mode data transactions and flyby mode data transactions.

All four on-chip DMA channels support flowthrough mode data transactions. In flowthrough mode, each DMA-controlled data transfer involves two bus operations: a read cycle to obtain the data from the source and a write cycle to transfer the data to the destination. The data is temporarily stored in the DMA device between the read and write operations. Flowthrough mode transactions use the same address, data, and control signals as CPU-initiated transactions and, therefore, require no additional external logic in a Z280-based system. Memory-to-memory, memory-to-peripheral, peripheral-to-memory, or peripheral-to-peripheral transfers are possible using flowthrough mode.

When transferring data under DMA control (with either flowthrough or flyby transact ions), one of three transfer modes can be selected: single transaction, burst, or continuous mode. Once DMA activity has been initiated, the transfer mode determines how many DMA-controlled data transfers are to occur before the DMA channel relinquishes the bus to the CPU or another DMA channel.

In the single transaction mode, the DMA controller transfers only one byte or word of data at a time. Control of the system bus is returned to the CPU between each DMA transfer; the DMA must make a new request for the bus before performing the next data transfer.

In the burst mode, once the DMA channel gains control of the bus, it continues to transfer data until the RDY input goes inactive. When the RDY line becomes inactive, the DMA releases the system bus; bus control then returns back to the CPU or to the next lower-priority DMA channel with a bus request pending.

In the continuous mode, the DMA channel retains control of the system bus until the entire block uf data has been transferred. If the RDY line

Flyby mode data transactions ace supported only by DMAO and DMA1. In a flyby mode transaction, the data is read from the source and written to the destination in a single bus operation. There are two types of flyby transactions: memory-to-

peripheral and peripheral-to-memory. For a memury-to-peripheral transaction, the DMA channel generates a memory read bus cycle and notifies the , , I/O device that a flyby transaction is in progress by activating the DMASTB output. The data must be written to the I/O device during the memory read operation. For a peripheral-to-memory flyby transaction, the DMA channel generates a memory write bus cycle while activating the DMASTB output; the data must be read from the I/O device during the memory write transaction. In other words, during flyby mode transactions, the DMA channel generates the bus signals needed to control the memory access, and DMASTB is used to notify the peripheral device when to read data from the bus (for memury-to-peripheral transfers) or when to put data onto the bus (for

peripheral-to-memory transfers.) Thus, flyby mode transactions require additional external logic to activate the appropriate peripheral device when DMASTB is active. However, flyby mode

transactions are faster than flowthrough mode transactions, since only one bus cycle is needed to complete a data transfer.

goes inactive before the entire data block is transferred, the DMA simply waits until RDY becomes active again, without releasing the bus. This mode is the fastest mode since it has the least response-time overhead when the RDY line momentarily goes inactive and returns active again. However, this mode does not allow any CPU activity for the duration of the transfer. Figure 9-8 summarizes the DMA transfer modes.

In any transfer mode, once a DMA-controlled data transfer begins, that transaction is completed in an orderly fashion, regardless of the state of the RDY input.

DMAO and DMA1 include a software RDY signal in the DMA Master Control register. The RDY input to these DMA channels is the logical OR of the RDY pin and the software-controlled RDY signal.

A DMA channel can be programmed to perform data transfers on a byte (8-bit), word (16-bit), or long word (32-bit) basis. If a DMA’s port address is a memory address that is auto-incremented or auto-decremented after each transfer, the size of the data transfer determines whether the memory address is incremented or decremented by a factor of 1, 2, or 4. For word and long word transfers to or from memory locations, the memory address must be even-valued (that is, the least significant bit of the memory address must be 0).

RELEASE BUS  
(CPU EXECUTES  
AT LEAST ONE  
MACHINE CYCLE)

SET STATUS FLAG

I

INTERRUPT RELEASE BUS

Figure 9-8b. Burst Mode

TRANSFER ONE BYTE OR WORD

NO

I

? v INTERRUPT RELEASE BUS

INTERRUPT  
RELEASE BUS

Figure 9-8a. Single Transaction Mode

Figure 9-8c. Continuous Mode

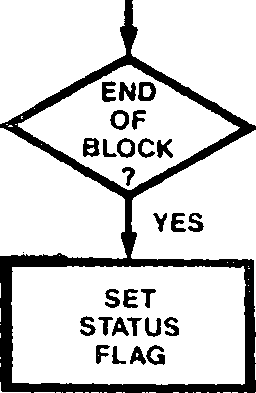
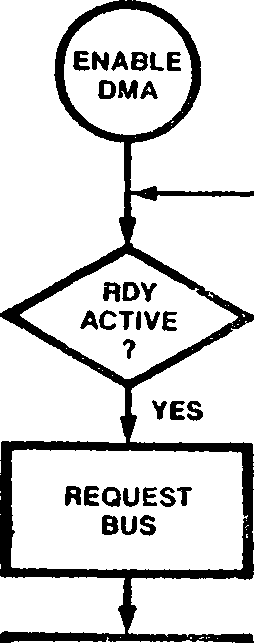
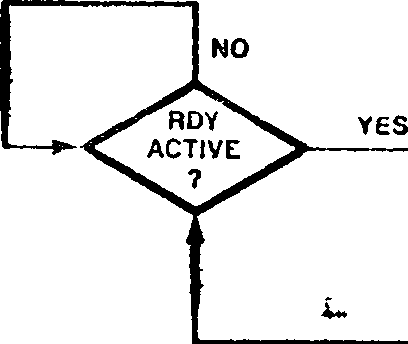
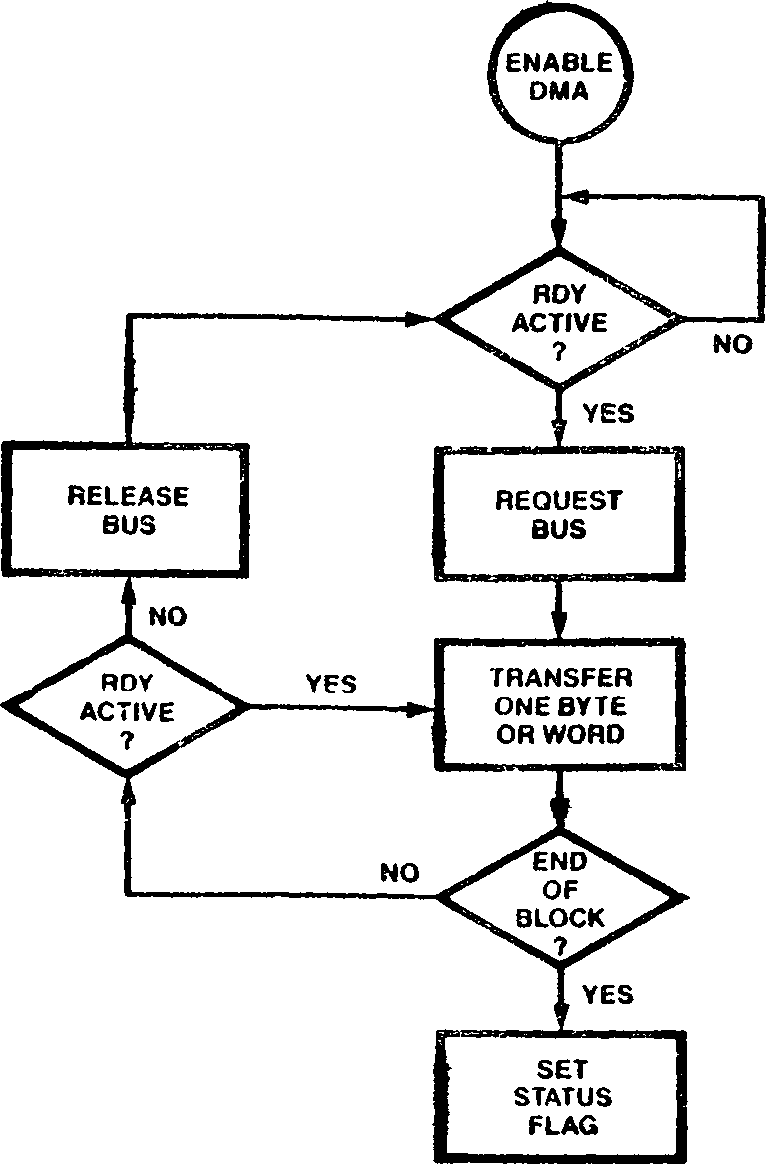
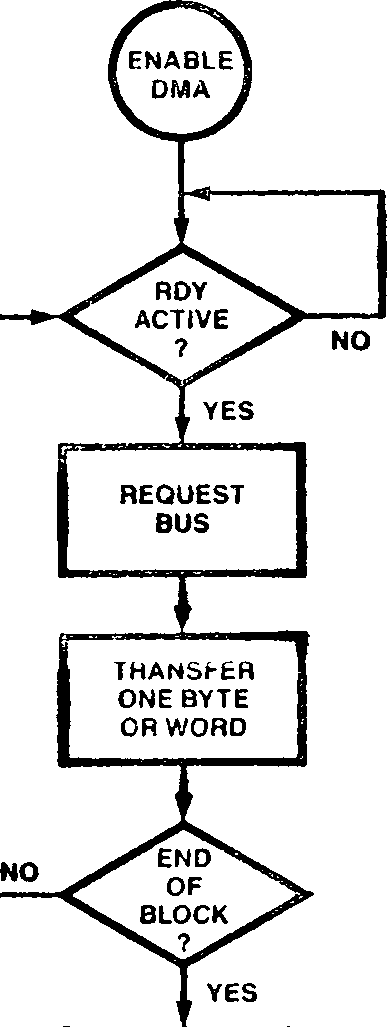


Figure 9-8. Modes of Operation

channel’s  
interrupt

Iransfers of unaligned data on 16-bit buses can be accomplished via byte transfers only. Long word transfers are used in applications where the Z280 MPU is acting as a DMA controller for a system with a 32-bit bus, such as a Z80,000-based system. During long word transactions, the Z280 MPU’s DMA channel provides only 24 bits of the address; the upper 8 bits of the 32-bit address have to be generated with external logic. Long word transfers are supported only in the flyby mode with the on-chip cache programmably disabled.

«

9.5.3 End-of-Process

An enable bit in the DMA Master Control register allows the Interrupt A input to be used as ar? end-of-process (EOP) input during DMA trans­actions. When enabled, transfers by DMA channels can be prematurely terminated by a low on the LOP (Interrupt A) line. Recognition of the EOP signal is not affected by the state of the Interrupt Request Enable bit for Interrupt A in the CPU’s Master Status register.

If the EOP signal goes active during the read portion of a flowthrough transaction, the DMA activity is aborted before the write portion of that transaction. If LOP becomes active during the write portion of a flowthrough transaction or during a flyby transaction, that transfer is completed before stopping the DMA operation.

When an active EOP signal terminates a DMA operation, the EOP Signaled (EPS) status bit in that channel’s Transaction Descriptor register is automatically set to 1 and the Enable bit in that same register is cleared to 0. If that Interrupt Enable bit is set to 1, an request to the CPU is generated.

The EOP signal is level-sensitive and shared by all four on-chip DMA channels. Thus, if an active EOP signal terminates the activity of one DMA channel and another DMA channel immediately requests the bus, the second DMA’s activity is terminated before any transactions can be generated if EOP is still active. In other words, the secund DMA channel also recognizes the EOP signal, and so on. Therefore, in order for the currently active DMA channel to be the only channel whose activity is terminated, EOP should be asserted for only one bus clock cycle in systems where the bus clock frequency is equal to or one-half of the processor clock frequency; EOP should be asserted for one-half of a bus clock cycle in systems where the bus clock frequency is one-fourth of the processor clock frequency.

If the end-of-process capability is enabled, a single input to the Z280 MPU can act as both the Interrupt A and the EOP signal; it acts as the Interrupt A Request line when the CPU controls the bus and as the EOP line when a DMA channel controls the bus. If an EOP signal terminates a

DMA operation, and that signal is still asserted when the CPU regains control of the bus, then the signal is interpreted as an interrupt request. Thus, a single signal can be used to stop DMA activity and generate an interrupt, if so desired. Note that the interrupt request generated by the DMA channel and the interrupt request generated by an active signal on the Interrupt A line are different interrupt requests, each with its own priority and its own enabling bit in the CPU's Master Status register.

* + 1. Priority Resolution

Prioritization of the four on-chip DMA channels is implemented via an internal ’’service request” latch. A DMA channel generates a service request, indicating that the channel needs to gain control □ f the bus, if that channel’s Enable bit in the Transaction Descriptor register is set to 1 and an active RDY signal is asserted. This service request signal is latched in the service request latch only if all preceding service requests have already been serviced (that is, there are no service requests active in the latch). Once a service request is latched, the service request latch is ’’closed” to all other service requests until the current requests are serviced; the latched requests are serviced in priority order, where DMA channel 0 has highest priority and DMA channel 3 has lowest priority. When all latched service requests have been serviced, the latch is ’’opened” so that new service requests can be latched.

This service request mechanism provides for non-preemptive prioritization' of DMA activity. For example, if DMA channel 1 requires servicing while the other channels are quiescent (that is, not currently controlling the bus or making a service request), channel 1’s service request is latched and the service request latch is closed. Thus, no other channel can preempt channel 1’s activity. If channels 0 and 2 activate service requests while channel 1 is being serviced, both those requests will be latched after channel 1\*s activity is completed, and channel 0 will be serviced next, followed by channel 2. No new service requests are latched until both channels 0 and 2 have been serviced, and so on.

All service requests from the on-chip DMA channels have priority over bus requests made via the BUSREQ input by external DMA controllers.

* + 1. DMA Linking

I he Z280 MPU ’ s on-chip DMA devices can be linked together to provide for DMA transfers to non-contiguous memory locations. In a linked configuration, one channel, called the master DMA, controls the actual data transfers to the memory and/or peripheral devices; the second channel, called the linked DMA, is used to load the master DMA’s control registers from memory when the master DMA completes an operation. The master DMA signals the linked DMA when a transaction is completed via an internal ’’ready” input to the linked DMA. The linked DMA then initiates the transfers that load the master DMA’s control registers from memory, allowing the master DMA to perform multiple data transfer operations without any CPU intervention.

Control bits in the DMA Master Control register allow DMA3 to be linked to DMA1, with DMA1 the master DMA and DMA3 the linked DMA, and DMA2 to be linked to DMAO, with DMAO the master DMA and DMA2 the linked DMA.

When the linked DMA loads the master DMA’s registers, the registers are written in the following order:

**e** Destination Address register (least significant

word) • Destination Address register (most significant

word)

Source Address register word)

register (most significant

(least

•••• f

significant

• Source Address

word) • Count register

• Transaction Descriptor register

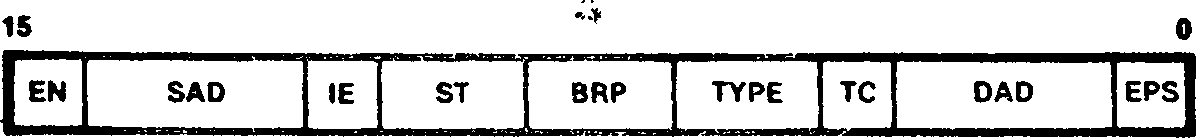
After the six words have been written to the master DMA, the master DMA deasserts the ready signal to the linked DMA and begins the new transfer operation. For Z-BUS configurations of the Z280 MPU, the linked DMA uses six word transactions on the bus to program the master DMA; for Z80 Bus configurations, the linked DMA uses twelve byte transactions to program the master DMA, with the least significant byte of each word being transferred first.

Control bits in the DMA Master Control register also allow DMAO to be proqrammably linked to the on-chip UARf’s receiver and DMA1 to be linked to the UARf’s transmitter. If so linked, an internal ’’ready” signal to DMAO is automatically generated when the UARf’s receive buffer is full. Similarly, an internal ’’ready” signal to DMA1 is automatically generated when the UARf’s transmit buffer is empty. The external ROY inputs are ignored while in this configuration.

The DMA Master Control register is cleared to all zeros by a reset, unless bootstrap mode is enabled during the reset operation (see sections 3.2.1 and 9.7). Bits 7 through 15 of this register are not used.

9.5.6.2 DMA Transaction Descriptor Register

Lach DMA channel has its own 16-bit Transaction



status informat ion.

Figure 9-10. Transaction Descriptor Register

Descriptor register. The Transaction Descriptor

register (Figure 9-10) describes the type of DMA

transfer to be performed and contains control and

9.5.6 DMA Registers

DMA registers consist of a DMA Master Control register that specifies the general configuration of all four channels, and a Transaction Descriptor register, Source Address register, Destination Address register, and Count register for each DMA channel. All DMA registers are accessed using word I/O instructions

9.5.6.1 DMA Master Control Register

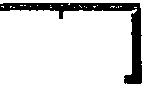
The 16-bit DMA Master Control register is illus­trated in Figure 9-9. The bit fields within this

**' End-of-Process Signaled (EPS).** This status bit is set to 1 automatically when an active End-of- Process signal prematurely terminates a DMA transfer. fhis bit can be set to 1 or cleared to

0 under software control.

register are described below.

15 о



SR1 SR0 EOP D3L D2L D1T DOR

Figure 9-9. DMA Master Control Register

**DMAO to Receiver Link (DOR).** While this bit is set to 1, DMAO is linked to the on-chip UARf’s receiver.

DMA1 to Transmitter Link (D1T). While this bit is

**Destination Address Descriptor (DAD).** This 3-bit control field determines the type of location (memory or 1/0) to be accessed as the destination port during DMA transfers, and whether the desti­nation address is to be incremented, decremented, or left unchanged between transfers, as shown in Table 9-4. When memory addresses are auto­incremented or auto-decremented, the incrementing or decrementing value is determined by the size of the data transfer, as specified in the ST field. 1/0 port addresses are always auto-incremented and auto-decremented by 1.

set to 1, DMA1 is linked to the on-chip UARf’s trcansmitter.

**DMA2 Link (D2L).** While this bit is set to 1, DMA2 is linked to DMAO.

**DMA? Link (D3L).** While this bit is set to 1, DMA? is linked to DMA1.

**End-of-Process (EOP).** While this bit is set to 1, the Interrupt A input acts as an End-of-Process input for the active DMA channel during DMA operations.

Table 9-4. Encoding of DAD and SAD Fields in DMA  
Transaction Descriptor Register

Encoding Address Modification Operation

000 Auto-increment memory location

001 Auto-decrement memory location

010 Memory address unmodified by transaction

011 Reserved

100 Auto-increment I /О location

101 Auto-decrement I/O location

110 I/O address unmodified by transaction

111 Reserved

enabled.

**Software Ready for DMAO (SRO).** While this bit is set to 1, DMAO requests use of the system bus if enabled.

**Software Ready for DMA1 (SR1).** While this bit is set to 1, DMA1 requests use of the system bus if

**Transfer Complete (TC).** This status bit is set to **1** automatically when the Count register has reached zero. This bit can be set to 1 or cleared to 0 under software control.

**Transaction Type (Type).** This 2-bit control field specifies the type of DMA operation to be performed, as shown in Table 9-5.

Table 9-5. Encoding of Type Field in Transaction Descriptor Register

|  |  |
| --- | --- |
| **Encoding** | **DMA Operation** |
| 00 | Flowthrough |
| 01 | Reserved |
| 10 | Flyby write (peripheral-to-memory) |
| 11 | Flyby read (memory-to-peripheral) |

**Bus Request Protocol (BRP).** This 2-bit control field determines the transfer mode for the DMA operation, as shown in Table 9-6.

**Table 9-6. Encoding of BRP Field in  
Transaction Descriptor Register**

**Encoding DMA Transfer Mode**

00 Single transaction

01 Burst

10 Continuous

11 Reserved

**Size of Transfer (ST).** This 2-bit control field specifies the size of the entity to be transferred during each DMA-controlled transaction, as shown in Table 9-7. If auto-increment or auto-decrement of a source or destination memory address is specified in the SAD or DAD fields, then the state of this field determines the size of the increment or decrement operation.

**Source Address Descriptor (SAD).** This 3-bit control field determines the type of location (memory or I/O) to be accessed as the source port during DMA transfers, and whether the source address is to be incremented, decremented, or left unchanged between transfers, as shown in Table

**DMA Enable (EN).** While this bit is set to 1, the DMA channel is enabled; while enabled, the DMA can request control of the system bus and, upon becoming bus master, initiate transactions on the bus. While this bit is a 0, the DMA channel is disabled and cannot request control of the bus. The DMA registers can be accessed regardless of the state of this bit.

For DMAO, a reset loads a 0100p| into the Trans­action Descriptor register. For the remaining three channels, the EN, IE, TC, and EPS bits are all cleared to 0 by a reset, and the remaining fields are unaffected.

* + - 1. Count Register

Each channel has a 16-bit Count register that is programmed to contain the number of DMA transfers to be performed. When the contents of the Count register reach zero (terminal count), further requests on the RDY line are ignored, and, if the IE bit in the Transaction Descriptor register is set to 1, an interrupt request is generated.

A reset loads a 0100|\_| into DMAO’s Count register; the other channels\* Count registers are unaffected by a reset.

completion of a DMA End-of-Process signal DMA operation. While no interrupt request is

Table 9-7. Encoding of ST Field in Transaction Descriptor Register

|  |  |  |
| --- | --- | --- |
| **Encoding** | **Size of Transfer** | **Number to Increment or Decrement By** |
| 00 | Byte | 1 |
| 01 | Word | 2 |
| 10 | Long word | 4 |
| 11 | Reserved |  |

**Interrupt Enable (IE).** While this bit is set to 1, the DMA channel generates an interrupt request to the CPU either when the Count register goes to zero, indicating the operation, or when an prematurely terminates a this bit is cleared to 0, generated.

* + - 1. Source Address and Destination Address

Registers

The 24-bit Source Address register and Destination Address register hold the port addresses used during DMA transfers. Ihese are physical addresses that are not translated by the MMU. In flyby mode, only one of these registers is used to supply the address for the transaction, as determined by the Type field in the Transaction Descriptor register. The contents of these registers can be automatically incremented or decremented by each DMA transaction, as determined by the SAD and DAD field in the Transaction Descriptor register.

The entire 24-bit Source Address or Destination

Address register is read and written via two word

accesses to the register. Twelve bits of the address are accessed by each word I/O operation; the format used when accessing these registers is shown in Figure 9-11.

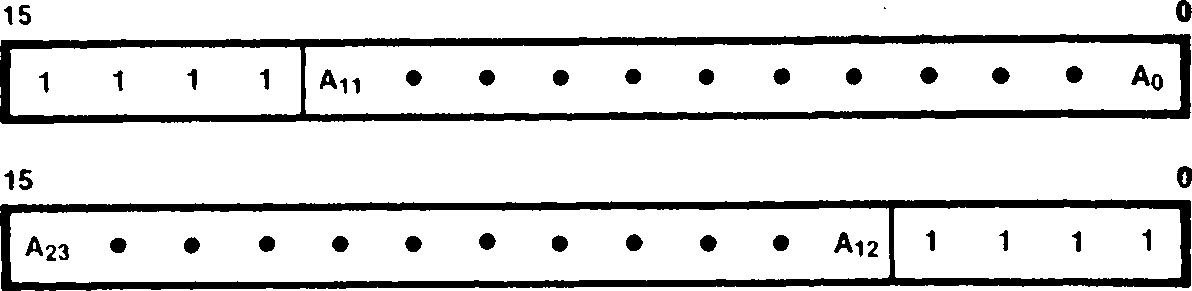


Figure 9-11. Source and Destination Address Registers Format

DMAO's Destination Address register is cleared to n\*»

0 by a reset; all other Source and Destination

Address registers are unaffected by a reset.

All DMA registers are located in I/O page ГГц. The DMA Master Control register is accessed at I/O port address FFxxIF. Table 9-8 lists the I/O port addresses for the other DMA registers. All DMA registers can be read or written using word I/O instructions.

**Table 9-8. I/O Addresses of DMA Registers**

**DMA Channel Register DMAO DMA1 DMA2 DMA3**

FFxxOO FFxxO8

Destination

Address (bits 0-11)

Destination Address (bits 12-23)

Source Address (bits 0-11)

Source Address (bits 12-23)

Count

Transaction Descriptor

FFxxOl FFxxO9

FFxxO2 FFxxOA

FFxxO3 FFxxOB

FFxxO4 FFxxOC

FFxxO5 FFxxOD

All addresses are in hexadecimal, “x” means “don’t care”.

FFxxIO FFxx18

FFxx11 FFxx19

FFxx12 FFxxIA

FFxx13 FFxxlB

FFxx14 FFxxIC

FFxx15 FFxxID

No checking is performed by the hardware to deter­mine if an invalid configuration is specified in the DMA registers, such as specifying word trans­actions on 8-bit data bus configuration of the

Z280 MPU; in such cases, DMA behavior is

unpredictable.

This section describes a typical sequence of events when a DMA channel is used in flowthrough or flyby mode to control data transfers.

Before a DMA channel can begin operation, that DMA channel must be configured for the particular application by loading its Destination Address, Source Address, Count, and Transaction Descriptor registers. DMA operations cannot take place while the EN bit in the Transaction Descriptor register is cleared to 0. Thus, the EN bit should be cleared to zero while configuring the DMA channel, and set to 1 as the last step in the configuration process; the EN bit can be set at the same time that the other bit fields in the Transaction Descriptor register are specified.

Once the EN bit is set to 1, the DMA channel requests use of the system bus only after an active RbY signal is received. The RDY signal is sampled by the DMA on the rising edge of each processor clock cycle. For DMAO and DMA1, the RbY signal is the logical OR of the external RbY input and the software RDY bit in the DMA Master Control register.

When the system bus is available for DMA transfers, the highest priority DMA channel with a request pending becomes the bus master. The priority of the on-chip DMA channels from highest to lowest is DMAO, DMA1, DMA2, and DMA3. The external Bus Request input has the next lowest priority after the on-chip DMA channels.

The number of data transfers performed by a DMA that has gained control of the bus is determined by the current transfer mode (single transaction, burst, or continuous) and the contents of the Count register. A DMA channel in single trans­action mode relinquishes the bus after a single data transfer; a DMA channel in burst mode relinquishes the bus when RDY is deasserted or when terminal count is reached; a DMA channel in continuous mode relinquishes the bus when the terminal count is reached. Regardless of the transfer mode, a DMA channel will relinquish the bus if an EOP is signalled or the terminal count is reached.

If the destination for a DMA-controlled data transfer is a memory location that corresponds to an entry in the on-chip memory (in either the cache or fixed-address mode), the on-chip memory is updated to reflect the new contents of that memory location.

For each DMA-controlled data transfer on the bus, that DMA's Count register is decremented by 1, regardless of the size of the data transferred. The Destination Address and Source Address registers might also be incremented or decre­mented, as determined by the DAD, SAD, and SI fields in the Transaction Descriptor register. When a DMA operation reaches completion, either by assertion of an EOP signal or by reaching terminal count (a count of 0) in the Count .register, the EN bit in the Transaction Descriptor register is automatically cleared to 0. If the IE bit is set to 1, an interrupt request to the CPU is generated. If the DMA operation terminated due to an active EOP signal, the EPS status bit is set to 1; if the DMA operation terminated due to reaching terminal count, the TC status bit is set to 1.

* + 1. DMA Programing: Linked DMAs

When two DMA channels are linked together, the master DMA's registers are written via memory-to-peripheral data transfers initiated by the linked DMA. Thus, to begin DMA operations, the linked DMA must be programmed to load the master DMA. While the linked DMA is being configured, the master DMA must be prohibited from asserting a RDY signal to the linked DMA. The internal RDY signal from the master DMA to the linked DMA is controlled by the TC status bit of the master DMA; therefore, before configuring the linked DMA, the TC bit of the master DMA's Transaction Descriptor register should be written with a 0. Then, the linked DMA is configured by writing to its registers. Finally, the TC bit in the master DMA should be set to 1; this causes the internal RDY signal to the linked DMA to go active, which in turn causes the linked DMA to request the bus and, upon acknowledgement of that request, initiates the transactions that program the master DMA.

The linked DMA must be configured for flowthrough- type data transfers. The transfer size must match the size of the external data bus (that is, byte for Z80 bus configurations and word for Z-BUS configurations). The Source Address register is loaded with the starting address of the memory block that holds the data to be written to the master DMA's registers; for the Z-BUS, this starting address must be even-valued (A0=0). The SAD field of the Transaction Descriptor register should specify an auto-increment or auto-decrement of the memory address. The Destination Address register must be set to FFxx00|\_| when DMA2 is the linked DMA, or FFxxO8^| when DMA3 is the linked DMA ("x" means don't care). The DAD field in the linked DMA's Transaction Descriptor register

should be set to 100^ (auto-increment I/O

address). Burst mode transactions must be specified. The contents of the Count register vary depending on the number of times that the linked DMA is required to reconfigure the master DMA.

When the master DMA has completed a transaction (terminal count is reached), an internal RDY signal to the linked DMA is activated. If the linked DMA is enabled, the linked DMA will generate the transactions that program the master DMA's registers. (The linked DMA's external RDY input is ignored when DMA linking is specified.)

When the linked DMA loads the master DMA's registers, the registers are written in the following order:

**0** Destination Address register (least significant word)

**0** Destination Address register (most significant word)

**e** Source Address register (least significant word)

о Source Address register (most significant word)

**e** Count register

**0** Transaction Descriptor register

After the six words have been written to the master DMA, the master DMA deasserts the ready signal to the linked DMA and begins the new "’transfer operation. For Z-BUS configurations of the Z280 MPU, the linked DMA uses six word transactions on the bus to program the master DMA; for Z80 Bus configurations, the linked DMA uses twelve byte transactions to program the master DMA, with the least significant byte of each word being transferred first.

Both the master and linked DMAs can be programmed to generate an interrupt request to signal the end of DMA activity. If the IE bit of the master DMA is set, an interrupt request is generated when the master DMA reaches terminal count and the linked DMA's TC bit is set (that is, when the last block has been transferred), or if EOP is asserted. If the IE bit in the linked DMA is set, an interrupt request is generated when the linked DMA reaches terminal count (that is, when the last block transfer has been programmed into the master DMA), or if EOP is asserted.

* + 1. DMA Programming: DMAs Linked to UART

The DOR and Dll bits of the DMA Master Control register specify whether DMAO is linked to the UART receiver and DMA1 is linked to the DARI transmitter, respectively.

When DMAO is linked to the UARI receiver, the state of the Source Address register and the SAD field in the Transaction Descriptor register do not affect DMA operation. Ihe Destination Address register is programmed with the starting address of the memory area or the address of the I/O device that will be used to store the received data; if the destination port is a memory block, the DAD field should specify an auto-increment or auto-decrement of the memory address. Flowthrouqh-type transactions and the byte transfer size must be specified. Single, burst, or continuous mode operation can be used.

When DMA1 is linked to the UARI transmitter, the Source Address register is programmed with the starting address of the memory area or the address of the 1/0 device that holds the data to be transmitted; if the source is a memory area, the SAD field should specify an auto-increment or auto-decrement of the memory address. The Destination Address register must be set to xxxx18^, and the DAD field to a 110|-|. Flowthrough type transactions and the byte transfer size must be specified. Single, burst, or continuous mode operation can be used.

9.6 UART

errors. Transmission and reception are performed independently.

The UART uses the same clock frequency for both the transmitter and the receiver. The UART’s clock input can be generated externally or internally. For externally generated clocks, Counter/Timer 1‘s input line is used as the source of the UART’s clock in addition to being an input to the counter/timer. The maximum external clock frequency is the CPU clock divided by 4. Alternately, the UART's clock can be provided by the output pulse from Counter/Timer 1, allowing the internal processor clock to be used for bit rate generation. The UART’s clock input is further scaled by a factor of 1, 16, 32, or 64 for clocking the transmitter and receiver.

The UART can be used in an interrupt-driven or polled environment. If enabled, separate transmit and receive interrupt requests are generated by the UART. Transmit interrupts occur when the transmitter’s data buffer is emptied, and receive interrupts occur when an entire character is received or an error is detected. In polled environments, status bits in UART registers can be read to determine if the transmit buffer is empty or receive buffer is full. As described in section 9.5.9, DMA channel 0 can be linked to the receiver and DMA channel 1 to the transmitter to provide for DMA-controlled transfers between the UART and memory.

The UART uses two external pins, Transmit (Tx) and Receive (Rx). Data that is to be transmitted is placed serially on the Transmit pin and data that is to be received is read from the Receive pin.

The on-chip

universal asynchronous receiver./

The UART contains five registers. UART operation

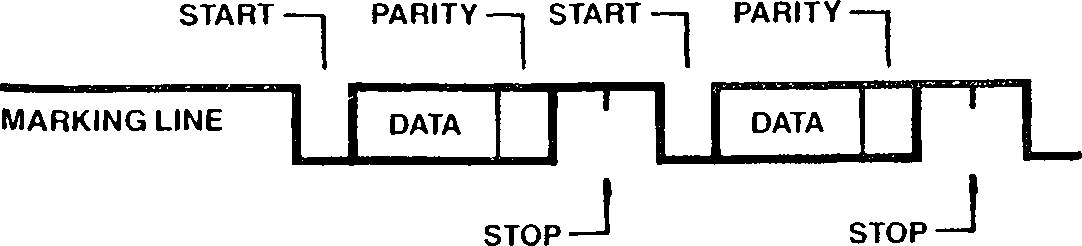
serial 1/0 capability.

The full-duplex UART

transmitter (UART) provides the Z280 MPU with

is controlled by three registers: the UART

Configuration register, which contains controls



transmits and receives

serial data using any

Figure 9-12 illustrates the general format for an asynchronous transmission using the Z280 MPU’s UART. Characters can contain five, six, seven, or eight bits, plus an optional even or odd parity bit. The transmitter can supply one or two stop bits per character. Break outputs can be produced by the transmitter at any time under program control; the receiver can detect breaks as well as parity errors, framing errors, and overrun

common asynchronous data communication protocol.

Figure 9-12. General Format for an  
Asynchronous Transmission

for both the transmitter and receiver, the Transmitter Control/Status register, and the Receiver Control/Status register. Received data is read from the Receive Data register, and data to be transmitted is written to the Transmit Data register.

* + 1. Transmitter Operation

Transmit operations are performed only л/hen the Transmitter Enable bit in the Transmitter Control/Status register is set to 1. In order to transmit data, the data character is written to the Transmit Data register. The UARI automati­cally adds the start bit, the programmed parity bit (if so specified), and the programmed number of stop bits to the data character to be trans­mitted. The (lumber of bits per character, the number of stop bits per character, and the type ofparity (even, odd, or none) is determined by the contents of the UARl Configuration register. When the transmit character size is five, six, or seven bits, the unused most significant bits in the Transmit Data register are ignored by the UARl.

Serial data is shifted out of the transmitter on the Tx pin at a rate egual to 1, 1/16th, 1/32nd, or 1/64th of the clock signal supplied to the UART, as determined by the contents of the UART Configuration register. Serial data is shifted on the falling edge of the clock input.

The Tx output line is held high (marking) when the transmitter has no data to send or is disabled. If transmit interrupts are enabled, an interrupt reguest is generated when the Transmit Data register is emptied. Under program control, break conditions can be generated, wherein the Tx line is held low (spacing) until the break command is cleared.

For each character assembled by the receiver, error flags in the Receiver Control/Status register indicate if an error condition was detected. These flags are loaded when the character assembly process is completed--that is, when the character is loaded into the Receive Data register from the receiver’s shift register. The receiver checks for parity errors, framing errors, and overrun errors for each received character.

A parity error occurs when the parity bit of the received character does not match the programmed parity, as determined by the contents of the UART Configuration register.

A framing error occurs if a character is assembled without any stop bits (that is, if a low level is detected for the stop bit). A built-in checking process prevents a framing error from being interpreted as a new start bit; detection of a framing error results in the addition of one-half

of a bit time to the point at a new start bit is begun.

which the search for

* + 1. Receiver Operation

An overrun error occurs if

a new character is

Receive operations are performed only when the Receiver Enable bit in the Receiver Control/Status register is set to 1. A low (spacing) condition

assembled and loaded

into the Receive Data

register before the previous character has been

read from that register.

Since the receiver is

on the Receive input line indicates a start bit; buffered by the Receive Data register in addition

if the low persists for at least one-half of a bit time, the start bit is assumed to be valid and the data input is sampled at mid-bit times until the entire character is assembled. Thus, reception is protected from transients on the input line by checking for a valid start bit one-half bit time after detecting a high-to-low transition on the Receive input; if the low does not persist (as with a transient), the character assembly process is not started. If the bit time is one clock period (the x1 clock mode), bit synchronization must be accomplished externally; received data is sampled on the rising edge of the clock.

to the receiver shift register, ample time is available for responding to a receiver interrupt and accepting a received character before the next character is assembled by the receiver.

* + 1. UART Registers

UART operation is controlled by three 8-bit registers: the UARl Configuration register, Transmitter Control/Status register, and Receiver Control/Status register. Data to be transmitted is written to an 8-bit Transmit Data register, and received data is read from an 8-bit Receive Data

Received characters are read from the Receive Data register. All UARl registers are accessed using

register. If parity is enabled, the parity bit is assembled as part of the character fur character lengths other than eight bits. If the resulting character is still less than eight bits, 1\*s are appended in the unused high-order bit positions.

byte 1/0 instructions.

9.6.3.1 UART Configuration Register

For example. Figure 9-13

illustrates how the

character register parity.

is  
when

assembled  
receiving

in the Receive Data

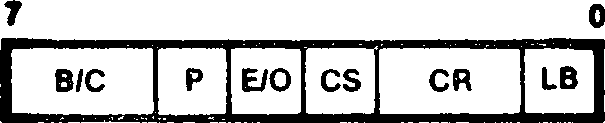
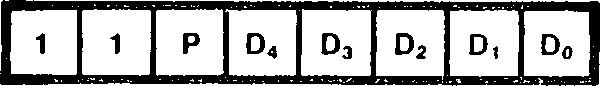
5-bit characters with

The 8-bit UARl Configuration register (Figure 9-14) contains control information for both the receiver and transmitter. The control fields

within this register are described below.

Figure 9-13. Byte Assembled by Receiver for 5-bit Character with Parity

Figure 9-14. UART Configuration Register



**Loop Back Enable (LB).** When set to 1, the UART is in local loopback mode; in this mode, the internal transmit data line is tied to the internal receiver input line and the external receiver input pin is ignored. Thus, all transmitted data is automatically received. When this bit is cleared to 0, the transmitter and receiver operate independentIу.

**Clock Rate (CR).** This 2-bit field determines the multiplier between the UART clock and data rates (that is, the number of clocks per bit time), as specified in Table 9-9. The same data rate is used by both the transmitter and receiver. If the X1 clock rate is selected, bit synchronization must be accomplished externally. In the X1 mode, the transmitter sends data on the falling edge of the clock and the receiver samples data on the rising edge of the clock.

Table 9-10. [f this field is changed while a character is being transmitted or received, the results are unpredictable.

**Table 9-10. BC Field of UART Control Register**

**BC Field Bits per Character**

00 5

**01** 6

**10** 7

**11** 8

A reset clears the UARl Configuration register to all zeros, unless bootstrap mode is selected (see

section 9.7).

* + - 1. Transmitter Control/Status Register

| **CR Field** | **UART Clock Rate** |
| --- | --- |
| 00 | X1 |
| 01 | X16 |
| 10 | X32 |
| 11 | X64 |

Table 9-9. CR Field of UART

Configuration Register

The 8-bit Transmitter Control/Status register, shown in Figure 9-15, specifies the operation of the UART transmitter, as described below.

7 0

I EN IE 0 SB BRKFRCVAL BE 1

Figure 9-15. Transmitter Control/Status Register

set to

both the transmitter and receiver, as specified in

**Clock Select (CS).** The state of this bit specifies the clock input for the UART. When this bit is set to 1, counter/timer I’s output pulse supplies the UART clock. When this bit is cleared to 0, counter/timer 1’s clock input pin provides the UART clock signal, thus allowing the use of an externally-generated clock. The content of the IPA field of C/T 1’s Configuration register does not affect these UART clocking modes.

**Parity (P).** When set to 1, an additional bit position (in addition to the number of bits per character specified in the BC field) is added to each transmitted character and expected in each received character; this additional bit is the parity bit. Parity bits in received characters are assembled as part of the character for character lengths of less than 8 bits.

**Parity Even/Odd (E/O).** If parity is specified (P = 1), this bit determines whether an odd or even parity bit is added to transmitted characters and whether odd or even parity is checked for in received characters. E/0 = 1 specifies even parity and E/0 = 0 specifies odd parity. If P = 0, then this bit is ignored.

**Bits per Character (B/C).** This 2-bit field determines the number of bits per character in

**Transmitter Buffer Empty (BE).** This status bit is automatically set to 1 whenever the Transmit Data register becomes empty and cleared to 0 whenever a character is loaded into the Transmit Data register. The BE bit is controlled by the UART circuitry; it can be read via an 1/0 read but is unaffected by an 1/0 write to this register. A reset loads a 1 into this bit.

**Value (VAL).** This bit determines the value of the bits transmitted by the UART when the FRC bit is set to 1 and ’’dummy” characters are loaded into the Transmit Data register. When the VAL bit is

a mark character (all 1s)

transmitted; when the VAL bit is cleared to 0, a break character (all 0s) is transmitted.

**Force Character (FRC).** When this bit is set to 1, writing a character to the Transmit Data register causes the transmitter output to be held high or low (depending on the state of the VAL bit) for the length of time reguired to transmit the character. Note that characters written to the Transmit Data register are not themselves trans­mitted while FRC is set to 1. When FRC is cleared to 0, the transmitter operates normally, sending characters that are written to the Transmit Data register.

**Send Break (BRK).** When this hit is set to 1, the transmitter is forced into the spacing condition, wherein the transmit data output is forced to 0. When this bit is cleared to 0, normal transmitter operation resumes.

**Stop Bits (SB).** The state of this bit determines the number of stop bits appended to each character by the transmitter. Setting this bit to 1 specifies two stop bits per character; clearing this bit to 0 specifies one stop bit per character.

**Transmitter Interrupt Enable (IE).** When this bit is set to 1, an interrupt reguest is generated whenever the Transmit Data register is emptied. When this bit is cleared to 0, no tranmsit inter­rupts are generated.

**Transmitter Enable (EN).** When this bit is cleared to 0, the transmitter is disabled and the transmitter output line is held high (marking). When this bit is set to 1, the transmitter is enabled and operates as specified by the UART Configuration register and the Transmitter Control/Status register. If this bit is cleared while a character is in the process of being transmitted, transmission of that character is completed.

A reset sets the Transmitter Control/Status register to a 01^. Bit 5 of this register is not used.

* + - 1. Receiver Control/Status Register

The 8-bit Receiver‘Control/Status register, shown1 in Figure 9-16, specifies the operation of the UART receiver, as described below.

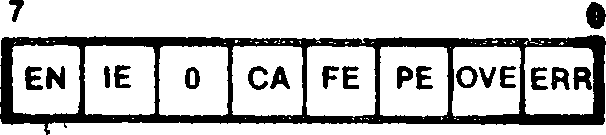


Figure 9-16. Receiver Control/Status Register

**Parity Error (PE).** When parity is enabled (P - 1 in the UART Configuration register) this bit is automatically set to 1 if a character is received without the specified parity. This bit is latched; once set, it remains set until cleared via software.

**Receiver Overrun Error (OVE).** This bit is automatically set to 1 if a new character is assembled and loaded into the Receive Data register before the previous character has been read from that register. Only the most recently received character is flagged with this error, but once this character is read, the OVE bit remains latched until cleared via software.

Receiver Character Available (CA). This bit is automatically set to 1 when a received character is available in the Receive Data register and automatically cleared to 0 when the Receive Data read. This bit is controlled by UART it can be read via an 1/0 read but altered by an 1/0 write to this register.

register is circuitry;

cannot be

**Receiver Interrupt Enable (IE).** When this bit is set to 1, an interrupt reguest is generated whenever the receiver has a character available in

the Receive Data register or when a receiver error is detected.

**Receiver Enable (EN).** When set to 1, receiver operation is enabled. This bit should be set after programming the UART Configuration register.

The Receiver Control/Status register is cleared to all zeros by a reset, unless bootstrap mode is \* selected (see section 9.7). Bit 5 of this

register is not used.

All UARI registers are in 1/0 page FE and are accessed via byte 1/0 instructions. Table 9-11 lists the 1/0 port addresses for the UART registers. « .

**Table 9-11. I/O Addresses of UART Registers**

**Receiver Error (ERR).** This bit is the logical OR of the PE, OVE, and FE bits.

**Framing Error (FE).** This bit is automatically set to 1 if the receiver detects a framing error when assembling the received character. Detection of a framing error adds an additional one-half bit time to the character to ensure that the framing error is not interpreted as a new start bit. This bit is not latched; once set, it remains set only until a new character is assembled and shifted into the Receive Data register.

**I/O Port**

**Register Address**

UART Configuration Register FExxIO

Transmitter Control/Status Register FExx12

Receiver Control/Status Register FExx14

Receive Data Register FExx16

Transmit Data Register FExx18

All addresses are in hexadecimal, "x” means "don’t care”.

9.6.4 UART Operation

Operation of the UART’s transmitter and receiver are enabled by the Transmitter Enable and Receiver Enable control bits in their respective control/status registers. Before enabling the UART by setting one of those bits, the UART’s configuration must be determined by programming the UART Configuration register. If the UART Configuration register is to be altered during system operation, the transmitter and receiver should be disabled before writing to the Configuration register, and then re-enabled afterwards. » ■ T

Once enabled, the UART can be used in an interrupt-driven or polled environment. Separate transmit and receive interrupts are controlled by the interrupt enable bits in the control/status registers. Receive interrupts are generated whenever a new character is available in the Receive Data register or when an error is detected. Transmit interrupts are generated whenever the Transmit Data register is emptied.

For polled environments, the Character Available bit in the Receiver Control/Status register must be monitored to determine when a character is to be read from the Receive Data register; this bit is automatically cleared when the received data is read. For transmitting characters, the Transmit Buffer Empty flag should be checked before writing to the Transmit Data register to prevent the overwriting of transmitted data.

The error flags in the Receiver Control/Status register are loaded at the same time that the received data character is moved from the receiver’s shift register to the Receive Data register. Since the parity and receiver overrun error flags are latched, the error status reflects any errors in the current character in the Receive Data register plus any parity or overrun errors that have been detected since the last write to the Receiver Control/Status register. To maintain correspondence between the state of the error flags and the data in the Receive Data register, the flags in the Receiver Control/Status register should be read before the data.

and clearing the VAL bit to 0 causes a break condition on the transmit data output each time a character is loaded into the Transmit Data register; this break output persists for the same amount of time that it would have taken to transmit the data written to the Transmit Data register had the FRC bit been 0. Note that the characters written to the Transmit Data register while the FRC bit is set to 1 are not actually transmitted.

9.7 UART BOOTSTRAPPING OPTION

The on-chip UART and DMA Channel 0 can be used to automatically initialize the Z280 MPU’s memory with values received by the UART following a reset. This system bootstrapping capability permits ROMless system configurations, where memory is initialized using a serial link prior to the first Z280 MPU instruction fetch after the reset.

As described in Section 3.2.1 and Chapter 11, bootstrap mode is selected by driving WAIT low and AD$ high while RESET is asserted. The appropriate UART and DMA registers are automatically programmed as shown in Table 9-12 as a result of selecting bootstrap mode. The UART is initialized to receive data in 8-bit characters with odd parity, an external clock source, and a x16 clock rate. DMA Channel 0 is initialized with the link to the UART receiver and end-of-process capability enabled, and set up for flowthrough byte transfers in continuous mode. The destination address starts at memory location 0, with an autoincrement after each transfer, and a transfer count of 256 (100H).

**Table 9-12. Reset Value of UART and DMA  
Registers When Bootstrap Mode is Selected**

**Register**

**UART Registers**

UART Configuration register E2

**Initial Hex  
Value**

Receiver Control/Status register 80

**DMA Registers**

Once the transmitter has been enabled, there are two ways to produce a break output on the transmit data line. Setting the BRK bit in the Transmitter Control/Status register forces a break condition on the transmit data output until that bit is cleared. Alternatively, setting the FRC bit to 1

[DMA Master Control register 0011](#bookmark443)

[Channel 0 Transaction Descriptor register 8100](#bookmark440)

Channel 0 Destination Address register 000000

Channel 0 Source Address register . Undefined

Channel 0 Count register 0100

If bootstrap mode is specified, the Z280 CPU automatically enters an idle state when RESET is deasserted. A minimum of 15 processor clock cycles must elapse after RESET is deasserted before tranmission of data to the UART receiver begins. DMA Channel 0 is then used to transfer characters received by the UART into memory. The data received is placed in memory starting at physical address 0. If an error is detected by the UART receiver, the Transmit Output (Tx) line is driven low; external circuitry can use this signal to restart the initialization procedure, if so desired. After 256 bytes of data have been received and transferred to memory, the Z280 CPU automatically begins execution with an instruction fetch from memory location 0.

**Chapter 10.**

**Multiprocessor Configurations**

10.1

INTR

UCTION

The Z280 MPU architecture provides support for

(Figure 10-1): slave processors, tightly coupled multiple CPUs, loosely coupled multiple CPUs, and coprocessors.

four types of multiprocessor configurations

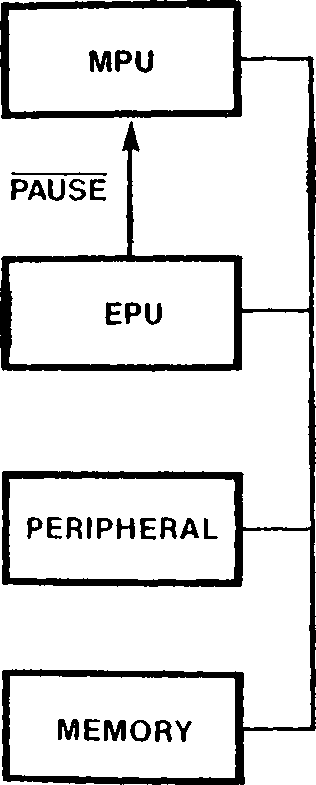
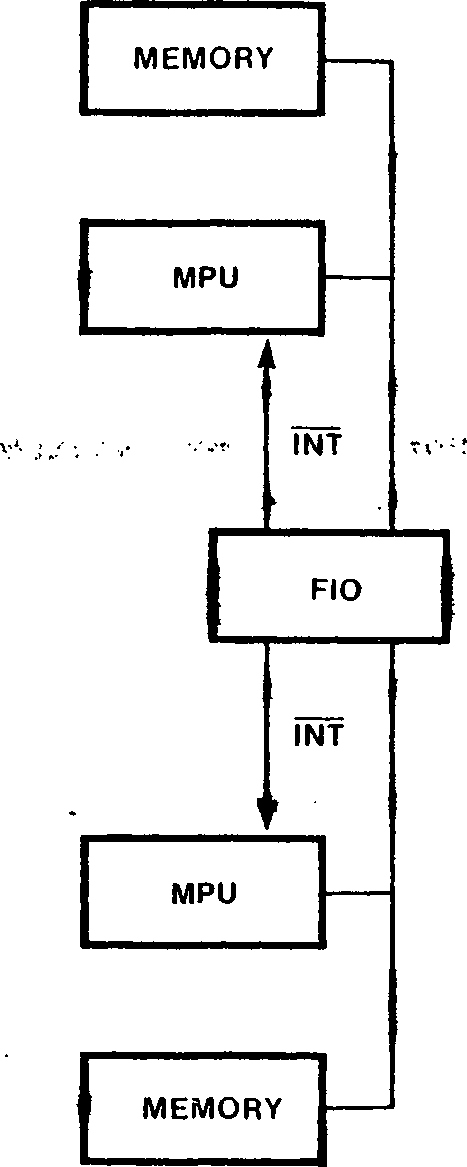
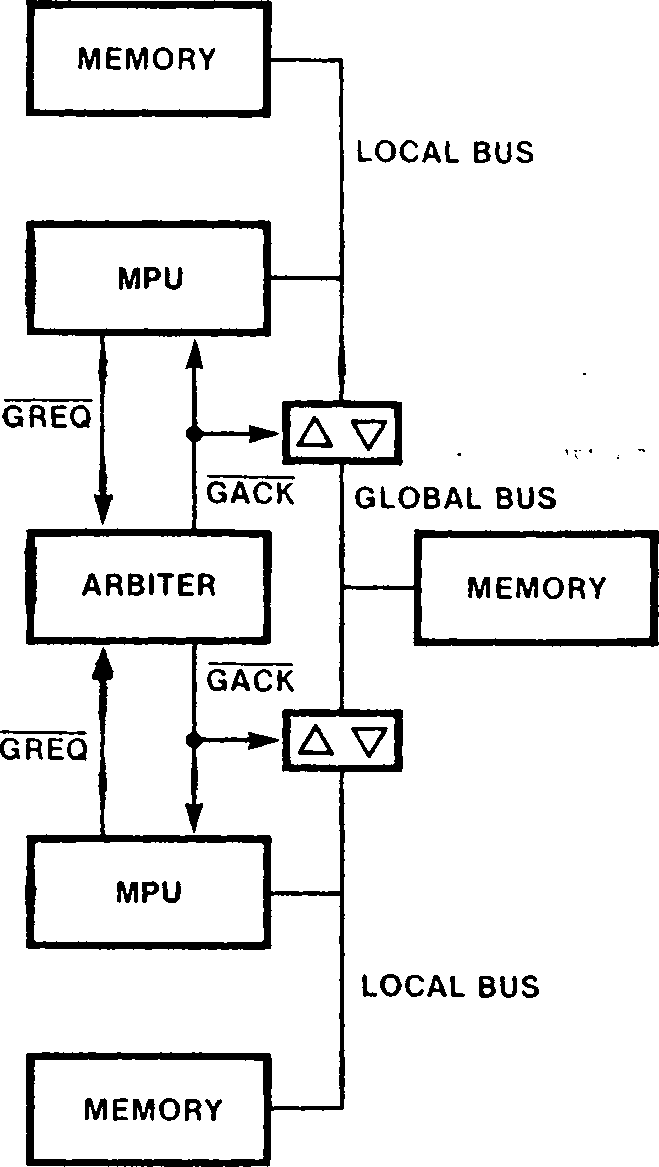
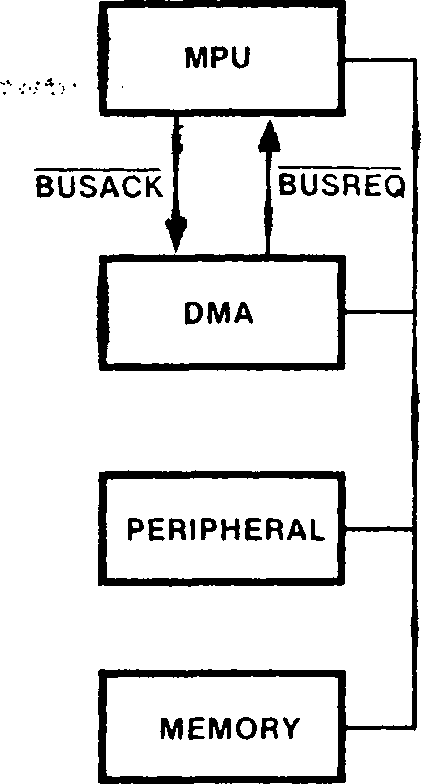
a) SLAVE PROCESSOR

b) TIGHTLY COUPLED MULTIPLE CPU

Figure 10-1. Multiprocessor Configurations

c) LOOSELY COUPLED MULTIPLE CPU

d) COPROCESSOR



* 1. SLAVE PROCESSORS

Slave processors, such as the Z8016 DMA Transfer Controller or other DMA devices, perform dedicated functions asynchronously to the CPU. The CPU and slave processors share a local bus, where the CPU is the default bus master. In order for a slave processor to use the bus, it must request control of the bus from the CPU and receive an acknowledgement of that request.

Two Z280 MPU signals are provided for supporting slave processors: BUSREQ and BUSACK. A bus request is initiated by pulling the BUSREQ input low. Several bus requestors may be wire-ORed to the BUSREQ pin; priorities are resolved external to the MPU, usually by a priority daisy chain. The external BUSREQ signal generates an internal, synchronous BUSREQ. If this signal is active at the beginning of any bus cycle, the Z280 MPU will relinquish the bus at the end of that bus cycle (with the exception of the TSET instruction, where the read-modify-write cycle is atomic). The MPU suspends execution of the current instruction and gives up control of the bus by 3-stating all address, address/data, bus timing, and bus status output pins. The BUSACK output is then asserted, signaling that the bus request has been accepted and the bus is free for use by the slave processor. The Z280 MPU remains in the bus disconnect state until BUSREQ is deasserted.

The BUSREQ input is sampled during each processor clock period by the external bus interface logic of the Z280 MPU. If BUSREQ is sampled active low while the Z280 MPU is involved in an internal operation, the external bus is relinquished to the bus requestor immediately. Internal processing can continue until a transaction involving the external bus is required; the MPU then suspends activity until regaining control of the bus. If BUSREQ is sampled active during a CPU-generated transaction on the external bus, the bus is not relinquished nor CPU activity suspended until the current transaction is completed.

The Z280 MPU regains control of the bus after BUSREQ rises, continuing execution from the point at which it was suspended. Any bus requestor desiring control of the bus must wait at least two bus cycles after BUSREQ has risen before asserting BUSREQ again.

In the case of simultaneous bus requests from multiple sources, the on-chip DMA channels have higher priority than external slave processors in Z280 MPU systems. After reset, the Z280 MPU acknowledges an active BUSREQ signal before performing any transactions.

* 1. TIGHTLY COUPLED MULTIPLE PROCESSORS

Tightly coupled multiple CPUs execute independent instruction streams from their own (local) memory locations and communicate through shared memory locations on a common (global) bus. Each CPU is the default master of its local bus, but the global bus master is chosen by an external arbiter.

The Z280 MPU’s multiprocessor mode of operation supports tightly coupled multiple CPU configurations. This mode is also useful when configuring the Z280 MPU as an I/O processor in a distributed processing system. Multiprocessor mode is selected by setting the Multiprocessor Configuration Enable (MP) bit in the Z280 CPU’s Bus Timing and Initialization register (see Section 3.2.1). While in the multiprocessor mode,1- the Z280 MPU is able to support both a local bus and a global bus. The Z280 CPU is the default bus master of the local bus, but must make a request and receive an acknowledgement before performing transactions on the global bus. Only memory

transactions can be performed on the global bus; I/O transactions always use the local bus. The range of memory addresses dedicated to the global and local buses is determined by the contents of the CPU’s Local Address register.

While in the multiprocessor mode, Counter/Timer O’s I/O and IN pins are used as global bus request (GREQ) and global bus acknowledge (GACK) signals, respectively. GREQ is a three-state output; an active low signal on this line requests use of the global bus. An active low level on the GACK input acknowledges a global bus request.

* + 1. The Local Address Register

During each memory transaction while in multi­processor mode, the Z280 CPU uses the Local Address register to determine if that transaction is to occur on the local or global bus. The Local Address register includes a 4-bit Base field and a 4-bit Match Enable field (Figure 10-2). For each bus transaction, the four most-significant bits of the physical address (address bits A20 through A23) are compared with the 4-bit Base field; the Match Enable field specifies which bits are going to be used during this comparison. If all the corresponding address bits match the Base field in the bit positions specified by the Match Enable field, then the bus transaction can proceed on the local bus without requesting the global bus. If there is a mismatch in at least one specified bit position, then the global bus is requested and the bus transaction does not proceed until the global bus acknowledge signal is asserted. (See section 3.2.3.)

0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ME23 | ME22 | ME21 | ME20 | B23 | B22 | B21 | в 20 |

Figure 10-2. Local Address Register

10.3^2 Bus Request Protocols

While in the multiprocessor mode, the BUSREQ and BUSACK signals control use of the local bus in the same manner as described in section 10.2. When a local bus request is granted, as indicated by an active BUSACK signal, the CPU places all output signals, including GREQ, in the high-impedance state.

When in control of its local bus, a Z280 CPU can initiate transactions with devices on the global bus that are shared with other CPUs. At any one time, only one CPU can control transactions on the global bus. Control of the global bus is arbitrated by external circuitry. Before initiating a transaction on the global bus, the CPU requests control of the global bus from the external arbiter circuitry by asserting GREQ and waiting for an active GACK in response. (The timing diagrams for global bus requests are shown in Figures 12-15 and 13-19.) The GACK input is asynchronous to the CPU clock; the Z280 CPU synchronizes GACK internally. Once GACK is asserted, the CPU performs the transaction on the global bus. The CPU then deasserts GREQ and waitsfor the arbiter circuit to deassert GACK.

always relinquishes the global bus by deasserting GREQ after each global transaction is completed, except during execution of a Test and Set (TSET) instruction (both the data read and write are during a burst-mode memory transfer (the entire sequence of burst-mode memory reads is completed ►

The CPU

before relinquishing the global bus).

A state diagram of the bus request

protocol is

completed before relinquishing the global bus) or

shown in Figure 10-3.

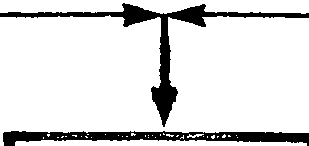
STATE 0

GREQ = H

BUSACK = H BUS = 2ST

(BUSREQ =  
(GACK = H)

(BUSREQ = H)\*(GACK = H) • (NEED—GBUS = H)



GACK = L

STATE 1

GREQ = 3ST  
BUSACK = L  
BUS = 3ST

D

BUSREQ = H

STATE 3

ERROR

STATE 2

GREQ = L  
BUSACK = H  
BUS = 2ST

E I (GACK = L)»(BUSREQ = H)

(GACK = L) •  
(BUSREQ = L)

STATE 4

GREQ = L  
BUSACK = H  
BUS = 2ST

(GACK = L)« [(BUSREQ = L)

4-(NEED—GBUS = L)J

G1GACK = H

ERROR

GREQ = H  
BUSACK = H  
BUS = 2ST

GACK = H

NOTES: Interface signals are High (H), Low (L), High or Low (2ST), or 3-stated (3ST).

NEED\_\_GBUS is an active High signal internal to the CPU.



Transition Legend

A A local bus request occurs.

В The global bus arbiter grants control of the global bus when no global bus request is pending. This is an error. The CPU remains in State 0.

C The CPU requests the global bus in response to the internally generated signal NEED\_GBUS.

D The local bus master relinquishes the bus.

E The global bus arbiter grants the global bus to the CPU while no local bus request is pending.

F The global bus arbiter grants the global bus to the CPU while a local bus request is pend­ing. The local bus request has preempted the CPU.

G The global bus arbiter reclaims the global bus before the CPU relinquishes the global bus. This is an error. The CPU's response to this error is undefined.

H The CPU relinquishes control of the global bus when it no longer needs the global bus or in response to a local bus request.

I The global bus arbiter reclaims the global bus.

State Legend

State 0 The CPU controls the local bus and is neither requesting nor controlling the global bus.

The CPU can perform transactions on the local bus.

State 1 The CPU has granted the local bus.

The CPU cannot perform transactions.

State 2 The CPU controls the local bus and is  
requesting the global bus.

The CPU cannot perform transactions.

State 3 The CPU controls the local and global buses.

The CPU can perform transactions on the global bus.

State 4 The CPU controls the local bus and is relinquishing control of the global bus. The CPU cannot perform transactions.

Figure 10-3. State Diagram for CPU Bus Request Protocol

While a Z280 CPU is asserting GREQ and waiting for an active GACK, if BUSREQ is asserted before GACK, the CPU releases the global bus request after GACK is asserted without performing any transactions.

The on-chip DMA channels may also initiate transactions on the global bus. During each DMA-controlled transaction, memory addresses generated by a DMA channel are compared to the contents of the Local Address register to determine if the global bus is to be requested, in the same manner as CPU-controlled bus transactions.

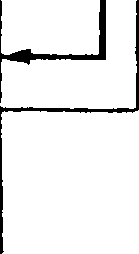
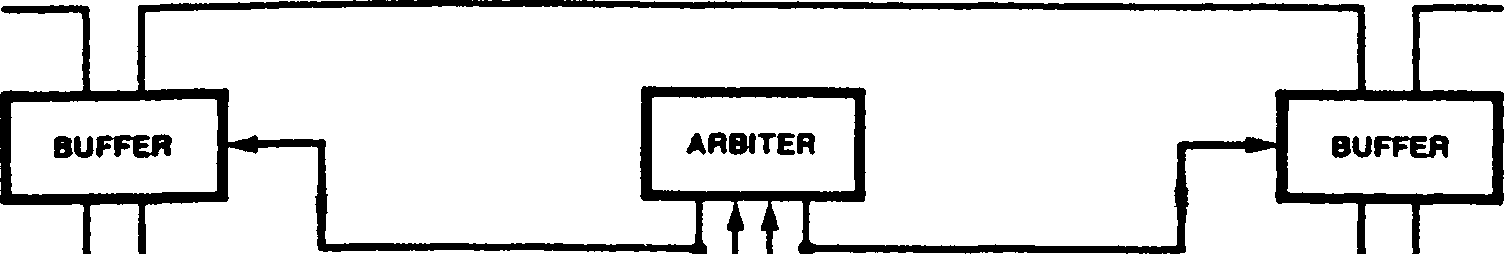
If the automatic memory refresh mechanism is enabled, refresh cycles are inhibited while either the CPU or a DMA channel has requested the global bus but not yet received the global bus acknowledge. No refresh transactions are ever performed on the global bus.

10.3.3 Examples of the Use of the Global Bus

The Z280 MPU’s multiprocessor mode of operation facilitates the development of tightly coupled multiprocessor systems and systems using the Z280 MPU as a front-end I/O processor.

Figure 10-4 is a block diagram illustrating the use of multiple Z280 MPUs as tightly-coupled processors. Access to the global memory via the global bus is controlled by a centralized bus arbitration circuit. The GACK circuit controls the buffers that connect or isolate the global bus from each MPU’s local bus. Each Z280 MPU can access its local memory independent of the other MPU’s activity. Only one MPU at a time can access the shared global memory. Note that memory-mapped I/O devices could also be shared using the global bUS - ■ г' .

GLOBAL  
MEMORY



GLOBAL BUS

LOCAL BUS

6aCR

GRES

MPU

|  | **GACK GrtEQ**  **MPU** | **LOCAL BUS** |
| --- | --- | --- |

LOCAL  
MEMORY

I/O  
DEVICE

LOCAL  
MEMORY

I/O  
DEVICE

Figure 10-4. Tightly Coupled Processors with Shared Global Memory

Figure 10-5 shows a tightly coupled multiple Z280 MPU system without a global memory, where each processor can directly access the local memory of the other processor. For this system, priority resolution logic would control both the local and global bus requests. A global bus request from

one processor is used to generate a local bus request to the other processor. When one

processor generates a global bus request, an active GACK signal is not returned to that processor until the other processor’s local bus is available, as indicated by BUSACK.

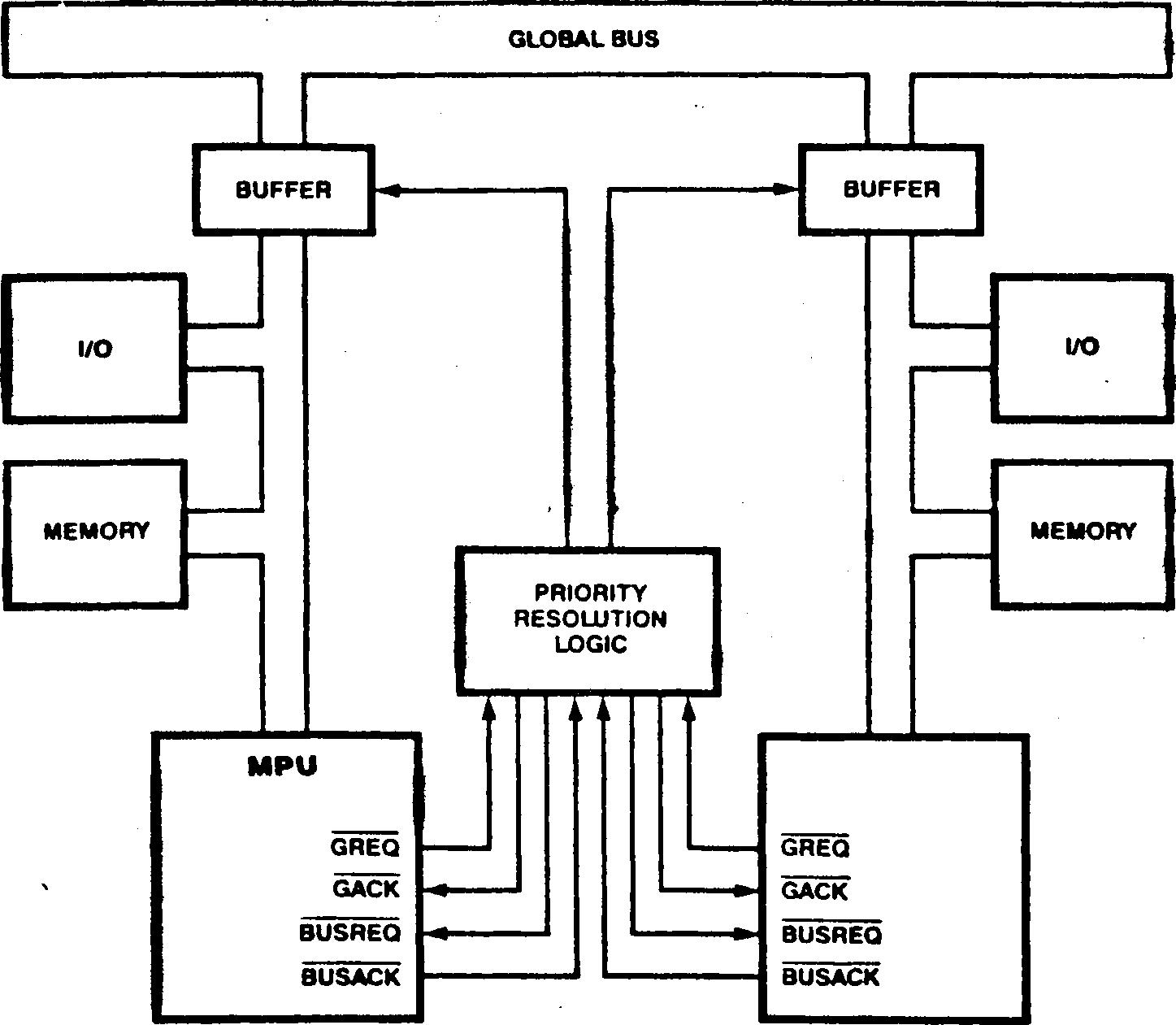
Although both Figure 10-4 and 10-5 show only two tightly coupled processors, more processors could be added to these systems in a similar manner.

Figure 10-5. Tightly Coupled Processors without Global Memory

Figure 10-6 illustrates the use of a Z280 MPU as an I/O processor in a Z8000-based system. The

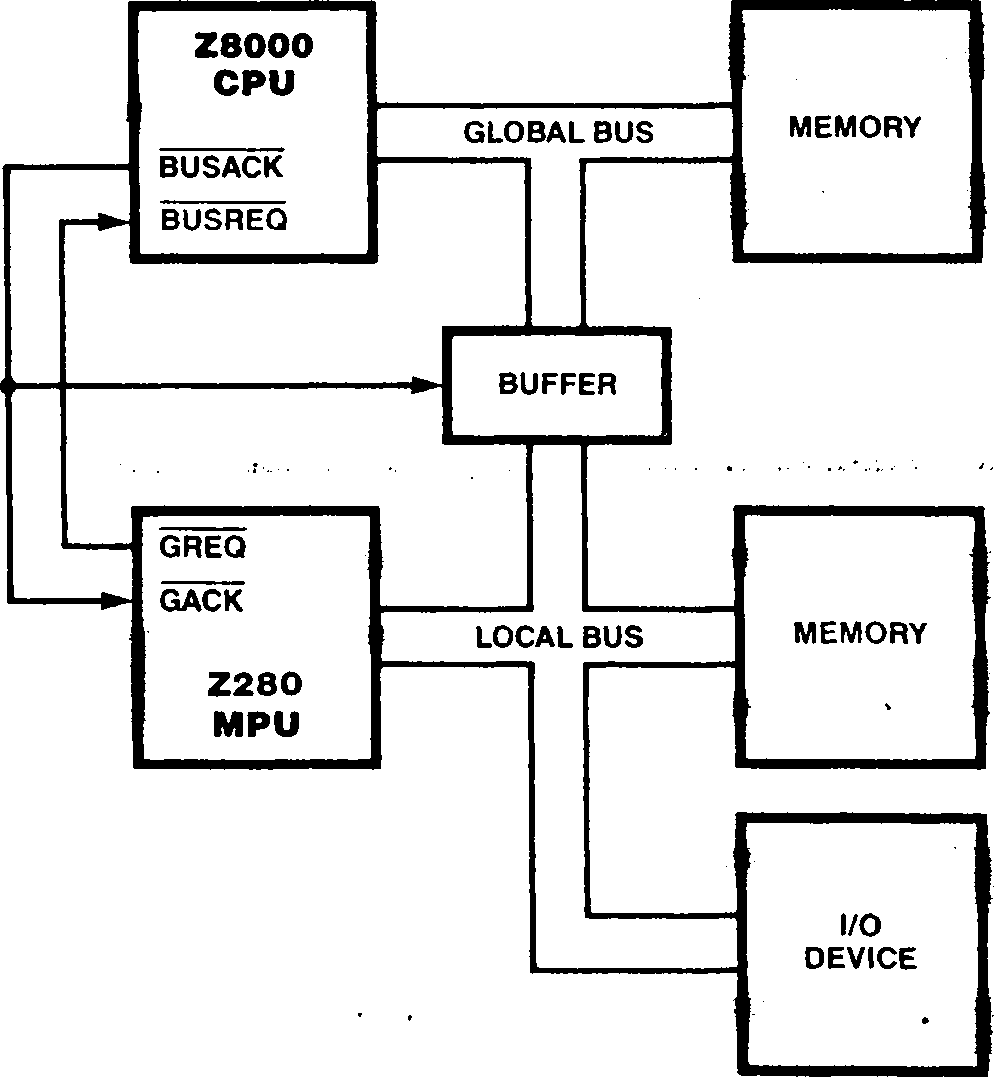
Z280 MPU’s GREQ signal is used as the bus request signal to the Z8000 CPU; the Z8000 CPU’s BUSACK signal is input directly to the Z280 MPU’s GACK, as well as controlling the buffers that normally isolate the Z280 MPU’s local bus from the Z8000 CPU’s bus.

Figure 10-6. Z280 MPU as an I/O Processor

10Л LOOSELY COUPLED MULTIPLE CPUS

Loosely coupled multiple CPUs generally communicate through a multiple-port peripheral, such as the Z8038 FIO (FIFO buffer I/O unit). The Z280 MPU’s I/O and interrupt facilities and the on-chip DMA channels support loosely coupled multiprocessing with the Z280 MPU.

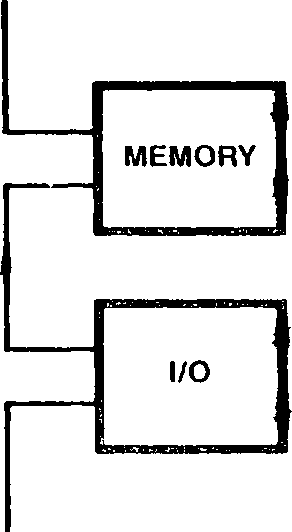
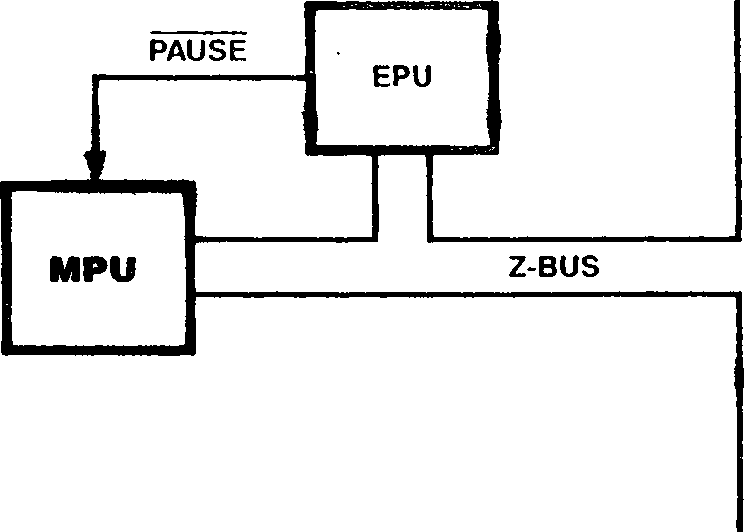
10.5 COPROCESSORS AM) THE EXTENDED PROCESSING ARCHITECTURE

can be used to execute complex, time-consuming tasks in order to unburden the CPU. EPUs connect directly to the Z-BUS; no extra external logic is required to interface an EPU to a Z280-based system (Figure 10-7). As the Z280 CPU fetches and executes instructions, the EPU continuously monitors the instruction stream on the bus. A special group of instructions, called extended instructions, are processed by EPUs. When the Z280 CPU encounters an extended instruction, it performs any specified data transactions, but otherwise assumes that the instruction will be

The Zilog Extended Processing Architecture (EPA) provides a flexible and modular approach to expanding the capabilities of the Z280 MPU through the use of coprocessors called Extended Processing Units (EPUs). The Extended Processing Architec­ture is available on the Z-BUS configurations of the Z280 MPU, but not the Z80 Bus configurations. Up to four EPUs can be connected to a single Z280 MPU.

An Extended Processing Unit is a coprocessor thatrecognized and handled by an EPU. (In systems without EPUs, extended instructions can be used to generate a trap condition.) Thus, when EPUs are added to a system, the instruction set is expanded to include the extended instructions applicable to those EPUs, thereby boosting the processing power of the whole system. The Z280 CPU and EPUs work together like a single central processor; a system with EPUs can be thought of as a system whose central processor consists of 1 + N separate devices, where N is the number of EPUs in the system.

Figure 10-7. EPU Connection in Z280 MPU System >я



The underlying philosophy of the Extended Processing Architecture is that the CPU is an instruction processor; that is, the CPU fetches an instruction, fetches data associated with that instruction, performs the specified operation, and stores the result. Extending the number of operations that can be performed does not affect the instruction fetch and address calculation portion of the CPU activity. The extended instructions exploit this feature. The CPU is responsible for fetching instructions, performing address calculations, and generating the timing signals for bus transactions; however, the actual data manipulation for extended instructions is handled by an EPU. Both the CPU and EPUs are, therefore, controlled via a single instruction stream, eliminating many significant system software and bus contention problems that can occur with other multiprocessing configurations.

* + 1. Extended Instructions

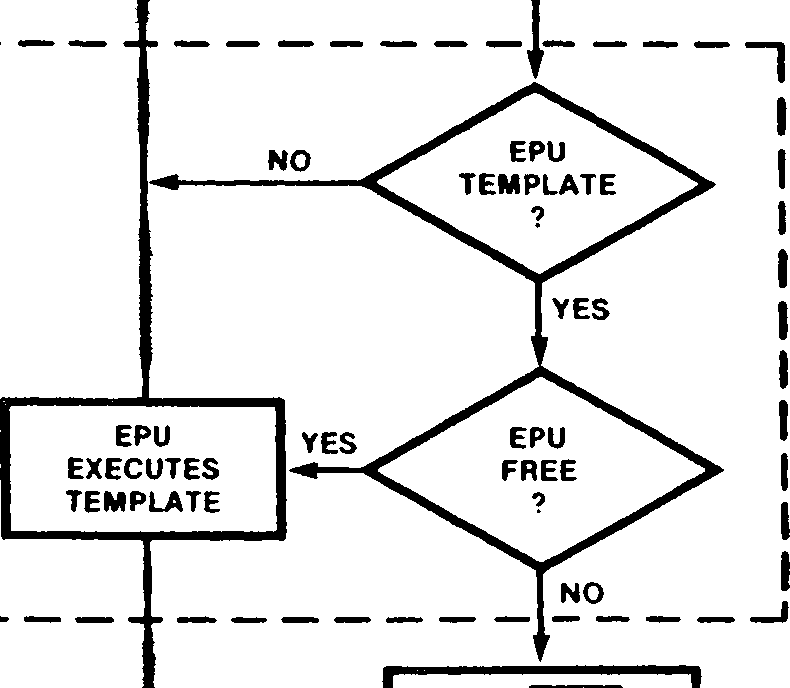
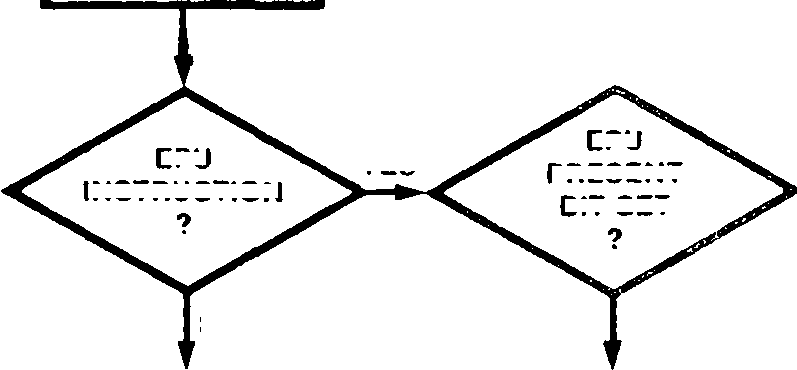
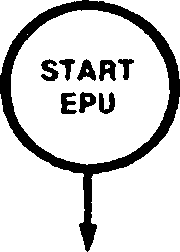
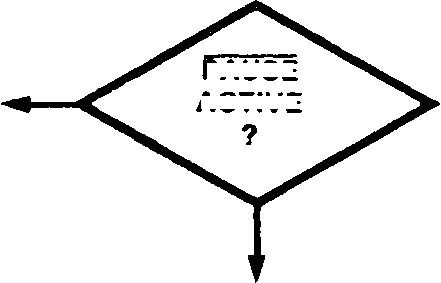
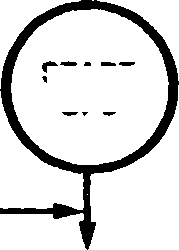
Extended Processing Units connect directly to the Z-BUS and continuously monitor the instruction stream. When the template portion of an extended instruction is fetched from memory, the appropriate EPU will detect that the instruction is meant for it and respond to the instruction. The CPU is always responsible for fetching instructions and delivering operands to the EPUs. The EPUs recognize the extended instruction templates and execute them, using data supplied with the template and/or data already within internal EPU registers.

There are four types of extended instructions in the Z280 instruction set: data transfers from memory to an EPU, data transfers from an EPU to memory, data transfers from an EPU to the CPU’saccumulator register, and EPU internal operations. Twenty-two instruction opcodes are used to implement these operations. Each extended instruction opcode includes two parts: a two- or four-byte instruction opcode used by the Z280 CPU to determine its activity and the address of the memory operand, and a four-byte instruction ’’template” that specifies the EPU activity. Six operand addressing modes are supported by the instructions that specify transfers between EPU registers and main memory: Direct Address, Indirect Register, Indexed, Stack Pointer Relative, Program Counter Relative, and Base Index. (See section 5.4.10 for a description of the extended instructions.)

In addition to the hardware-implemented capabilities of the EPA, there is an extended instruction trap mechanism that permits software simulation of EPU functions. The state of the EPU Enable bit in the CPU’s Trap Control register indicates whether EPUs are present in the system (see section 3.3.5). If the EPU Enable bit is cleared to 0, indicating that there are not EPUs in the system, the CPU will execute an Extended Instruction trap if an extended instruction is encountered in the instruction stream. The

service routine for this trap could perform a software simulation of an EPU’s functions. This trap mechanism facilitates the design of systems in which EPUs are not present but may be added later. Initially, the ’’extended” function is executed as the Extended Instruction trap service routine; when EPUs are added to the system, the trap routine is eliminated and the EPU Enable bit is set to 1. This change would be transparent to applications programs. (The Extended Instruction trap is described in section 6.3.1.)

* + 1. Extended Instruction Execution Sequence

The CPU and EPU instruction execution sequence is diagrammed in figure 10-8. When the CPU fetches an extended instruction, the EPU Enable bit in the Trap Control register is examined. If the EPU Enable bit is a 0, an Extended Instruction trap is executed. If the EPU Enable bit is a 1, indicating that there is an EPU in the system, then the CPU fetches the four-byte instruction template from memory. The fetching of the template is indicated by the STj-STq status lines from the CPU. EPUs must continuously monitor the address/data bus and STj-STq status lines for its templates. A 2-bit identification field in the template can select one of up to four EPUs for execution of a given extended instruction. If the extended instruction calls for the transfer of data between the CPU and EPU or between the EPU and memory, the CPU generates the appropriate bus transaction cycles. These transactions are identified by unique encodings of the STj-STq status lines. The EPU monitors the status and

PAUSE

YES

NO

CPU IDLES IN PAUSE STATE

START CPU

ACTIVE

YES

YES

NO

NO

CPU EXECUTES INSTRUCTION

FETCH NEXT INSTRUCTION

~~———~~

EPA TRAP SERVICE ROUTINE

EPU

PRESENT

BIT SET

EPU

INSTRUCTION

CPU FETCHES  
EPU TEMPLATE  
FROM MEMORY

CPU GENERATES  
DATA/ADDRESS  
AND PLACES  
ON Z-BUS

MONITORS Z-BUS FOR EPU TEMPLATE

SET PAUSE LINE AT CPU UNTIL EPU FREE

Figure 10-8. CPU-EPU Instruction Execution Sequence

timing signals output by the CPU to determine when to participate in the data transaction; the EPU supplies or captures the data when DS is active. For transactions between an EPU and memory, the CPU 3-states its address/data lines while DS is active so that the EPU or memory can supply the data. (See section 13.5.5 for a description of the bus transaction timing.)

The number and type of bus cycles required to fetch the extended instruction template depends on whether the template is aligned on an even address boundary. The four-byte long template can be fetched with two word transactions if the template begins on an even memory address or with one byte and two word transactions if the template begins at an odd memory address, as described in Table 10-1. (In the case of an odd starting address for the template, the EPU captures only the upper byte from the bus during the second word transaction.) The template is always fetched from memory using the CPU’s external bus interface, regardless of the current state of the on-chip cache memory.

Table 10-1. Bus Transactions Involved in Fetch of Extended Instruction Template

**Address at Address >**

**Template Start Bus Cycle from Z280 Byte/Word ST3-ST0**

Even 1 **П** Word 1101

- ■ - \* 2 ' пч-2 Word 1100

Odd 1 **n** Byte 1101

2 n+1 Word 1100

3 n+3 Word 1100

If the extended instruction specifies an internal EPU operation, the Z280 CPU can proceed to fetch and execute subsequent instructions. Thus, the CPU and EPUs may be processing in parallel. The PAUSE signal is used to synchronize CPU-EPU activity in the case of overlapping extended instructions. If the CPU fetches another extended instruction template intended for an EPU that is still executing a previous instruction, the EPU activates the PAUSE input to the CPU to halt further CPU activity until the EPU can finish the original operation. While PAUSE is asserted, all CPU activity is suspended except responses to ? refresh requests, bus requests, and resets.

CPU activity following the fetch of the extended instruction template is governed by the type of extended instruction being processed. In the case of an EPU internal operation, no further bus transactions are required by the extended instruction, so the CPU will proceed to fetch the next instruction. However, the CPU will still honor an active PAUSE input and suspend execution until PAUSE is released.

In the case of an EPU-to-CPU transfer instruction, the next non-refresh transaction following the fetch of the template (and after an active PAUSE signal is deasserted) will be the EPU-to-CPU bus transaction. EPU-to-CPU bus transactions are

identified by a 1110 status code on the ST^-STq status lines and are word transactions. The address emitted by the CPU during this cycle is the memory address of the previous transaction (that is, the address used during the last fetch of the instruction template).

In the case of EPU-to-memory or memory-to-EPU transfer instructions, the next one to sixteen non-refresh transactions following the fetch of the template (and after an active PAUSE signal is deasserted) will be the appropriate data transfer cycles. Up to 16 bytes of data may be transferred as the result of a single extended instruction; the number of data transfers to be performed is encoded in the instruction template. The 1010 status code on the STj-STq status lines identifies bus cycles that transfer data between an EPU and memory. The EPU must supply the data for write operations or capture the data for read operations during each transaction, just as if it were part of the CPU. The number and type of transactions generated also depends on whether the starting memory address of the data block to be moved is an even-valued address, as defined in Table 10-2. The case where only one byte is transferred is degenerate and shown separately in Table 10-2 for clarity. These transfers are always performed on the Z280 MPU’s external bus, regardless of the current state of the on-chip cache memory.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table 10-2. Sequence of Transactions for Data Transfers between an EPU and Memory** | | | | |
| **Starting Memory Address** | **Number of Bytes (n)** | **Byte/Word Status of Transfers** | **Type of Addresses** | **Total Number of Transactions** |
| Even | Even | word, word word | All even | n/2 |
| Even | Odd | word, word word, byte | All even | (n+1)/2 |
| Even | One | byte | Even | **1** |
| Odd | Even | byte, word word, byte | First odd, others even | (n *+2)12* |
| Odd | Odd | byte, word,....word, word | First odd, others even | (n +1)/2 |
| Odd | One | byte | Odd | 1 |

**Reset**

**■•■■■■■■■■имивявиаииииявиииииивииииииииииквяшиваипаи**

В таблице 11-1 показано влияние сброса на

другие регистры процессора. Сброс переводит процессор в

Аппаратный сброс осуществляется активным входом сброса и переводит MPU Z280 в известное состояние. При необходимости регистр синхронизации шины и инициализации может быть инициализирован до заданного системой значения во время сброса. Входной сигнал СБРОСА внутренне синхронизируется с тактовой частотой и затем отбирается в конце каждого тактового цикла процессора. При обнаружении активной линии сброса разрешается завершить текущую транзакцию шины перед запуском процесса сброса. Сброс переопределяет все остальные операции, включая прерывания, ловушки и запросы шины. Для инициализации MPU Z280 как части последовательности включения необходимо выполнить аппаратный сброс.

Ввод сброса должен быть запрограммирован минимум на 128 тактовых циклов процессора. В течение этого времени линии MPU Z280 принимают свои значения сброса: в строках адреса и адреса/данных указано значение 3, а все управляющие линии имеют принудительное высокое значение. Пока выполняется сброс, тактовая выходная строка (CLK) представляет собой тактовую частоту процессора, деленную на четыре.

Когда выборка RESET высока (деассертирована), производится выборка состояния входного сигнала ОЖИДАНИЯ. Если заявлено ОЖИДАНИЕ, содержимое строк ADg-ADy дискретизируется по падающему фронту тактовой частоты процессора и загружается в регистр синхронизации и инициализации шины; если выбран этот метод инициализации, ADy должно быть равно 1, а AD^ должно быть равно 0 при дискретизации шины, и состояние строки AD^ определяет, выбран ли параметр режима начальной загрузки. ОЖИДАНИЕ должно быть заявлено как минимум на два такта процессора после отмены сброса, чтобы регистр синхронизации шины и инициализации регистрировался, тем самым задавая тактовую частоту шины, равную половине тактовой частоты процессора, никаких автоматических состояний ожидания при доступе к меньшим 8 мбайт памяти и отключая многопроцессорный режим работы..

‘режим прерывания 0; таким образом, поле IM в регистре состояния прерывания будет равно 0. Биты включения вектора прерывания в регистре состояния прерывания также сбрасываются до 0 путем сброса, а биты ожидания прерывания будут отражать текущее состояние запросов на прерывание. Сброс не влияет на все остальные регистры CPU и MMU, включая оставшиеся регистры в файле регистров CPU, регистры дескриптора страницы MMU и указатель таблицы векторов прерываний/ловушек.

Влияние сброса на программируемые регистры встроенных периферийных устройств показано в таблице 11-2. Встроенные счетчики/таймеры всегда отключаются при сбросе. Встроенные каналы DMA и UART также отключаются при сбросе, если не выбран режим начальной загрузки (см. раздел 9.7). Сброс не влияет на постоянную времени счетчика/таймеров и регистры времени отсчета. Сброс также не влияет на адрес назначения каналов DMA, адрес источника и регистры подсчета, за исключением адреса назначения канала DMA0 и регистров подсчета.

В многопроцессорной системе, использующей несколько Mpu Z280 с общей шиной, необходимо синхронизировать внутренние тактовые частоты процессора для MPU Z280. Тактовые частоты процессора генерируются путем деления входных данных XTALI на два. Нисходящий фронт сброса используется внутри для синхронизации тактовых импульсов процессора и может использоваться для синхронизации тактовых импульсов процессора в многопроцессорной системе. Если все Mpu Z280 в системе имеют идентичные входные сигналы XTAL1 и RESET, их внутренние тактовые частоты процессора будут инициализированы одинаковым образом путем сброса.

Если на переднем крае сброса обнаружен активный запрос шины, MPU Z280 предоставляет доступ к шине до получения первой команды из местоположения 0. Таким образом, внешнее устройство DMA может инициализировать оперативную память перед началом выполнения. Если запрос шины не подтвержден, центральный процессор начинает выполнение с выборки из местоположения 0, если только не действует режим начальной загрузки.

Value Loaded on Reset Register (Hexadecimal) Comments

Program Counter 0000

System Stack Pointer 0000

I 00

R ’ 00

Master Status 0000

Bus Timing and Control 30

Bus Timing and Initialization 80

I/O Page 00

Cache Control 20

Trap Control 00

System Stack Limit 0000

Local Address 00

Interrupt Status OOxx

Interrupt/Trap Vector Table Pointer

CPU Registers AF, BC, DE, HL, IX, IY,

AF', BC', DE', HL'

User Stack Pointer

MMU Master Control 0000

MMU Page Descriptor Register, Page

Descriptor Register Pointer

System mode, Single-Step disabled, Breakpoint-on-Halt disabled

All maskable interrupts disabled

No automatic wait states for I/O, upper 8M bytes of memory, or interrupt acknowledges

CLK output 2 x processor clock period, no automatic wait states for lower 8M bytes of memory, bootstrap mode disabled

I/O Page 0 in use

Cache enabled for instructions

All valid bits cleared to 0

Burst mode disabled

EPA trap disabled, I/O not privileged

System Stack Overflow Warning trap disabled

All memory transactions are made to local bus

Interrupt mode 0, nonvectored interrupts, current state of interrupt requests (indicated by xx)

Unaffected

Unaffected • . - . . •• . ■

U naffected

MMU disabled

U naffected .

Register

Value Loaded on Reset  
(Hexadecimal)

Comments

Refresh

Refresh enabled, rate = 32

Counter/Timers: Configuration Command/Status

00

00

Timer mode, single-cycle mode Timer disabled

DMA Channels:

Master Control

DMA0 Transaction Descriptor

DMA1/2/3 Transaction Descriptor

0000

0100

No DMA linking, EOP disabled, Software Ready disabled DMA0 disabled, continuous mode

EN, IE, TC, and EPS fields cleared, other fields unaffected

DMA0 Destination Address

DMA0 Count

000000

0100

UART:

Configuration

00

Transmitter Control/Status

Receiver Control/Status

01

00

5 bits/character, parity disabled, external clock, x 1 clock rate, loop back disabled

. , Transmitter disabled, transmit buffer empty

Receiver disabled

Unless bootstrap mode is selected.

**Z280 Bus External Interface ■■■■■■■■■■шммимшмшаЕЯнмшанкаБгявяаЕимптжямшавпвкя**

12.1 INTRODUCTION

MPU Z280 обычно является только одним компонентом в системе, который может включать в себя память, периферийные устройства, подчиненные процессоры, сопроцессоры и другие процессоры, подключенные через системную шину. С MPU Z280 доступны две различные схемы шинного соединения компонентов: шина Z80 и Z-BUS.

В этой главе описываются внешние проявления (то есть активность на выводах), возникающие в результате активности центрального процессора или встроенной периферии для конфигураций шины Z80 MPU Z280. (Внешний интерфейс Z-BUS описан в главе 13.) Поскольку контакты подключены к системной шине, большая часть этого обсуждения будет сосредоточена на

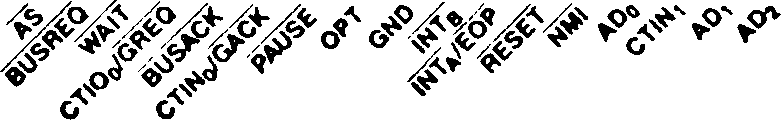
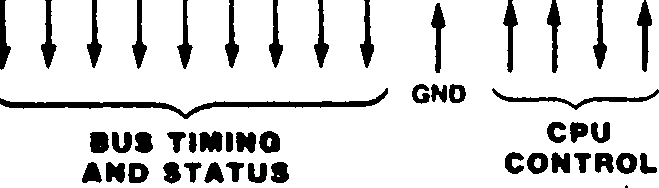
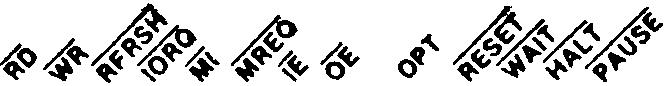
Состояние сигнального вывода ORG определяет конфигурацию интерфейса шины для MPU Z280; конфигурация шины Z80 выбирается путем применения логического значения 0 (заземления) на выводе OPT. A

> Шина Z80 в MPU Z280 включает в себя 24-разрядную адресную шину, 8-разрядную шину данных и связанные с ней сигналы состояния и управления. Шина данных мультиплексируется с 8 младшими разрядами адресной шины. На рисунке 12-1a показаны функции вывода для конфигурации шины Z80 MPU Z280. Описанная здесь шина Z80 совместима с периферийными устройствами семейств Z8400 и Z8500 от Zilog.

шина и операции с шиной.

ADDRESS!

DATA



BUS CONTROL

BUSREQ

BUSACK

INTERRUPTS

**inta/eop**

INTb INTc

Au

RxD

zsso  
MPU

\*15

Ait

TxD

ON-CHIP PERIPHERALS

**ctin/gaCk •**

Aie

CTIO/GREQ \*

ROY

4

DMASTB

A 23

| HALT | 10 | 60 |
| --- | --- | --- |
| DMASTBo | 11 | 59 |
| WR | 12 | 58 |
| DMASTB 1 | 13 | 57 |
| RFRSH | 14 | 56 |
| iORQ | 15 | 55 |
| ОЁ | 16 | 54 |
| IE | 17 | 53 |
| **ADDRESS + 5V** | 18 | **Z280 52**  **MPU** |
| + 5V | 19 | 51 |
| CTIOt | 20 | 50 |
| Mi | 21 | 49 |
| MREO | 22 | 48 |
| CTIOj | 23 | 47 |
| RD | 24 | 46 |
| CTINj | 25 | 45 |
| Г INTc | 28 | 44 |

9 8 7 6 5 4 3 2 1 60 67 66 65 64 63 62 61

ADe

Ajt

RDY3

AD5 RDYi

ROYq AO4 GND RESERVED GND XTALI XTALO RxD CLK IXD

Am АОэ

•MuHlpbxMl with CT1NO

•‘Mult Iptlxed with CTIOq

Figure 12-1a. Pin Functions

27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43

Figure 12-1b. Pin Assignments

Figure 12-1. Z80 Bus Configuration (Input OPT tied to GND)

Interrupt request.

A request initiated by a

На шине Z80 могут выполняться два вида операций: транзакции и запросы. В любой момент времени только одно устройство (либо центральный процессор, либо устройство, запрашивающее шину, такое как канал DMA) может управлять шиной; это устройство называется ведущим устройством шины. Транзакции всегда инициируются ведущим устройством шины и на них отвечает какое-либо другое устройство на шине. Одновременно может выполняться только одна транзакция. Запросы могут инициироваться устройством, которое не контролирует шину.

На шине Z80 могут выполняться семь типов транзакций, как описано ниже:

Транзакция с памятью. Управляемая процессором или DMA передача данных в ячейку памяти или из нее.

Транзакция RETI. Транзакция, инициируемая центральным процессором, используемая в сочетании с логикой прерывания периферийных устройств семейства Z8400.

Транзакция остановки. Транзакция, указывающая, что центральный процессор переходит в состояние остановки из-за выполнения команды остановки или фатальной последовательности перехватов.

Освежить. Транзакция, которая обновляет динамическую память; транзакции обновления не связаны с передачей данных.

Транзакция ввода-вывода. Управляемая процессором или DMA передача данных на периферийное устройство или с него.

Подтверждение прерывания. Транзакция, управляемая процессором, используется для подтверждения прерывания и считывания данных с прерывающего устройства.

Транзакция обхода DMA. Транзакция, управляемая DMA, которая передает данные между ячейкой памяти и периферийным устройством.

На шине Z80 могут выполняться запросы двух типов, как описано ниже:

периферийное устройство для привлечения внимания центрального процессора.

**Запрос шины. Запрос внешнего устройства (обычно канала DMA) на получение контроля над шиной для инициирования транзакций.**

**Центральный процессор отвечает на запрос в соответствии с его типом: для запросов на прерывание генерируется последовательность подтверждения прерывания; для запросов на шину центральный процессор отключает шину и активирует сигнал подтверждения.**

**Функции выводов для конфигурации шины Z80 MPU Z280 проиллюстрированы на рисунке 12-1a. Назначение выводов показано на рисунке 12-1b. Функциональное описание каждого вывода приведено ниже:**

**A8-A23. Адрес (выходной, активный максимум, 3 состояния). Эти адресные строки содержат адреса ввода-вывода и адреса памяти во время транзакций по шине.**

**AD0-AD7. Адрес/данные (двунаправленный, активный максимум, 3 состояния). Эти восемь мультиплексированных строк данных и адреса передают адреса ввода-вывода, адреса памяти и данные во время транзакций по шине.**

**Стробоскоп AS. Address (выход, активный низкий уровень, 3-состояние). Возрастающий фронт AS указывает на начало транзакции и показывает, что адрес действителен.**

**BUSACK. Подтверждение шины (вывод, активный низкий уровень). Низкий уровень в этой строке указывает на то, что центральный процессор отказался от управления шиной в ответ на запрос шины.**

**BUSREQ. Запрос шины (ввод, активный низкий уровень). Низкий уровень в этой строке указывает на то, что внешнее устройство, запрашивающее шину, получило или пытается получить управление шиной.**

**CLK. Тактовый выход (output). Частота тактовой частоты процессора определяется по входу генератора (внешний генератор) или частоте кристалла (внутренний генератор) путем деления входного сигнала кристалла или внешнего генератора на два. Тактовая частота процессора далее делится на единицу, две или четыре (как запрограммировано) и затем выводится в этой строке.**

**CTIN. Вход счетчика/таймера (input, активный максимум). Эти строки принимают сигналы от внешних устройств для счетчика/таймеров.**

**CTIO. Ввод/вывод счетчика/таймера (двунаправленный, активный максимум, 3 состояния). Эти линии ввода/вывода передают сигналы между счетчиком/таймерами и внешними устройствами.**

**DMASTB. Стробоскоп DMA Flyby (выходной, активный минимум). Эти строки выбирают периферийные устройства для промежуточных передач.**

**EOP. Завершение процесса (вход, активный низкий уровень). Внешний источник может прервать выполняемую операцию DMA, понизив EOP. EOP всегда применяется к активному каналу; если ни один канал не активен, EOP игнорируется. ■**

**GACK. Глобальное подтверждение (ввод, активный низкий уровень). Низкий уровень в этой строке указывает, что процессору предоставлено управление глобальной шиной.**

**GREQ. Глобальный запрос (вывод, активный низкий уровень, 3-состояние). Низкий уровень в этой строке указывает, что центральный процессор получил или пытается получить управление глобальной шиной.**

**GND. Ground. Ссылка на землю.**

**остановка. Остановка (выход, активный низкий уровень, 3-состояние). Этот сигнал указывает на то, что центральный процессор находится в состоянии остановки и ожидает прерывания, прежде чем сможет возобновить работу**.

**IE.** *Input Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is toward the MPU.

**INT.** *Maskable Interrupts* (input, active Low). A Low on these lines requests an interrupt. .

**IORQ.** *Input/Output Request* (output, active Low, 3-state). This signal indicates that ADg-ADy and А^-Арз of the address bus hold a valid I/O address for an I/O read or write operation. An IORQ signal is also generated with an M1 signal when an interrupt is being acknowledged, to indicate that an interrupt response vector can be placed on the data bus.

**M1.** *Machine Cycle One* (output, active Low, 3-state). This signal indicates that the current transaction is the opcode fetch cycle of a RETI instruction execution. M1 also occurs with IORQ to indicate an interrupt acknowledge cycle.

**MREQ.** *Memory Request* (output, active Low, 3-state). This signal indicates that the address bus holds a valid address for a memory read or write operation.

**NMI.** *Nonmaskable Interrupt* (input, falling-edge activated). A High-to-Low transition on this line requests a nonmaskable interrupt.

**OE.** *Output Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is away from the MPU.

**OPT.** *Bus Option* (input). This signal establishes the bus option during reset.

OPT Bus Interface

0 Z80 Bus, 8-bit

**1** Z-BUS, 16-bit

**PAUSE.** *MPU Pause* (input, active Low). While this line is Low the MPU refrains from transferring data to or from an Extended Processing Unit in the system or from beginning the execution of an instruction.

**RD.** *Read* (output, active Low, 3-state). This signal indicates that the CPU or DMA peripheral is reading data from memory or an I/O device.

**RDY.** *DMA Ready* (input, active Low). These lines are monitored by the DMAs to determine when a peripheral device associated with a DMA port is ready for a read or write operation. When a DMA port is enabled to operate, its Ready line indirectly controls DMA activity; the manner in which DMA activity is controlled by the line varies with the operating mode (single-transaction, burst, or continuous).

**RESET.** *Reset* (input, active Low). A Low on this line resets the CPU and on-chip peripherals.

**RFSH.** *Refresh* (output, active Low, 3-state). This signal indicates that the lower ten bits of the Address bus contain a refresh address for dynamic memories and the current MREQ signal should be used to perform a refresh to all dynamic memories.

**RxD.** *UART Receive* (input, active High). This line receives serial data at standard TTL levels.

**TxD.** *UART Transmit* (output, active High). This line transmits serial data at standard TTL levels.

**WAIT.** *Wait* (input, active Low). A Low on this line indicates that the responding device needs more time to complete a transaction.

**WR.** *Write* (output, active Low, 3-state). This signal indicates that the bus holds valid data to be stored at the addressed memory or I/O location.

**XTALI.** *Clock/Crystal Input* (time-base input). Connects a parallel-resonant crystal or an external single-phase clock to the on-chip oscillator.

**XTALO.** *Crystal Output* (time-base output). Connects a parallel-resonant crystal to the on-chip oscillator.

**+ 5V.** *Power Supply Voltage. (*+ 5 nominal).

Z280 MPU’s external bus timing: the Bus

register, Bus Timing and  
Address register, and

Четыре регистра управления процессором Z280 задают определенные характеристики интерфейса и определяют регистр управления ограничением и инициализацией, регистр управления локальным кэшем.

Синхронизация шины определяется частотой внешнего источника тактовой частоты или кристалла Z280 MPU и содержимым регистра синхронизации и инициализации шины, который получает свои начальные значения как часть процесса сброса (см. раздел 3.2.1). Частота тактовой частоты процессора составляет половину частоты внешнего источника тактовой частоты или кристалла. Тактовая частота процессора может быть дополнительно разделена на коэффициент 1, 2 или 4, чтобы обеспечить синхронизацию шины, как указано содержимым поля масштабирования тактовой частоты в регистре синхронизации шины и инициализации. Тактовый сигнал шины выводится MPU в виде сигнала CLK. На логических диаграммах синхронизации, которые приведены ниже, переходы сигналов на шине показаны в зависимости от тактового сигнала шины, CLK.

Количество автоматических состояний ожидания, включенных в данную транзакцию, определяется содержимым регистров синхронизации и инициализации шины и регистров синхронизации и управления шиной. Адресное пространство физической памяти разделено на две секции на основе наиболее значимого бита физического адреса, A23. До трех автоматических состояний ожидания могут быть добавлены к транзакциям в нижней половине памяти (адреса, где A23 = 0); аналогично, до трех автоматических состояний ожидания могут быть добавлены к транзакциям в верхней половине памяти (A23 = 1), ко всем транзакциям ввода-вывода и для прерывания подтверждать транзакции.

Состояние бита включения многопроцессорной конфигурации в регистре синхронизации и инициализации шины и содержимое регистра локального адреса определяют, какие транзакции с памятью требуют использования глобальной шины, как описано в разделе 10.3. Содержимое регистра управления кэшем и состояние адресных меток и допустимых битов в кэш-памяти определяют, какие транзакции используют кэш-память, а какие транзакции используют интерфейс внешней шины, как описано в главе 8.

В любой момент времени одно устройство (либо центральный процессор, либо устройство, запрашивающее шину) управляет шиной и называется ведущим устройством шины. Транзакция инициируется ведущим устройством шины и на нее отвечает какое-либо другое устройство на шине. Передача информации (как инструкций, так и данных) в MPU Z280 и обратно осуществляется с помощью транзакций. Все транзакции начинаются, когда адресный стробоскоп (AS) понижается, а затем повышается.

если для транзакции требуется адрес, адрес действителен на переднем крае AS. AS может использоваться для привязки адресов MPU Z280 для демультиплексирования строк адреса/данных Z280. Если адрес сгенерирован, строка Output Enable (OE) активируется одновременно с утверждением AS.

Строки чтения (RD) и записи (WR) используются для определения времени передачи данных. Для транзакций, которые не связаны с передачей данных (обновление и остановка транзакций), ни RD, ни WR не активируются. Для операций записи низкое значение WR указывает на то, что в строках AD находятся действительные данные от ведущего устройства шины. Строка разрешения вывода продолжает утверждаться до тех пор, пока WR не будет отменен. Для операций чтения мастер шины понижает уровень линии RD, когда адресуемое устройство должно передать свои данные на шину. Совпадая с утверждением RD, мастер шины указывает 3 строки AD, и OE отключается.; Разрешение ввода (IE) утверждается на половину тактового цикла позже. Мастер шины выполняет выборку данных по падающему фронту тактовой частоты непосредственно перед снятием RD и IE. Возрастающий фронт RD или WR отмечает завершение транзакции.

Вход ожидания MPU Z280 обеспечивает механизм, с помощью которого время выполнения конкретной транзакции может быть увеличено для размещения памяти или периферийного устройства с длительным временем доступа. Строка ожидания отбирается на падающем тактовом фронте, когда данные должны быть отобраны (т.е. непосредственно перед повышением RD или WR) во время транзакции. Если очередь ожидания низкая, к транзакции добавляется еще один тактовый цикл шины перед выборкой данных (повышается RD или WR). В этом добавленном цикле и во всех последующих циклах, добавленных из-за низкого уровня ожидания, строка ожидания отбирается по падающему фронту тактовой частоты, и, если она низкая, к транзакции добавляется еще один цикл перед выборкой данных. Таким образом, транзакция может быть расширена внешней логикой до произвольной длины с шагом в один такт шины.

Вход WAIT является синхронным и должен соответствовать указанному времени настройки и удержания для правильной работы MPU Z280. I для этого требуется асинхронность!u сгенерированные сигналы ожидания должны быть синхронизированы с выходом CLK, прежде чем они будут введены в MPU Z280. Автоматические состояния ожидания также могут быть сгенерированы путем программирования регистра синхронизации шины и управления, а также регистра синхронизации шины и инициализации; они вставляются в транзакцию перед выборкой внешнего сигнала ожидания.

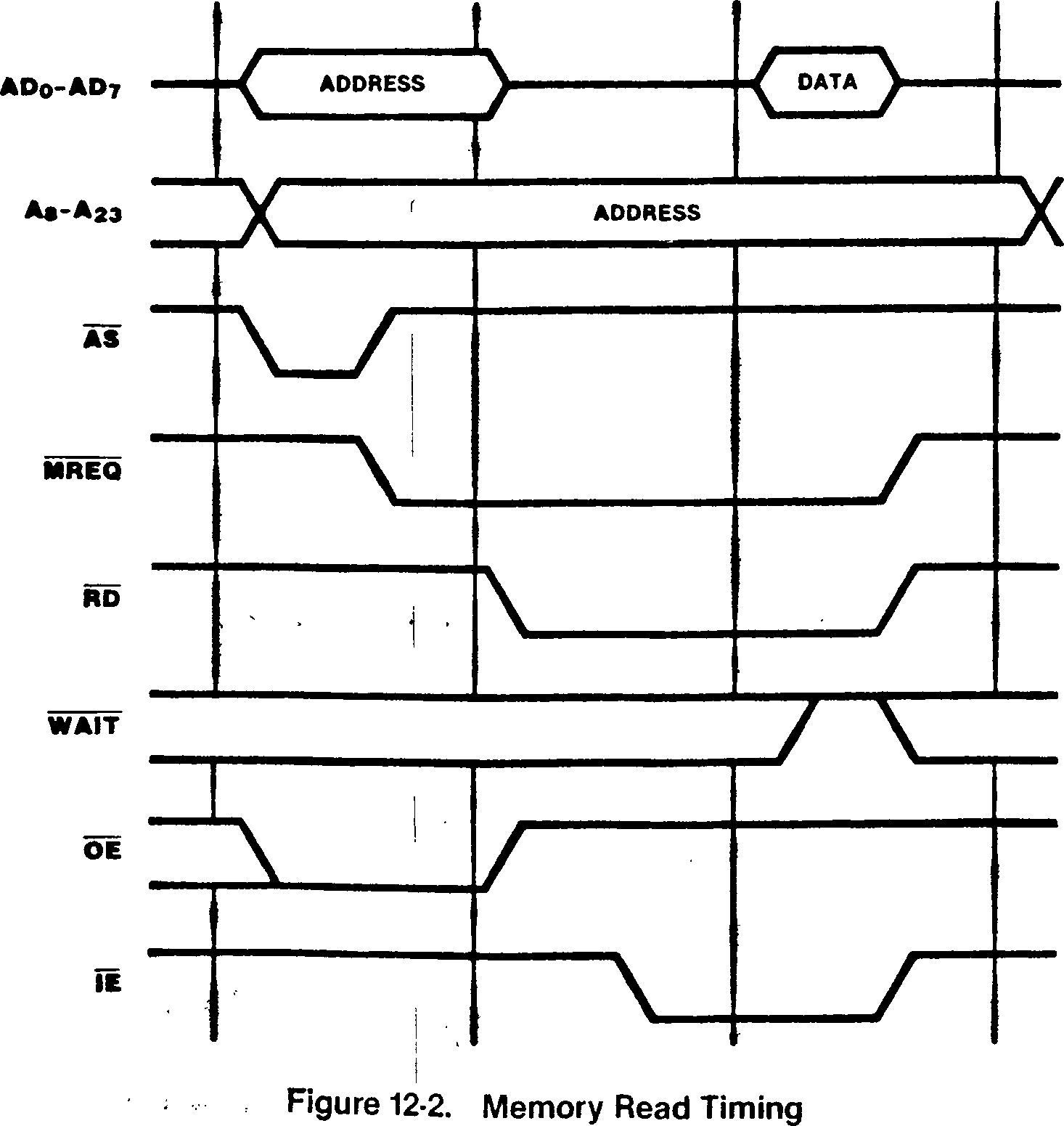
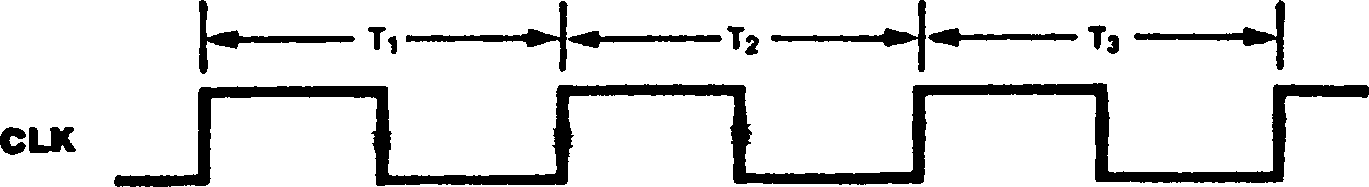
* + 1. Memory Transactions

Транзакции памяти перемещают инструкции или данные в память или из памяти, когда мастер шины осуществляет доступ к памяти. Таким образом, они генерируются во время выполнения программы для извлечения инструкций из памяти, а также для извлечения и сохранения данных памяти. Они также генерируются для сохранения старого состояния программы и получения нового состояния программы во время обработки прерываний и перехватов, а также для передачи информации во время обращений к памяти, контролируемых DMA. Транзакция с памятью длится три цикла шины, если не расширена аппаратными и/или программными состояниями ожидания, как объяснялось ранее.

Синхронизация транзакций с памятью показана на рисунках 12-2 и 12-3. Во время первого цикла шины,

Как утверждается, указывает на начало транзакции; в это время также утверждается разрешение вывода (OE). Сигнал MREQ становится активным во второй половине этого цикла шины, что указывает на транзакцию с памятью. Для операции считывания (рис. 12-2) RD активируется в течение первой половины второго цикла шины, после того как мастер шины установил 3 строки AD; OE отключается в начале второго цикла, а разрешение ввода (IE) устанавливается во второй половине второго цикла. Мастер шины выполняет выборку информации, возвращаемой из памяти на шине адреса/данных, на падающем фронте тактовой частоты во время третьего цикла шины; после выборки данных RD, MREQ и IE отключаются. Для операции записи (рис. 12-3) строка WR утверждается во второй половине второго цикла, после того как мастер шины поместил данные, подлежащие записи, в строки AD, и OE остается активным на протяжении всей транзакции.

Входной сигнал ОЖИДАНИЯ также дискретизируется на падающем фронте тактовой частоты во время третьего тактового цикла; если ОЖИДАНИЕ низкое, добавляется еще один тактовый цикл шины перед выборкой данных. Состояния ожидания также могут быть добавлены путем программирования регистра синхронизации шины и инициализации и регистра синхронизации шины и управления. Например, на рисунках 12-4, 12-5 и 12-6 показаны операции с памятью с одним состоянием ожидания.



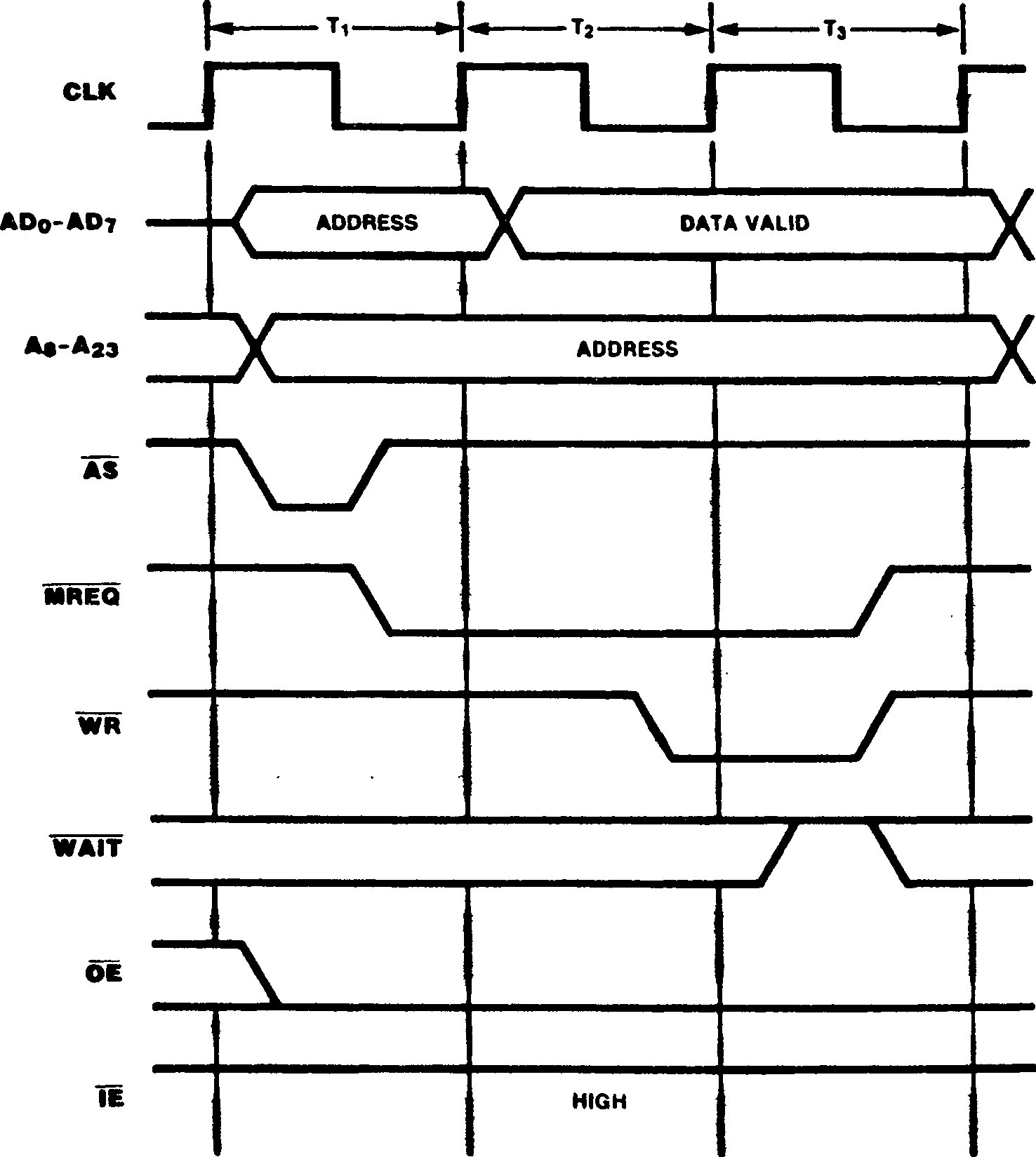
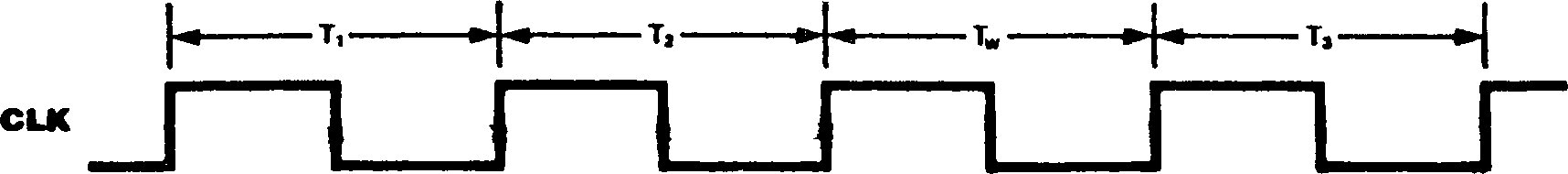


Figure 12-3. Memory Write Timing



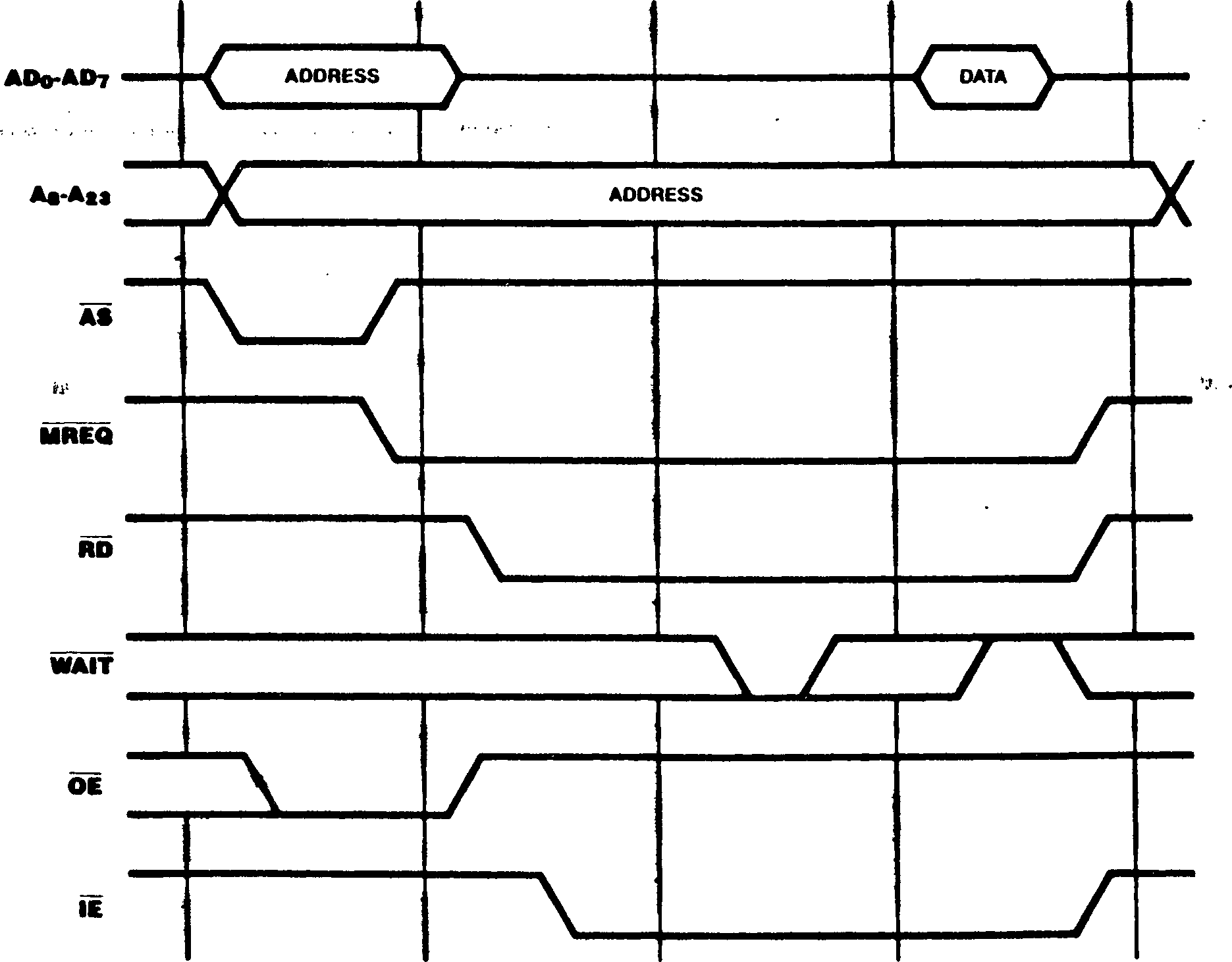
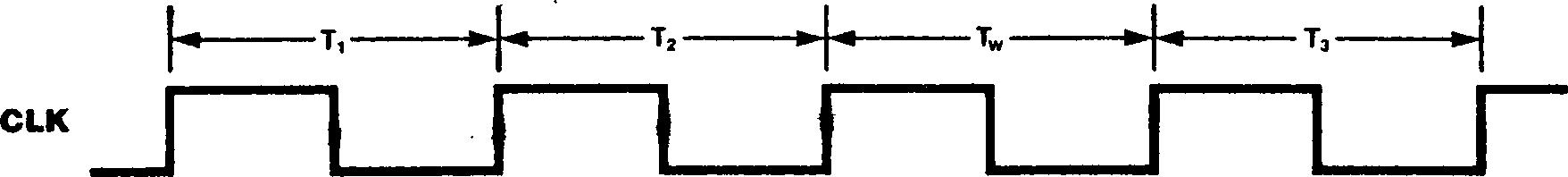


Figure 12-4. Memory Read Timing with One External Wait State

/



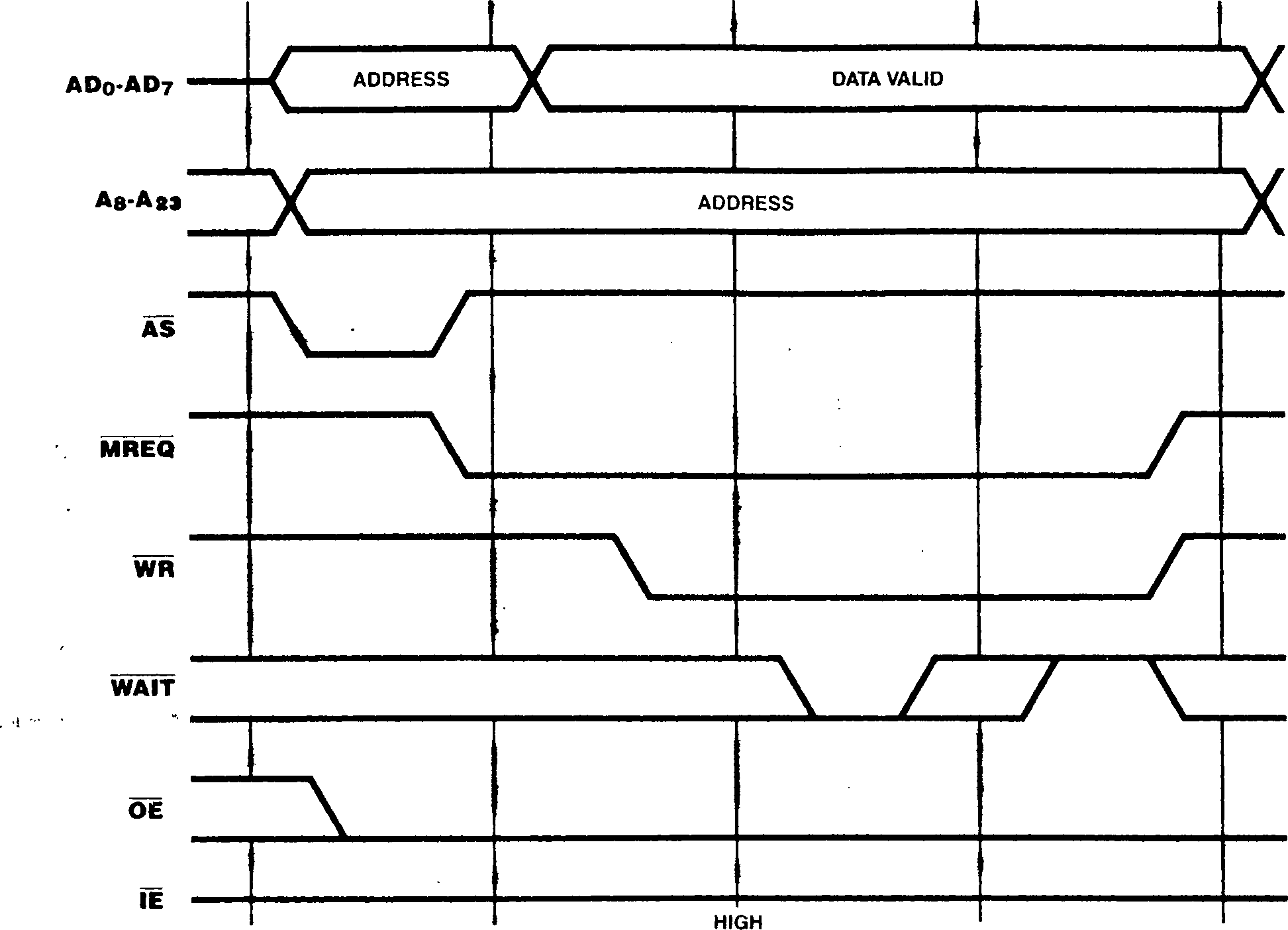
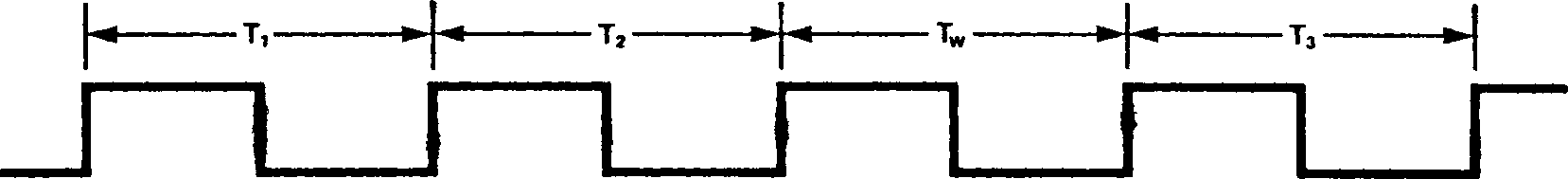


Figure 12-5. Memory Write Timing with One External Wait State



CLK

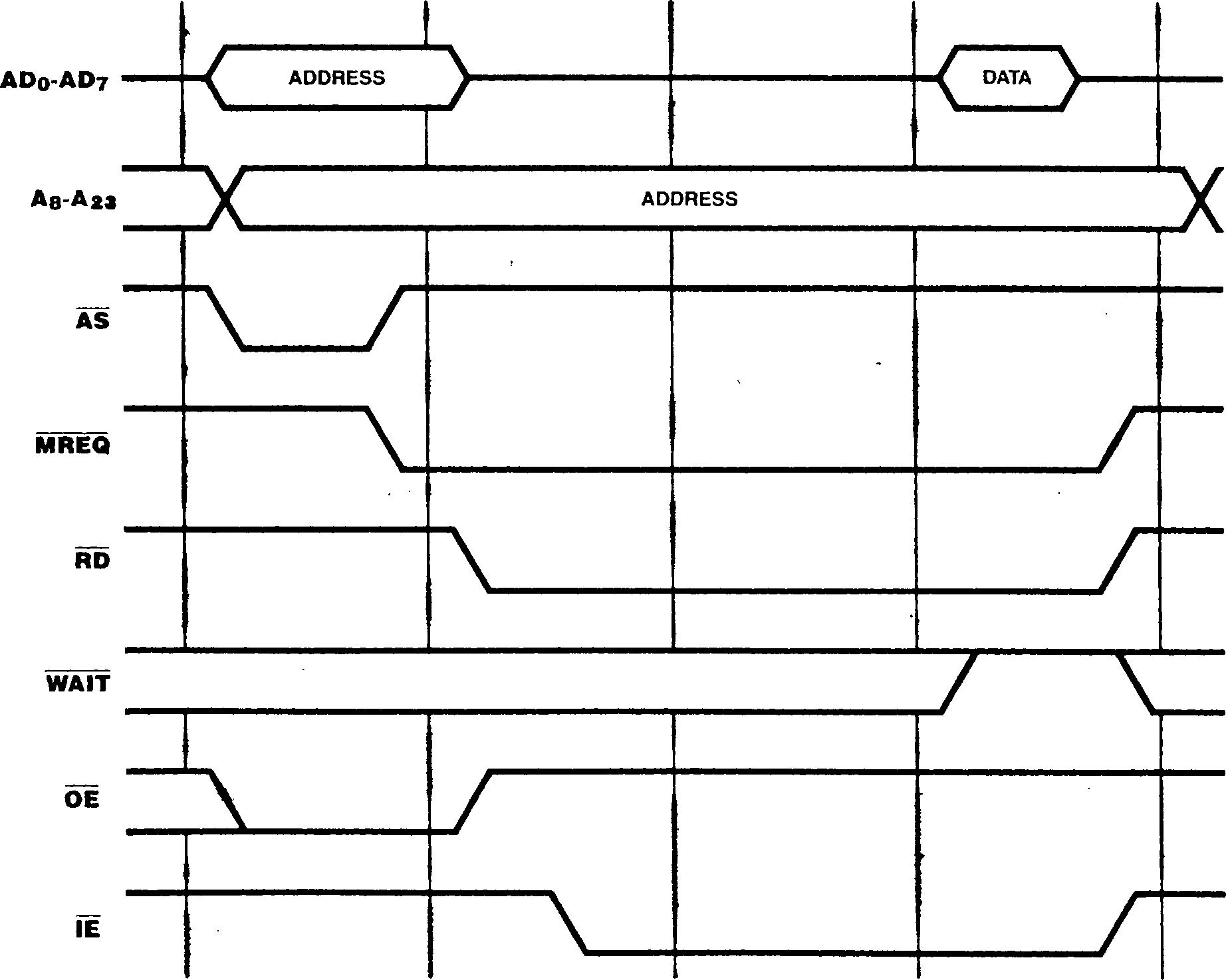
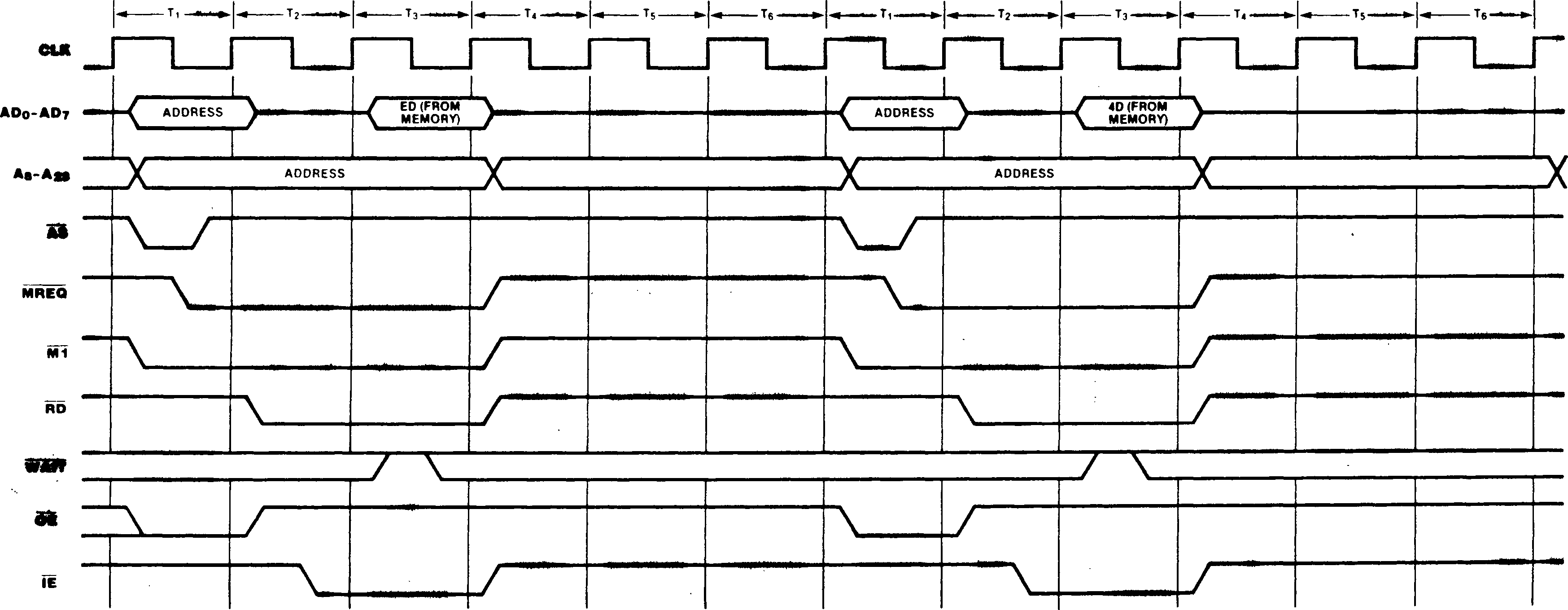


Figure 12-6. Memory Read Timing with One Internal Wait State

12-8

Figure 12-7. RETI Read Timing



* + 1. RETI Transactions

RETI transactions (Figure 12-7) are similar to memory read transactions with two exceptions: M1 is asserted throughout each read transaction, falling early in the first bus cycle, and MREQ, M1, RD, and IE are deasserted on the rising edge of the clock following the third cycle» Each of the read transactions is followed by a minimum of three bus cycles of inactivity. These trans­actions are invoked whenever an RE 11 instruction is encountered in the instruction stream; they are used to re-fetch the instruction from external memory so that interrupt logic within Z8400 family peripherals that monitor the bus for this instruction will function correctly."

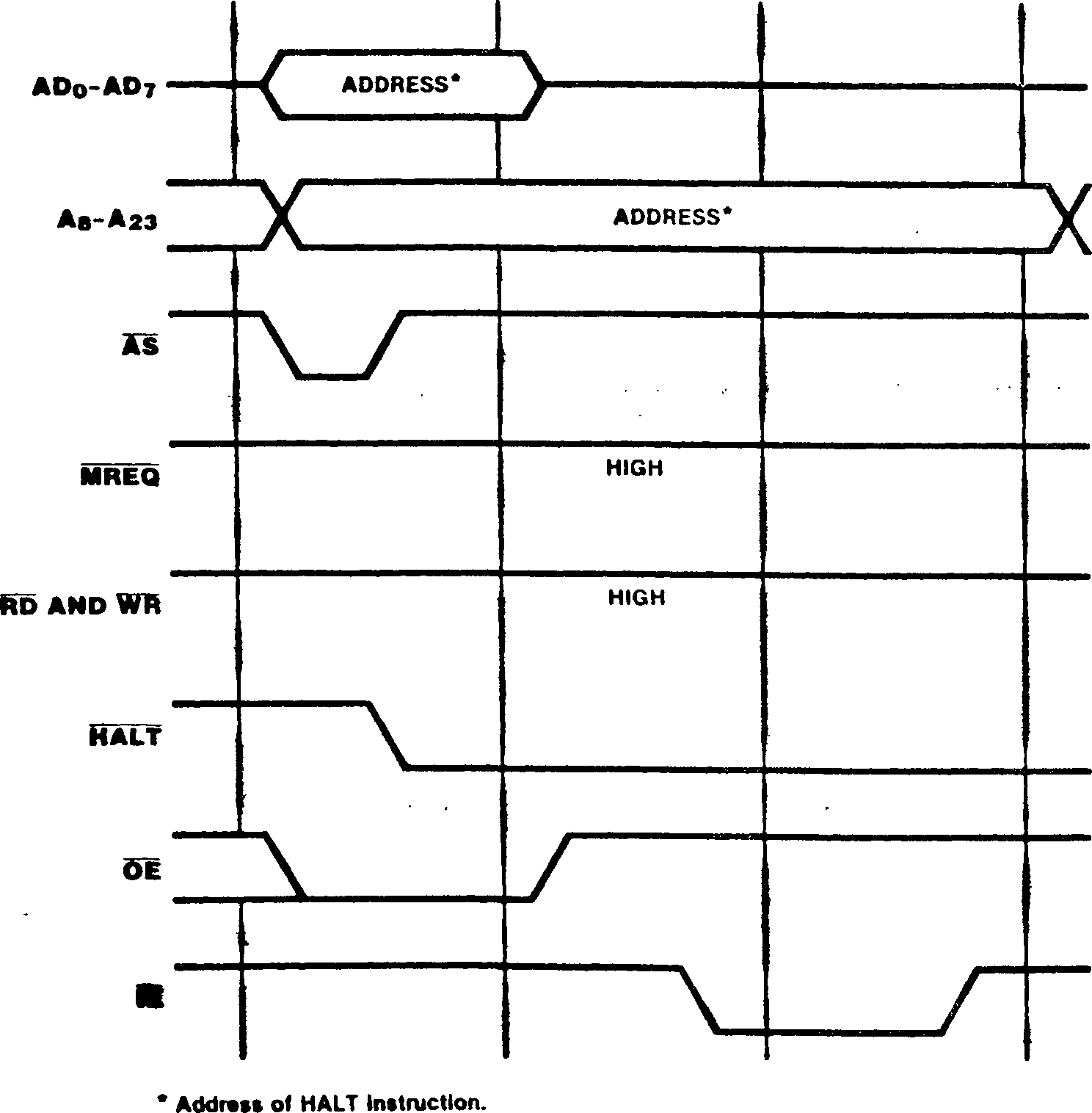
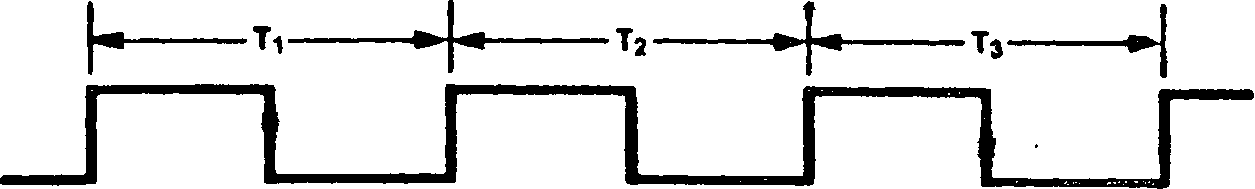
* + 1. Halt and Refresh Transactions

There are two types of bus transactions that do not transfer data: Halt and Refresh transactions. These transactions are similar to memory transactions, except that RD and WR remain high, the WAIT input is not sampled, and no data is Halt transactions (Figure 12-8) are identical to memory read transactions except that HALT is asserted throughout the transaction, falling during the second half of the first bus cycle, and remains asserted after the transaction is completed. This transaction is invoked when a HALT instruction is executed or a fatal seguence of traps occurs. Гог Halt transactions generated by the HALT instruction, once the Halt transaction is completed, all subseguent CPU activity is suspended until an active interrupt reguest or reset is detected. After Halt transactions generated due to a fatal condition, all CPU activity is suspended until an active reset is detected (see section 6.6). The HALT line remains asserted until the interrupt reguest is acknowledged or the reset is received. Refresh transactions or DMA transfers may occur while HALT is asserted; also, the bus can be granted. Ihe address put out during the address phase of the Halt transaction is the address of the Halt instruction or the instruction that initiated the fatal seguence of traps.

transferred.

cue

Figure 12-8. Halt Timing



*A* memory refresh transaction (Figure 12-9) is generated by the Z280 MPU refresh mechanism and can occur immediately after the final clock cycle of any other transaction. The memory refresh counter’s 1O-bit address is output on ADg-ADy, Ag, and A9 when AS is asserted; the remaining address lines are undefined. The RFSH line is activated concurrent with MREQ. This transaction can be used to generate refreshes for dynamic RAMs. Refreshes may occur while the CPU is in the Halt state.

* + 1. I/O Transactions \*

I/O transactions move data to or from peripherals and are generated during the execution of I/O instructions or during DMA-controlled transfers. I/O transactions to devices in I/O pages FE^ and FF^| do not generate external bus transactions.

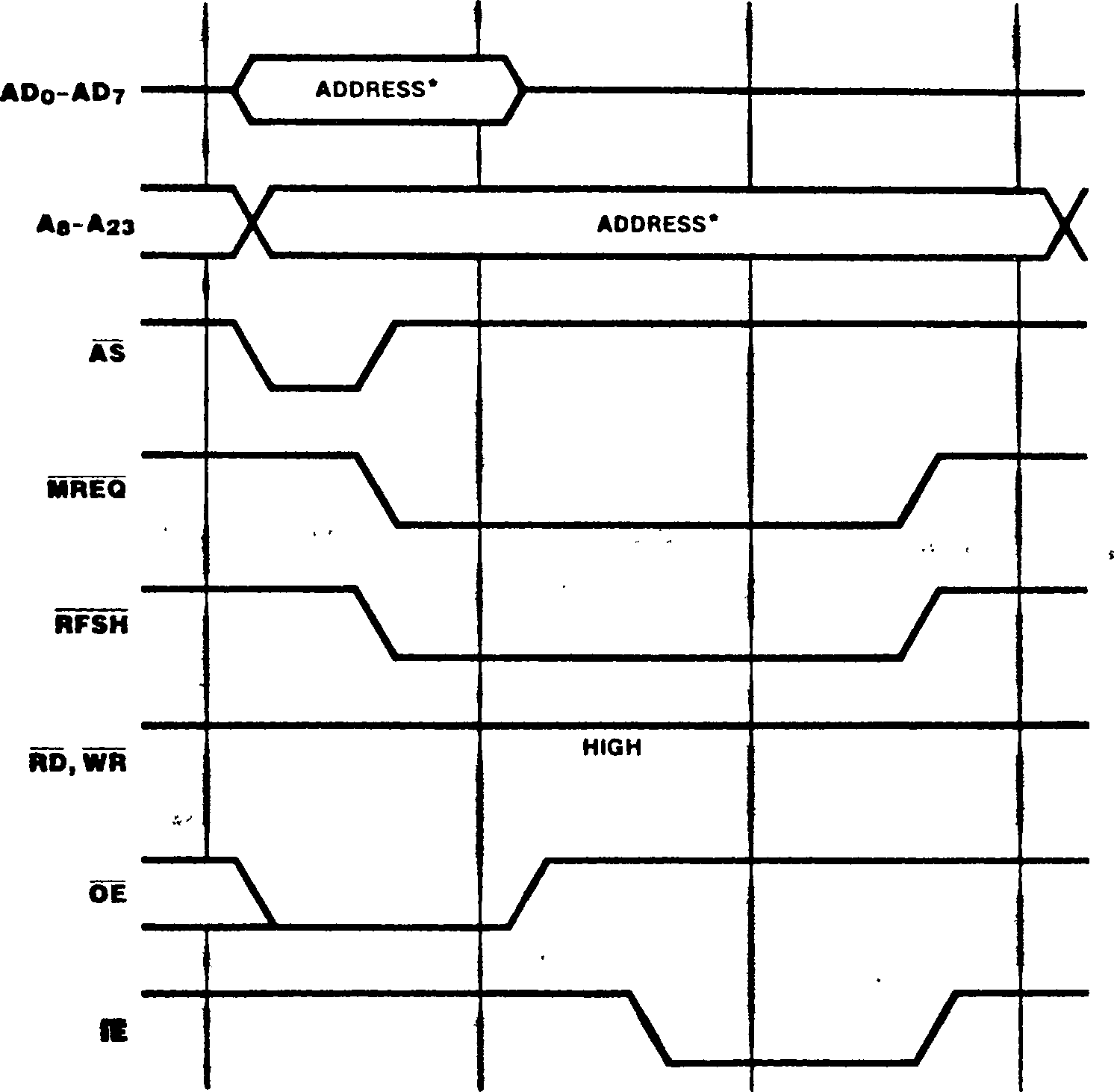
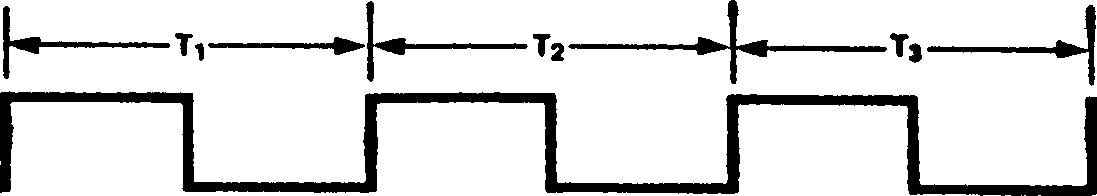
Figures 12-10 and 12-11 illustrate 1/0 transaction timing. I/O transactions are four clock cycles long at a minimum, and, like memory transactions, may be lengthened by the addition of wait cycles. I/O transaction timing is similar to memory transaction timing with one automatic wait state.

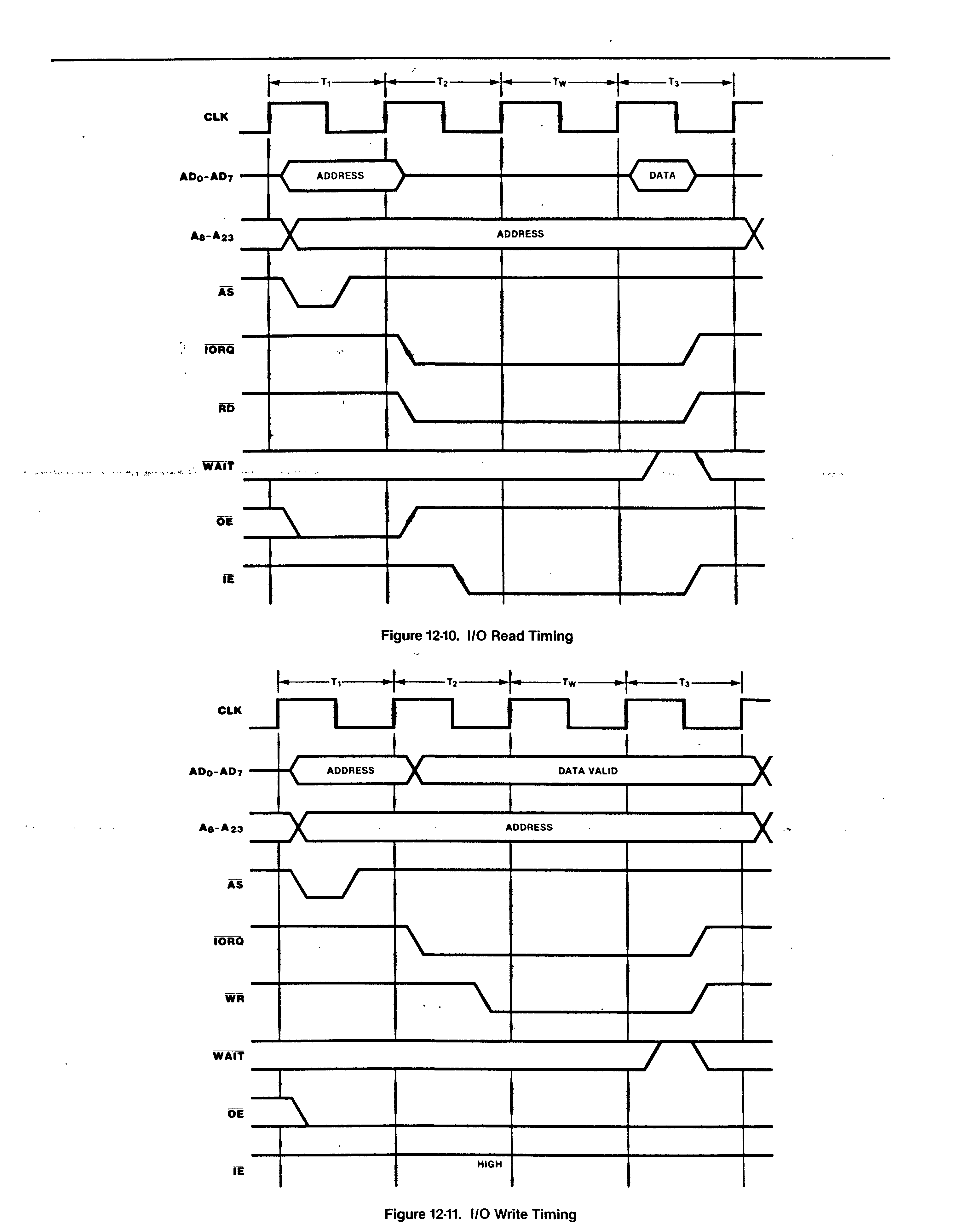
The IORQ line indicates that an I/O transaction is taking place. The 1/0 address is found on ADg-ADy and Ag-A2j when AS rises. For read operations, RD and IE are asserted during the second clock cycle, and input data from the peripheral is sampled by the bus master during the fourth cycle (unless additional wait states are inserted in the transaction). For write operations, WR is asserted during the second cycle with OE remaining asserted; output data to the peripheral is placed on the bus at this time.

CLK

\*10 least significant bits are Refresh address, the rest are undefined.

Figure 12-9. Memory Refresh Timing





* + 1. Interrupt Acknowledge Transactions

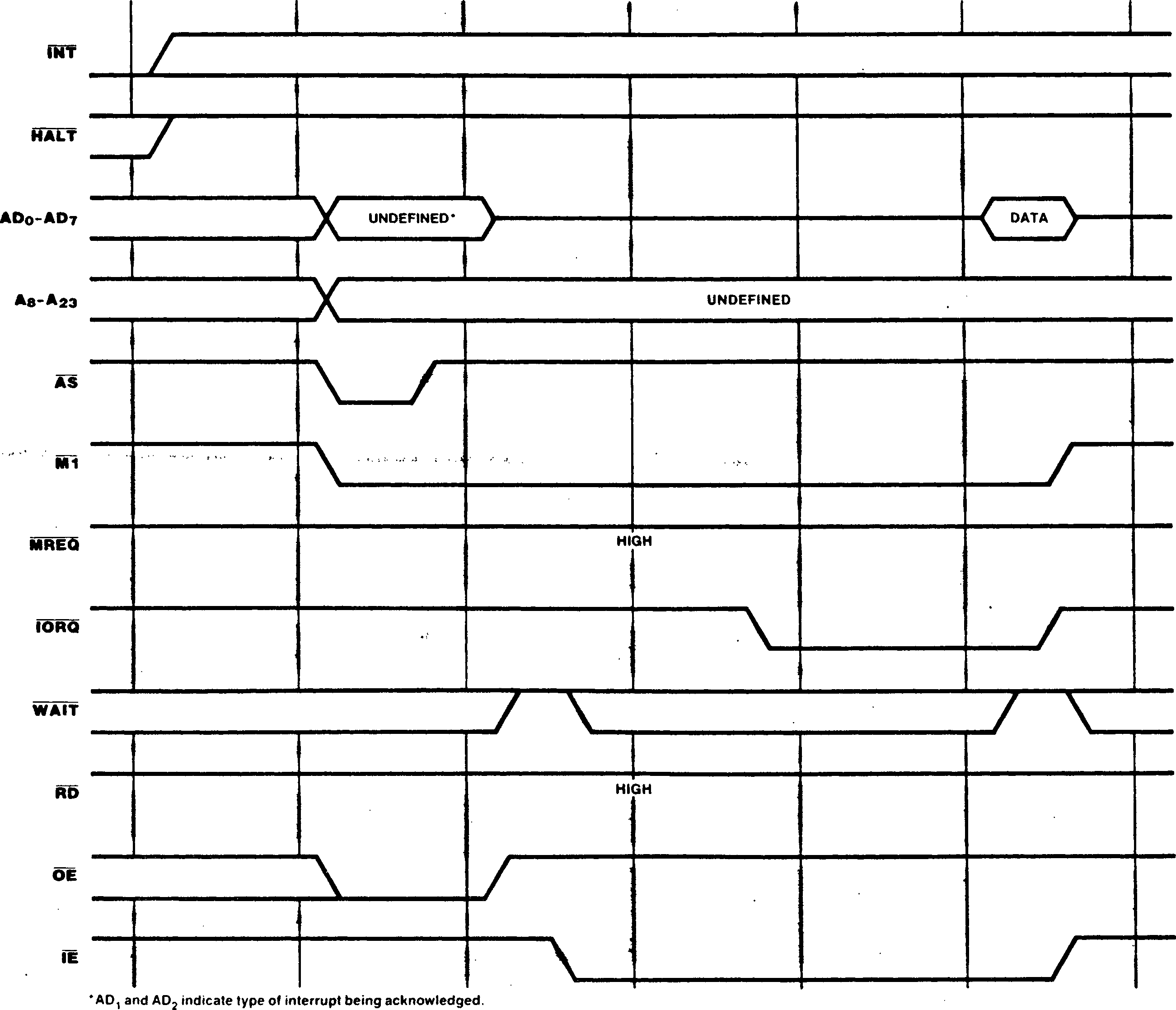
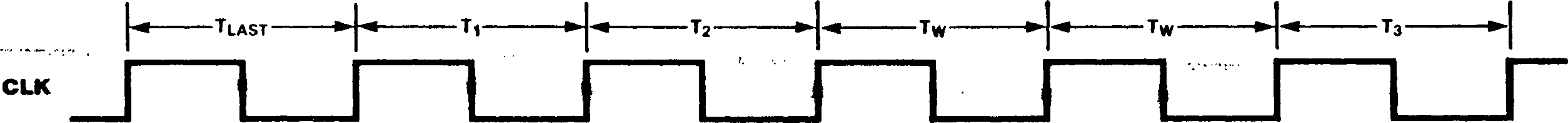
Interrupt acknowledge transactions acknowledge an interrupt and read information from the device that generated the interrupt. These transactions are generated automatically by the CPU when an interrupt request is detected.

Interrupt acknowledge transactions are five cycles long at a minimum, with two automatic wait cycles (Figure 12-12). The wait cycles are used to give the interrupt priority daisy chain (or other priority resolution devices) time to settle before the identifier or vector is read. Additional automatic wait states can be generated by programming the Bus Timing and Control register.

The interrupt acknowledge transaction is indicated by an M1 assertion without MREQ during the first cycle. The AD] and AD2 address lines indicate the type of interrupt being acknowledged when AS is asserted (see Table 6-4); the remaining address lines are undefined. The IORQ signal becomes active during the third cycle to indicate that the interrupting device can place an 8-bit identifier or vector on the bus. It is captured from the AD lines on the falling clock edge before IORQ is raised high.

There are two places where the WAI I line is sampled and, thus, where wait states can be inserted by external circuitry. The first, during T2, serves to delay the falling edge of IORQ to

Figure 12-12. Interrupt Acknowledge Sequence



hardware- and software­can be added to the trans-

WA1T signal is sampled at

during the automatic wait

allow the daisy chain a longer time to settle; the secund, during 13, serves to delay the point at which the identifier or vector is read. Software-generated wait states can also be added at either time via programming of the DC and I/O fields in the Bus liming and Control register. As always, software-generated wait states are inserted into the transaction before the external

WAIT signal is sampled.

* + 1. DMA Flyby Transactions

On-chip DMA channels 0 and 1 can transfer data between memory and peripheral devices using flyby type transfers; external DMA controllers in Z280 MPU systems may also have this capability. The timing of flyby transactions is identical to memory transaction timing, with the exception that the DMA Flyby Strobe (DMASIB) signal is activated; the DMASTB signal is used to select the partici­pating I/O device that must capture or supply the data during the memory access transaction.

Flyby transactions controlled by the on-chip DMA channels always include one automatic wait state (Figures 12-13 and 12-14). As with all memory transactions, other generated wait states action. The external two different times: state and during f3.

For flyby transactions that read from memo?) and write to a peripheral (Figure 12-13), DMASIB is asserted during the automatic wait state and any subsequent wait states added by an active WAIT signal sampled during the automatic wait state. Thus, if the WAIT input is asserted during the automatic wait state, the additional wait states extend the width of the DMASTB pulse. Wait states added via the assertion of WAIT during T3 (after DMASTB is deasserted) stretch the RD signal without affecting DMASIB.

For flyby transactions that read from a peripheral and write to memory (Figure 12-14), DMASIB is asserted at the beginning of T2 and remains asserted until the second half of T3. The WR signal is asserted only during the automatic wait state and any subsequent wait states added by sampling WAI I during the automatic wait state. Wait states added via the assertion of WAII during T3 stretch the DMASTB signal without affecting WR.

Figure 12-13. On-Chip DMA Channel Flyby Memory Read Transaction

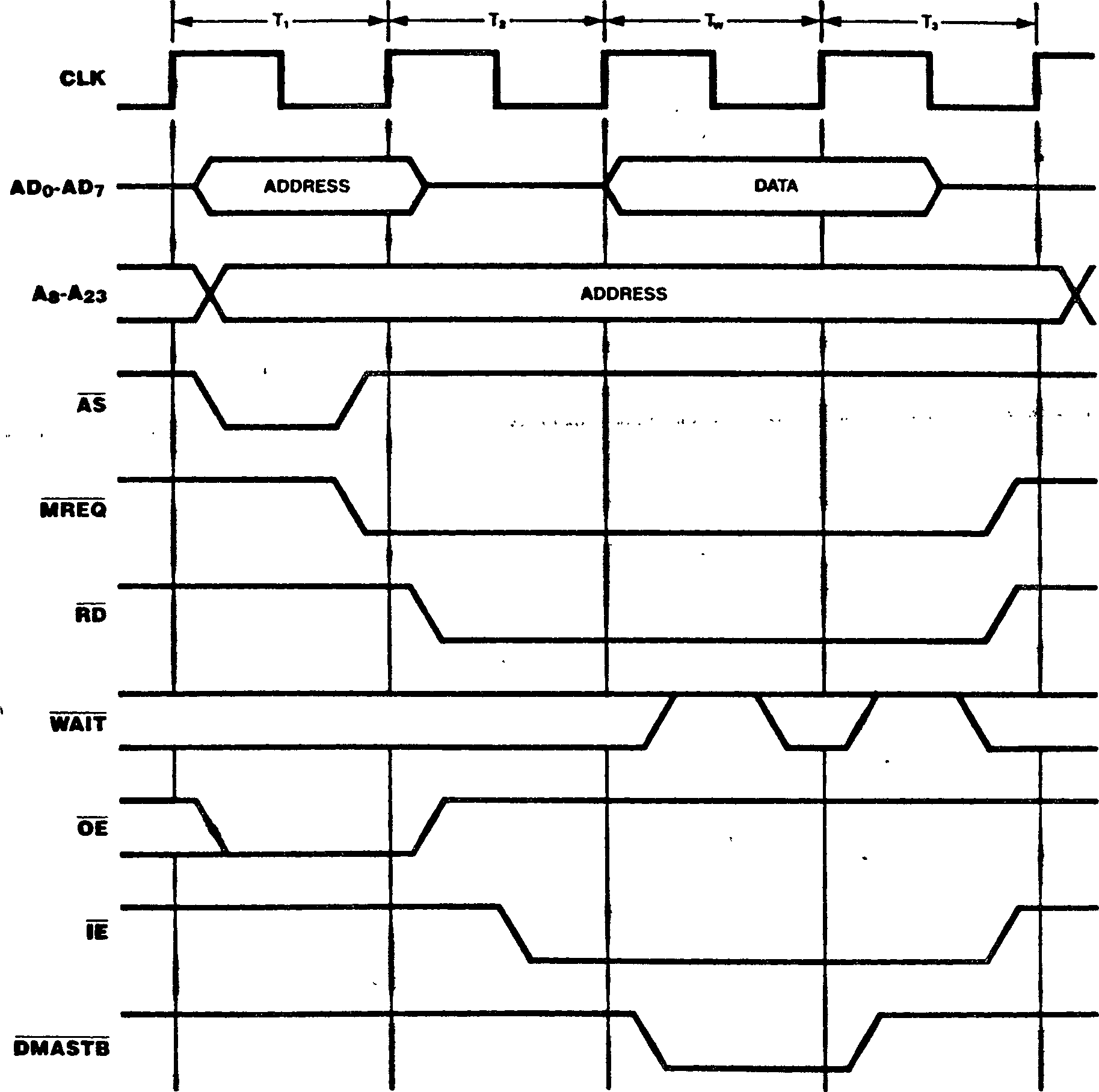
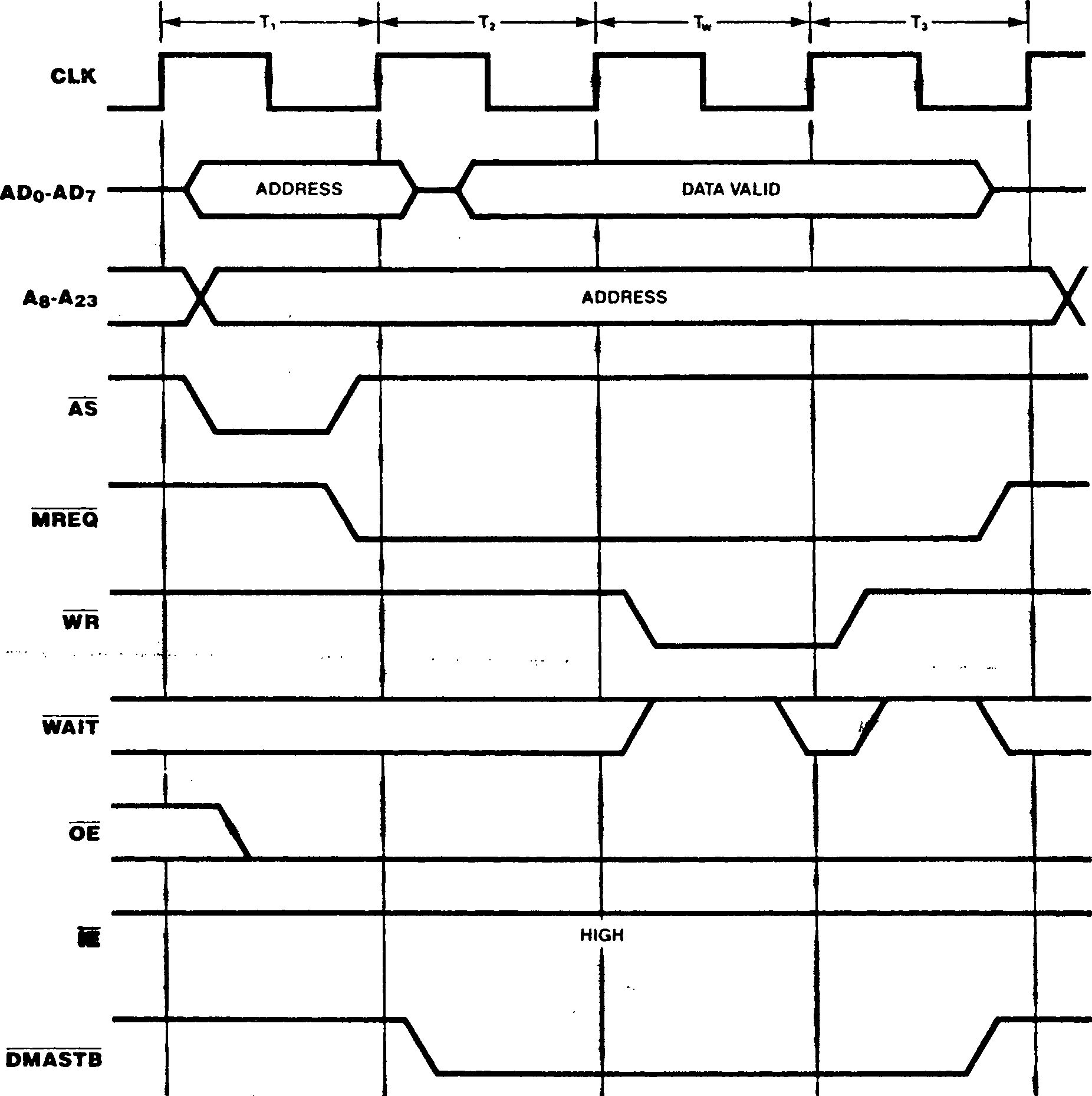


Figure 12-14. On-Chip DMA Channel Flyby Memory Write Transaction



family peripherals request line or, interrupts should peripherals using Iines.

12.6 REQUESTS

The Z280 MPU supports three types of request signals: interrupt requests, local bus requests, and global bus requests. A request is answered according to its type. Interrupt requests are generated by peripheral devices; the Z280 MPU responds with an Interrupt Acknowledge trans­action. Local bus requests are initiated by an external potential bus master; the Z280 MPU responds by relinquishing the bus and generating an active Bus Acknowledge signal. Global bus requests are generated by the Z280 CPU or an on-chip DMA channel to access a global bus; the Z280 MPU receives a Global Bus Acknowledge signal in response to the request.

NMI inputs; several devices can be connected to one interrupt request input, with interrupt priorities established via external logic or a priority daisy chain. However, all Z8400 family peripherals in a Z280-based system will respond to the RETI transaction. Therefore, either all Z8400

should use the same interrupt alternatively, no nesting of

be allowed among the Z8400 different interrupt request

Nonmaskable interrupt requests are edge-triggered, but maskable interrupts are level-triggered. Any high-to-low transition on the NMI input is asynchronously edge-detected, and an internal NMI

latch is set. At the beginning of the last clock

* + 1. Interrupt Requestscycle during execution of an instruction, the maskable interrupt inputs are sampled along with

interrupt enabled in processing

The Z280 CPU supports two types of interrupts, maskable 1NT and nonmaskable (NMI). The interrupt request line from a device capable of generating interrupts can be Lied to the Z280 MPU's INI orthe state of the internal NMI latch. If an

is detected, and that interrupt is the Master Status register, interrupt proceeds in accordance with the current interrupt mode, as described in Chapter 6.

* + 1. Local Bus Requests

To generate transactions on the bus, a potential bus master (such as a DMA controller) must gain control of the bus by making a bus request. A bus request is initiated by pulling BUSREQ low; the Z280 MPU responds by 3-stating its address, data, bus control, and bus status outputs and asserting an active BUSACK, as described in section 10.2. The CPU regains control of the bus after BUSREQ rises. The on-chip DMA channels have higher priority than external devices requesting the bus via BUSREQ.

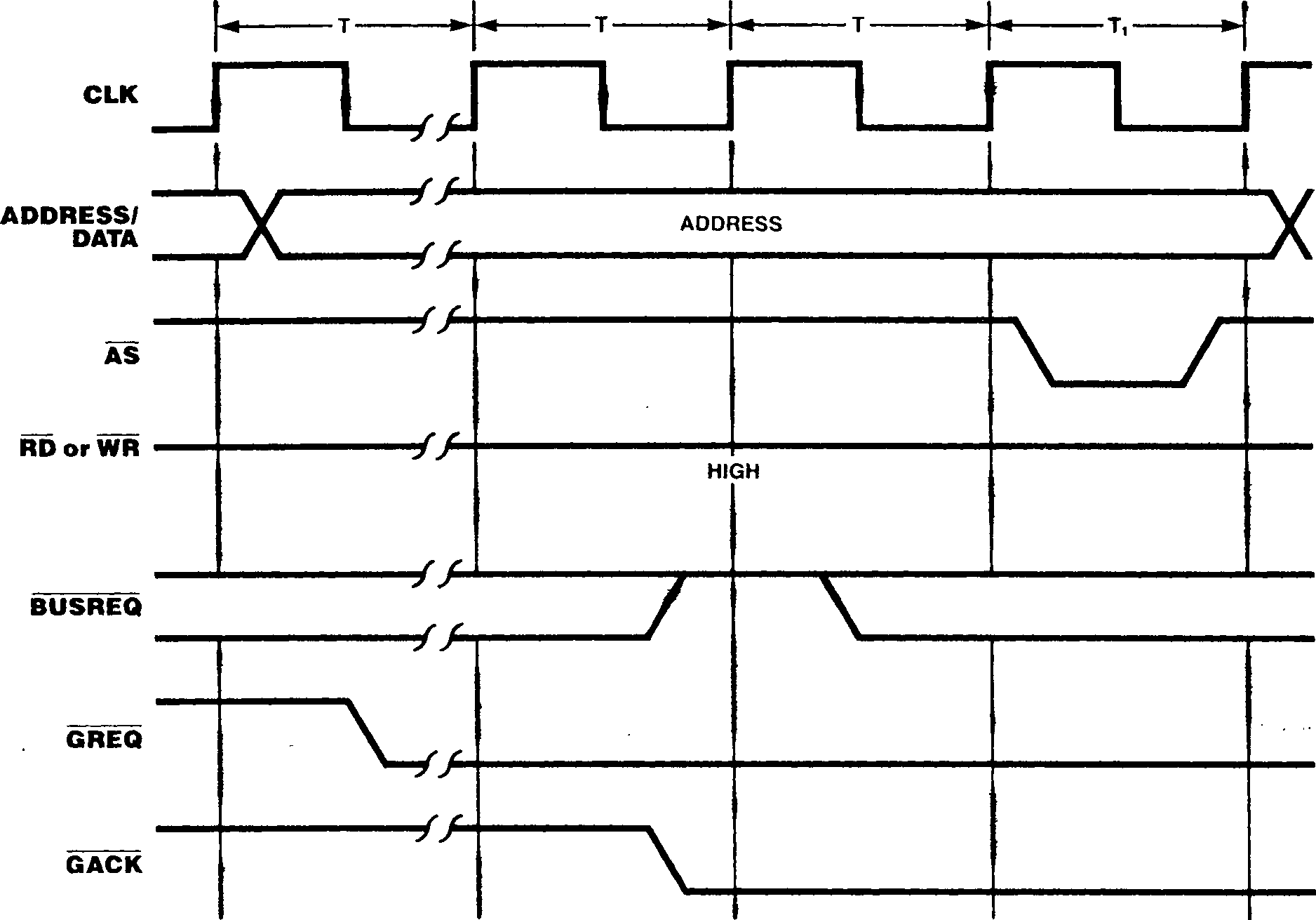
* + 1. Global Bus Requests

If the multiprocessor mode is specified in the Bus Timing and Initialization register, then the contents of the Local Address register determine the range of memory addresses dedicated to the shared global bus. Before accessing an address on the global bus, the Z280 MPU must issue a Global Bus Request (GREQ) and receive an active Global Bus Acknowledge (GACK) signal, as described in Section 10.3.

Figure 12-15 illustrates the timing of the global bus cequest/acknowledge sequence. When the Z280 MPU needs to access a location on the global bus, GREQ is asserted in order to request use of the global bus. GACK is then sampled on each successive rising edge of the clock; when GACK becomes active (and if BUSREQ is not asserted), the memory transaction proceeds as described in section 12.5.1. GREQ is deasserted in the bus clock cycle immediately following the end of the memory transaction (except when executing the Test and Set instruction, where both the memory read and write operations are executed before deasserting GREQ).

■'n. ■ »• ?''' •” ‘ ’ ’’ ’•'i' u '\*■ ? -

Figure 12-15. Multiprocessor Mode Timing



Chapter 13.

Z-BUS External Interface

13.1 INTRODUCTION

peripherals, slave other CPUs, all

The Z280 MPU is typically only one component in a system that may include memory, processors, coprocessors, and connected via a system bus. Two different component-interconnect bus schemes are available with the Z280 bPU: the Z80 Bus and the Z-BUS.

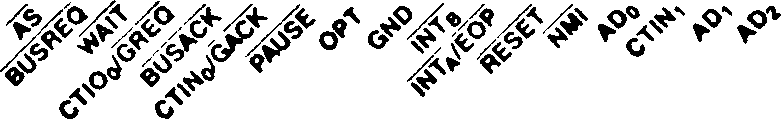
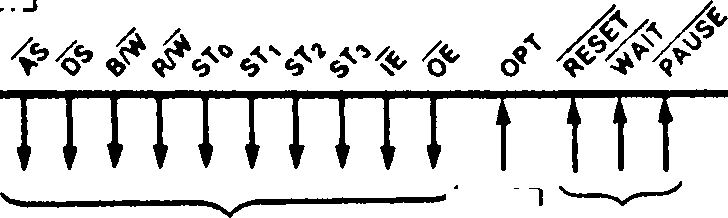
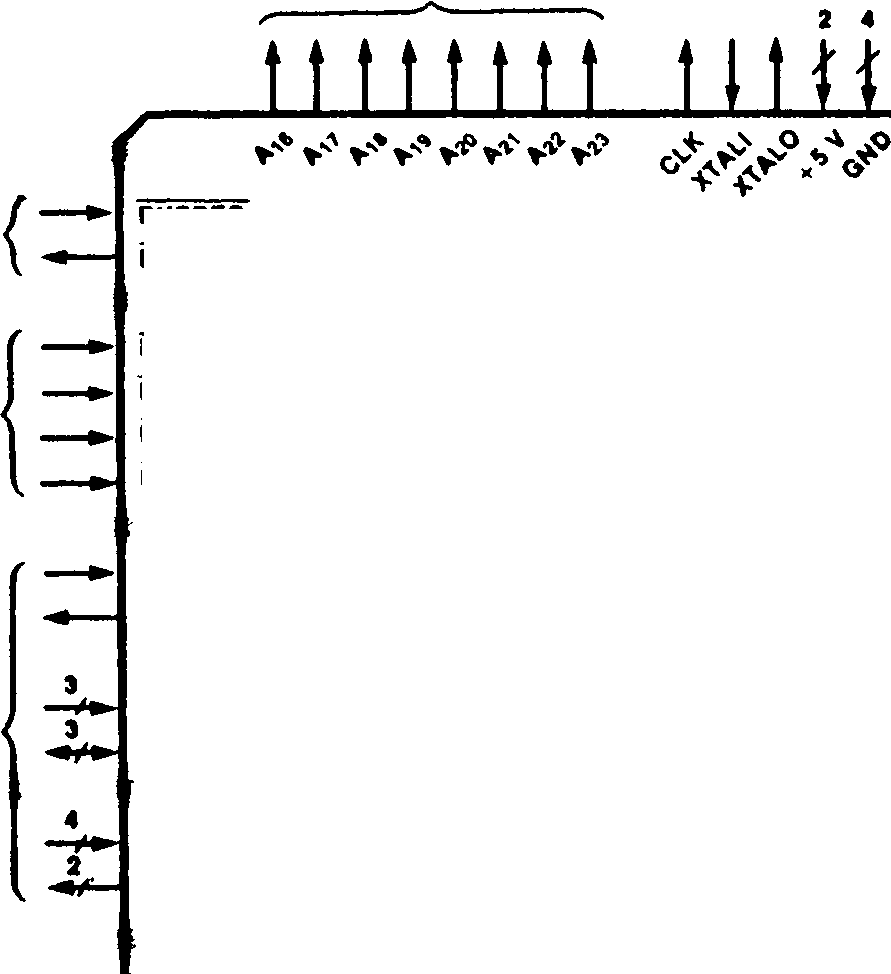
This chapter describes the external manifestations (that is, the activity on the pins) that result from CPU or on-chip peripheral activity for the Z-BUS configurations of the Z280 MPU. (The Z80 Bus external interface is described in Chapter 12.) Since the pins are connected to the system bus, most of this discussion will center on the bus and bus operations.

The condition of the ОРГ pin determines the configuration of the bus interface for the Z280 applying a logical 1 (Vqq) level on the OP I pin or by leaving the ОРГ pin disconnected.

The Z-BUS on the Z280 MPU includes a 24-bit address bus, 16-bit data bus, and associated status and control signals. The data bus is multiplexed with the low-order 16 bits of the address bus. The Z-BUS configuration of the Z280 MPU supports the use of Extended Processing Units and burst-mode memories. Figure 13-1 shows the pin functions and pin assignments for the Z-BUS configuration of the Z280 MPU. The Z-BUS described here is compatible with Zilog’s Z8000 family of peripheral devices. Other Z-BUS

compatible components include the Z8000 family of CPUs. Refer to Zilog’s Component Data Book for a complete description of the Z-BUS Component Interconnect convention.

MPU; the Z-BUS configuration is selected either by



ADDRESS

BUSREQ

•US CONTROL

BUSACK

INTKRRUPTS

ad8

RXD \*

TxD

ADi2

AD13

RDY

AD15

DMASTB

CTIN/GACK

CTIO/GREQ

ADe

ADio ADn

Z280 MPU

Multiplexed with CTINo

Multiplexed with CTIOq

NMI

ЙПА/ЁОР

INTe

INTc

A Do AD, ADa AD3 AD< ADS ADe AD;

ON-CHIP

PERIPHERALS

•US TIMING AND STATUS

✓ NC OR ♦5V CPU CONTROL

|  |  | **».•> >•₽ >•?** |  |  |
| --- | --- | --- | --- | --- |
|  | *\ 9* | 8 7 6 5 4 3 2 1 68 67 66 65 64 63 62 I | 51 |  |
| B/W | 10 |  | 60 | AO, |
| \* DM AST Bq | 11 |  | 59 | A2, |
| R/W | 12 |  | 58 | RDY3 |
| DMASTBi | 13 |  | 57 | **ad5** |
| ST0 | 14 |  | 56 | ROY, |
| ST, | 15 |  | 55 | RDYo |
| OE | 16 |  | 54 | **ad4** |
| IE | 17 |  | 53 | GNO |
| **ADDRESS/** | 18 | **Z280** | 52 | RESERVED |
| **DATA** |  | **' ■ MPU** |  |  |
| ♦ 5V | 19 |  | 51 | GND |
| CTIO, | 20 |  | 50 | XTALI |
| ST2 | 21 |  | 49 | XTALO |
| ST3 | 22 |  | 48 | RxD |
| CTIO2 | 23 |  | 47 | CLK |
| DS | 24 |  | 46 | IkD |
| ctin2 | 25 |  | 45 | A 2Q |
| INTc | 26 |  | 44 | AD3 |
|  | 27 | 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 | 43 |  |

Figure 13-1b. Pin Assignments

**Figure 13-1a. Pin Functions**

**Figure 13-1. Z-BUS Configuration (Input OPT tied to + 5V or not connected)**

a device that does

A request initiated by a

gain the attention of the

Two kinds of operations can occur on the Z-BUS: transactions and requests. At any given time only one device (either the CPU or a bus requestor such as a DMA channel) can be in control of the bus; this device is called the bus master. Trans­actions are always initiated by the bus master and are responded to by some other device on the bus. Only one transaction can proceed at a time.

Requests can be initiated by not have control of the bus.

Seven types of transactions can occur on the Z-BUS, as described below:

**Memory transaction.** CPU- or DMA-controlled

transfer of data to or from a memory location.

**Halt transaction.** Transaction indicating that the CPU is entering the Halt state due to execution of a HALT instruction or a fatal sequence of traps.

**Refresh.** Transaction that refreshes dynamic memory; refresh transactions do not involve a transfer of data.

**I/Ь transaction.** CPU- or DMA-controlled transfer of data to or from a peripheral device.

**Interrupt Acknowledge.** CPU-controlled transaction used to acknowledge an interrupt and read data from the interrupting device.

**EPU transaction.** A transfer of data from an

Extended Processing Unit (EPU) to the CPU.

**DMA Flyby transaction.** A DMA-controlled : transaction that transfers data between a memory location and a peripheral device.

Two types of requests can occur on the Z-BUS, as

described below:

**Interrupt request,** peripheral device to CPU.

**Bus request. A** request by an external device (typically a DMA channel) to gain control of the

bus in order to initiate transactions.

A request is answered by the CPU according to its type: for interrupt requests, an interrupt acknowledge sequence is generated; for bus requests, the CPU relinquishes the bus and activates an acknowledge signal.

The pin functions and assignments for the Z-BUS configuration of the Z280 MPU are illustrated in Figure 13-1. A functional description of each pin is given below:

**A16-A23.** *Address* (output, active High, 3-state). These address lines carry I/O addresses and memory addresses during bus transactions.

ADq>ADi5. *Address/Data* (bidirectional, active High, 3-state). These 16 multiplexed address and data lines carry I/O addresses, memory addresses, and data during bus transactions.

**AS.** *Address Strobe* (output, active Low, 3-state). The rising edge of Address Strobe indicates the beginning of a tra\_nsaction and shows that the address, status, R/W, and B/W signals are valid.

**BUSACK.** *Bus Acknowledge* (output, active Low). A Low on this line indicates that the CPU has relinquished control of the bus in response to a bus request.

**BUSREQ.** *Bus Request* (input, active Low). A Low on this line indicates that an external bus requester has obtained or is trying to obtain control of the bus.

**B/W.** *Byte/Word* (output, Low = Word, 3-state). This signal indicates whether a byte or a word of data is to be transmitted during a transaction.

**CLK.** *Clock Output* (output). The frequency of the processor timing clock is derived from the oscillator input (external oscillator) or crystal frequency (internal oscillator) by dividing the crystal or external oscillator input by two. The processor clock is further divided by one, two, or four (as programmed), and then output on this line.

**CTIN.** *Counter/Timer Input* (input, active High). These lines receive signals from external devices for the counter/timers.

**CTIO.** *Counter/Timer I/O* (bidirectional, active High, 3-state). These I/O lines transfer signals between the counter/timers and external devices.

**DMASTB.** *DMA Flyby Strobe* (output, active Low). These lines select peripheral devices for DMA flyby transfers.

**DS.** *Data Strobe* (output, active Low, 3-state). This signal provides timing for data movement to or from the bus master.

**EOP.** *End of Process* (input, active Low). An external source can terminate a DMA operation in progress by driving EOP Low. EOP always applies to the active channel; if no channel is active, EOP is ignored.

**GACK.** *Global Acknowledge* (input, active Low). A Low on this line indicates the CPU has been granted control of a global bus.

**GREQ.** *Global Request* (output, active Low, 3-state). A Low on this line indicates the CPU has obtained or is trying to obtain control of a global bus.

**IE.** *Input Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is toward the CPU.

**INT.** *Maskable Interrupts* (input, active Low). A Low on these lines requests an interrupt.

**NMI.** *Nonmaskable Interrupt* (input, falling-edge activated). A High-to Low transition on this line requests a nonmaskable interrupt.

**OE.** *Output Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is away from the MPU.

**OPT.** *Bus Option* (input). This signal establishes the bus option during reset as follows:

OPT Bus Interface

0 ........ Z80-Bus, 8-bit ....

1 Z-BUS, 16-bit

**PAUSE.** *CPU Pause* (input, active Low). While this line is Low the CPU refrains from transferring data to or from an Extended Processing Unit in the system or from beginning the execution of an instruction.

**RDY.** *DMA Ready* (input, active Low). These lines are monitored by the DMA channels to determine when a peripheral device associated with a DMA channel is ready for a read or write operation. When a DMA channel is enabled to operate, its Ready line indirectly controls DMA activity; the manner in which DMA activity is controlled by the line varies with the operating mode (single-transaction, burst, or continuous).

**RESET.** *Reset* (input, active Low). A Low on this line resets the CPU and on-chip peripherals.

**R/W.** *Read/Write* (output, Low = Write, 3-state). This signal determines the direction of data transfer for memory, I/O, or EPU transfer transactions.

**RxD.** *UART Receive* (input, active High). This line receives serial data at standard TTL levels.

**ST0-ST3.** *Status* (output, active High, 3-state). These four lines indicate the type of transaction occurring on the bus and give additional information about the transaction. ■»

**TxD.** *UART Transmit* (output, active High). This line transmits serial data at standard TTL levels.

**WAIT.** *Wait* (input, active Low). A Low on this line indicates that the responding device needs more time to complete a transaction. .

**XTALI.** *Clock/Crystal Input* (time-base input). Connects a parallel-resonant crystal or an external single-phase clock to the on-chip clock oscillator.

**XTALO.** *Crystal Output* (time-base output). Connects a parallel-resonant crystal to the on-chip clock oscillator.

**+ 5V.** *Power Supply Voltage.* ( + 5 nominal).

**GND.** *Ground.* Ground reference.

Four Z280 CPU control registers specify certain characteristics of the Z280 MPU’s external interface and determine bus timing: the 3us Timing and Initialization register, Bus Timing and Control register, Local Address register, and Cache Control register.

At any given time, one device (either the CPU or a

bus requester) has control of the bus and is known

as the bus master.

A transaction is initiated by

Bus timing is determined

by the frequency of the

Z280 MPU’s external clock source or crystal and

the bus master and is responded to by some other device on the bus. Information transfers (both instructions and data) to and from the Z280 MPU are accomplished through the use of transactions. All transactions start when Address Strobe (AS) is driven low and then raised high.

the contents of the Bus Timing and Initialization

The state of

Initialization

Local Address transactions described in

Cache Control

register, which receives its initial values as part of the reset process (see section 3.2.1).

The frequency of the processor clock is one-half of the frequency of the external clock source or crystal. The processor clock can be further divided by a factor of 1, 2, or 4 to provide the bus timing clock, as specified by the contents of the Clock Scaling field in the Bus Timing and Initialization register. The bus timing clock is output by the MPU as the CLK signal. In the logical timing diagrams that follow, signal transitions on the bus are shown in relation to the bus clock, CLK.

The number of automatic wait states included in a given transaction is determined by the contents of the Bus Timing and Initialization and Bus Timing and Control registers. The physical memory address space is divided into two sections based on the most significant physical address bit, A2j\* Up to three automatic wait states can be added to transactions to the lower half of memory (addresses where A23 = 0); similarly, up to three automatic wait states can be added to transactions to the upper half of memory (A23 = 1), to all I/O transactions, and to interrupt acknowledge transactions.

the Multiprocessor Configuration

Enable bit in the Bus Timing and register and the contents of the register determine which memory require use of a global bus, as

section 10.3. The contents of the register and the state of the address tags and valid bits in the cache memory determine which transactions employ the cache memory and which transactions use the external bus interface, as described in Chapter 8.

On the rising edge of AS, the bus status signals (SIq-STj, R/W, and B/W) are valid. The SFq-STj status lines indicate the type of transaction being performed (Table 13-1). Typically, these signals are decoded and used to enable the appropriate buffers, drivers, and chip select logic necessary for proper completion of the data transfer.

**Table 13-1. ST Status Line Decode**

**Status Lines 3\*®0 Type of Transaction**

0000 Reserved

0001 Refresh •

0010 I/O transaction

0011 Halt

0100 Interrupt acknowledge line A

0101’ NMI acknowledge

0110 Interrupt acknowledge line В

0111 Interrupt acknowledge line C

1000 Transfer between CPU and memory, cacheable

1001 Transfer between CPU and memory,

non-cacheable

. •- 1010 \* Data transfer between EPU and memory

1011 Reserved

1100 ' EPU Instruction fetch, template, subsequent

• words

1101 EPU Instruction fetch, template, first word

1110 Data transfer between EPU and CPU

1111 Test and Set (data transfers).

If the transaction requires an address, the address is valid on the rising edge of AS. Thus, AS can be used to latch Z280 MPU addresses to de-multiplex the Address/Data lines. No address is required for EPU-CPU or Interrupt Acknowledge transactions; the contents of the A and AD lines are undefined while AS is asserted during these transactions. If an address is generated for a transaction, the Output Enable (0E) signal is activated coincident with AS assertion.

Whether the memory read-modify-write execution of a (sr3-sr0 = 1111).

The Z-BUS MPUs use Data Strobe (DS) to time the transfer of data. For transactions that do not involve the transfer of data (Refresh and Halt transactions), DS is not activated. During write operations (R/W = low), a low on DS indicates that valid data from the bus master is on the Address/Data lines. The Output Enable line continues to be asserted until DS is deasserted. For Read Operations (R/W = high), the bus master drives DS low when the addressed device is to put its data on the bus. Coincident with the assertion of DS during a read operation, the AD lines are 3-stated by the bus master, OE is deasserted, and Input Enable (IE) is asserted. The bus master samples the data on the falling clock edge just before deasserting DS and IE.

The Z280 MPU’s WAIT input provides «э mechanism whereby the timing of a particular transaction can be extended to accommodate a memory or peripheral device with a lonq access time. The WAIT line is sampled on the falling clock edge when data is to be sampled (i.e. just before DS rises) during a transaction. If the WAIT line is low, another bus clock cycle is added to the transaction before data is sampled and DS rises. In this added cycle, and all subseguent cycles added due to WAIT being low, the WAIT line is sampled on the falling edge of the clock and, if it is low, another cycle is added to the transaction. In this way, the transaction can be extended by external logic to an arbitrary length, in increments of one bus clock cycle.

The WAIT input is synchronous, and must meet the specified setup and hold times in order for the Z280 MPU to function correctly. This requires asynchronously-generated WAIT signals to be synchronized to the CLK output before they are input into the Z280 MPU. Automatic wait states can also be generated by programming the Bus Timing and Control register and Bus Timing and Initialization register; these are inserted in the transaction before the external WAIT signal is sampled.

* + 1. Меяюгу Transactions

Memory transactions move instructions or data to or from memory when a bus master makes a memory access. Thus, they are generated during program execution to fetch instructions from memory and to fetch and store memory data. They are also generated to store old program status and fetch new program status during interrupt and trap handling, and to transfer information during DMA- controlled memory accesses. A memory transaction is three bus cycles long unless extended with hardware- and/or software-generated wait states, as explained previously.

During memory transactions, the STj-STg status lines indicate that a memory transaction is occurring and provide the following information:

* Whether the memory access is cacheable (ST3-STq = 1000) or noncacheable (SFj-STq = 1001).
* Whether the memory access is a fetch of an extended instruction’s template intended for an EPU (ST3-Sl0 = 1100 or 1101).
* Whether the data is supplied or captured by an Extended Processor Unit while executing an extended instruction (Sf3-STq = 1010).

access is part of an atomic operation during the Test and Set instruction

A memory read is distinguished from a memory write via the R/W signal. ' r- -vj .

* + - 1. Byte/Word Organization

The byte is the basic addressable memory element in Z280 MPU systems. However, although memory is addressed as bytes, the Z-BUS configuration of the Z280 MPU has a 16-bit data path, and memory trans­actions can be byte or word transfers. Each 16-bit word in memory is made up of two 8-bit bytes, where the least-significant byte preceeds the most-significant byte of the word, as in the Z80 CPU architecture. For example, the word at memory location 5000ц has its low-order byte at location 5000ц and its high-order byte at location 5001ц.

Bytes transferred to or from odd memory locations (address bit 0=1) are always transmitted on lines ADg-ADy. Bytes transferred to or from even memory locations (address bit 0=0) are always transmitted on lines ADg-AD^. For byte reads B/W = high, R/W = high), the CPU or on-chip DMA channel uses only the byte whose address it put out on the bus. In other words, for a byte read with an odd address, the CPU or DMA channel will only read the lower half of the bus; for a byte read with an even address, the CPU or DMA channel will only read the upper half of the bus. For byte writes (B/W = high, R/W = low), the CPU or on-chip DMA channel (flowthrough mode) places the byte to be written on both halves of the bus, and the proper byte must be selected in the memory control logic by testing address bit 0.

For word transfers (B/W = low), all 16 bits are captured by the CPU or DMA channnel during reads (R/W = high) or stored by the memory during writes

(R/W - low). The most-significant byte of the word is transferred or. ADq-ADj and the least­significant byte on ADg-ADjc,; thus, the bytes of data will appear swapped on the bus, with the most significant byte on the lower half of the bus and the least significant byte on the upper half of the bus. Word transfers always use even-valued addresses (address bit 0 = 0) and result in an access to the byte at the even address and the next consecutive byte at the following odd address. For example, a word access to location 5000^ would access the byte at location 5000H (transferred on ADg-AD^) and the byte at location 5001 p| (transferred on ADq-AD?).

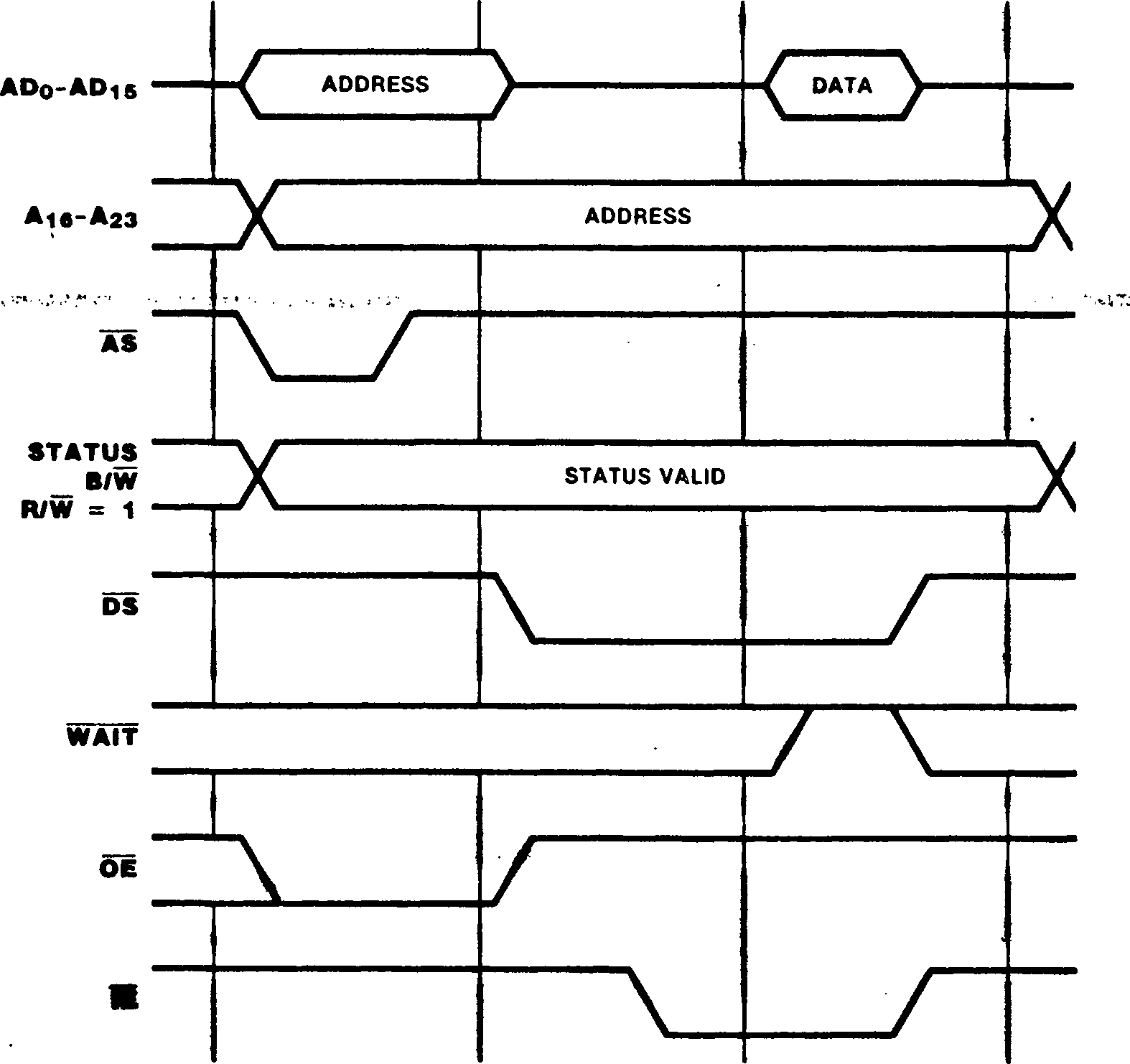
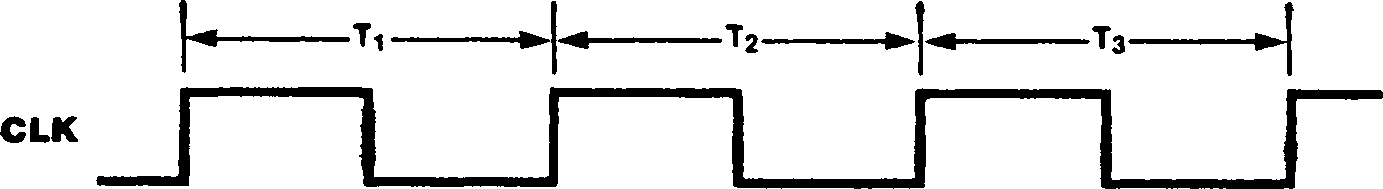
Instruction fetches are always executed as word transactions. However, instruction opcodes need not be aligned on even-address boundaries; the CPU will use only one byte of the fetched word if appropriate.

Data accesses may be byte or word accesses. Data words aligned at even-address memory boundaries are accessed via one word transaction. Data words on odd-address boundaries are accessed via two consecutive byte transactions.

* + - 1. Memory Transaction Timing

Memory transaction timing is illustrated in Figures 13-2 and 13-3. During the first bus cycle, AS is asserted to indicate the beginning of a transaction; Output Enable (0E) is also asserted at this time. All address and status information is guaranteed valid on the rising edge of AS. The ЯГ0-5Г3 status lines indicate that a memory trans­action is occurring. For a read operation (Figure 13-2), DS is activated during the first half of the second bus cycle, after the bus master has 3-stated the AD lines; 0E is deasserted at the beginning of the second cycle and Input Enable (IE) is asserted during the second half of the second cycle. The bus master samples the information returned from memory on the Address/ Data bus on the falling edge of the clock during the third bus cycle; after the data is sampled, OS and IE are deasserted. For a write operation (Figure 13-3), DS is asserted during the second half of the second cycle, after the bus master has placed the data to be written on the AD lines, and 0E stays active throughout the transaction.

Figure 13-2. Memory Read Timing



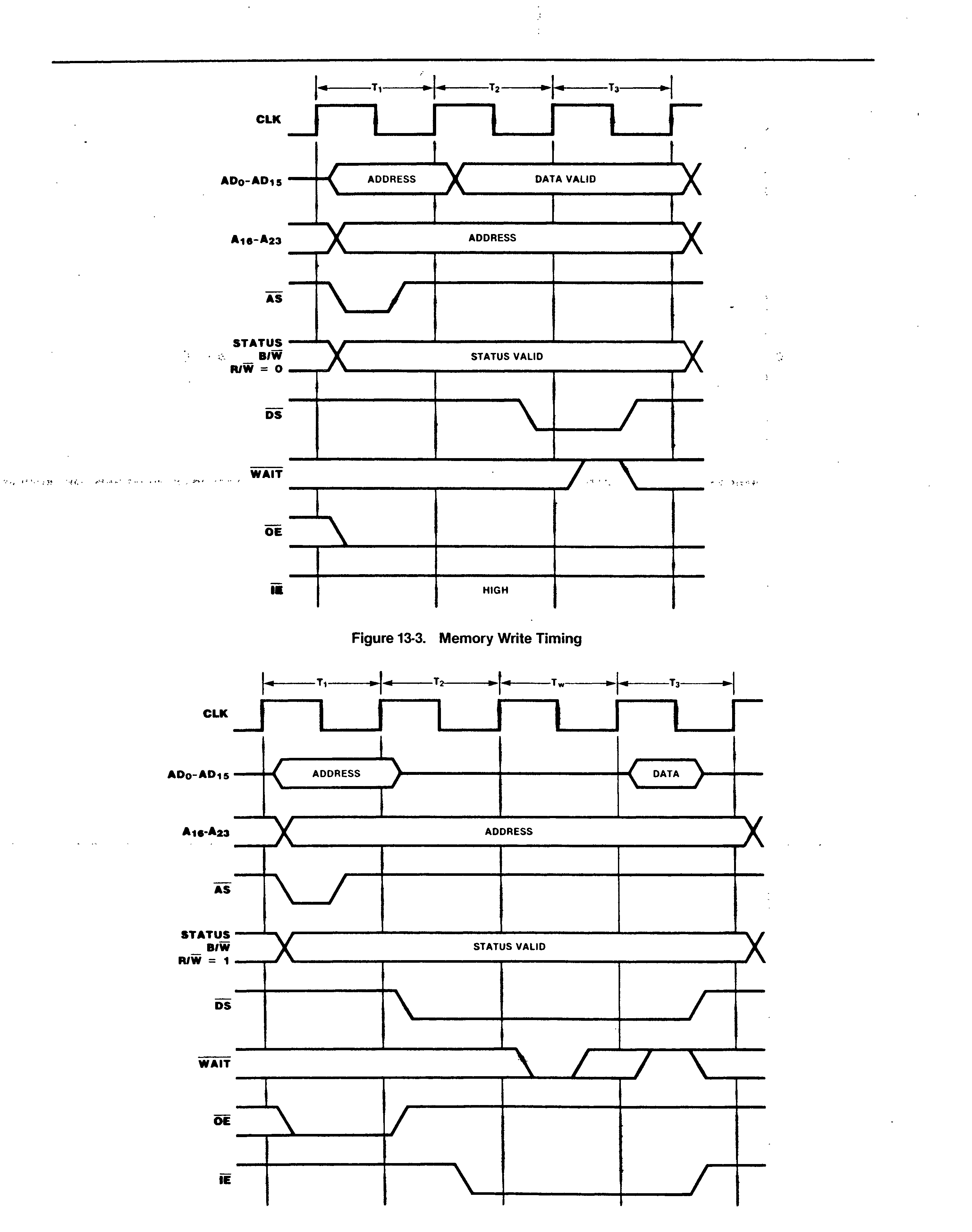
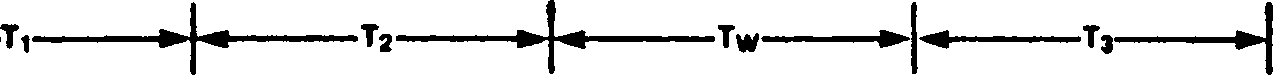
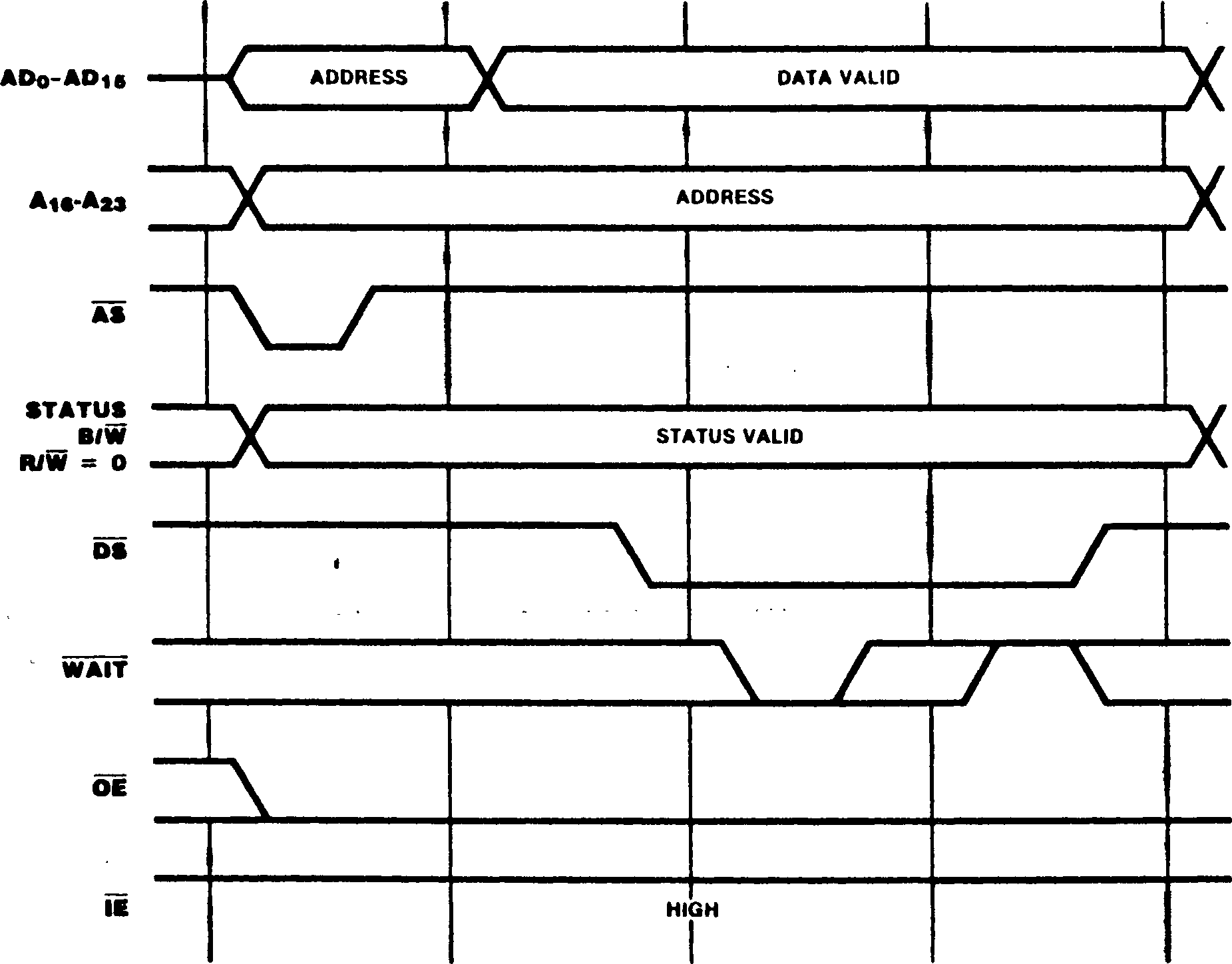
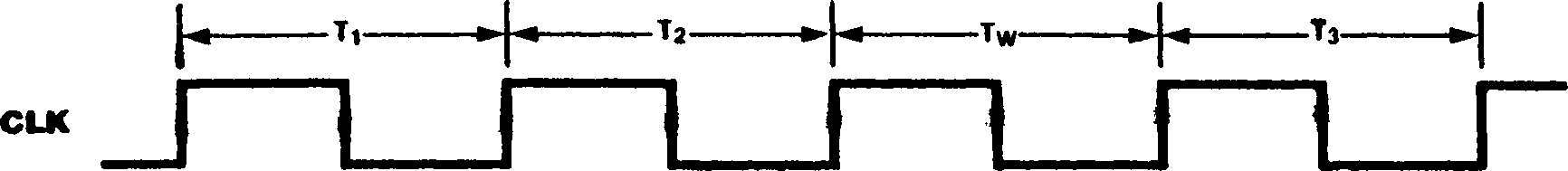


Figure 13-4. Memory Read Timing with External Wait Cycle

Figure 13-5. Memory Write Timing with External Wait Cycle



“J—! l" I . :—LJ-i\_r

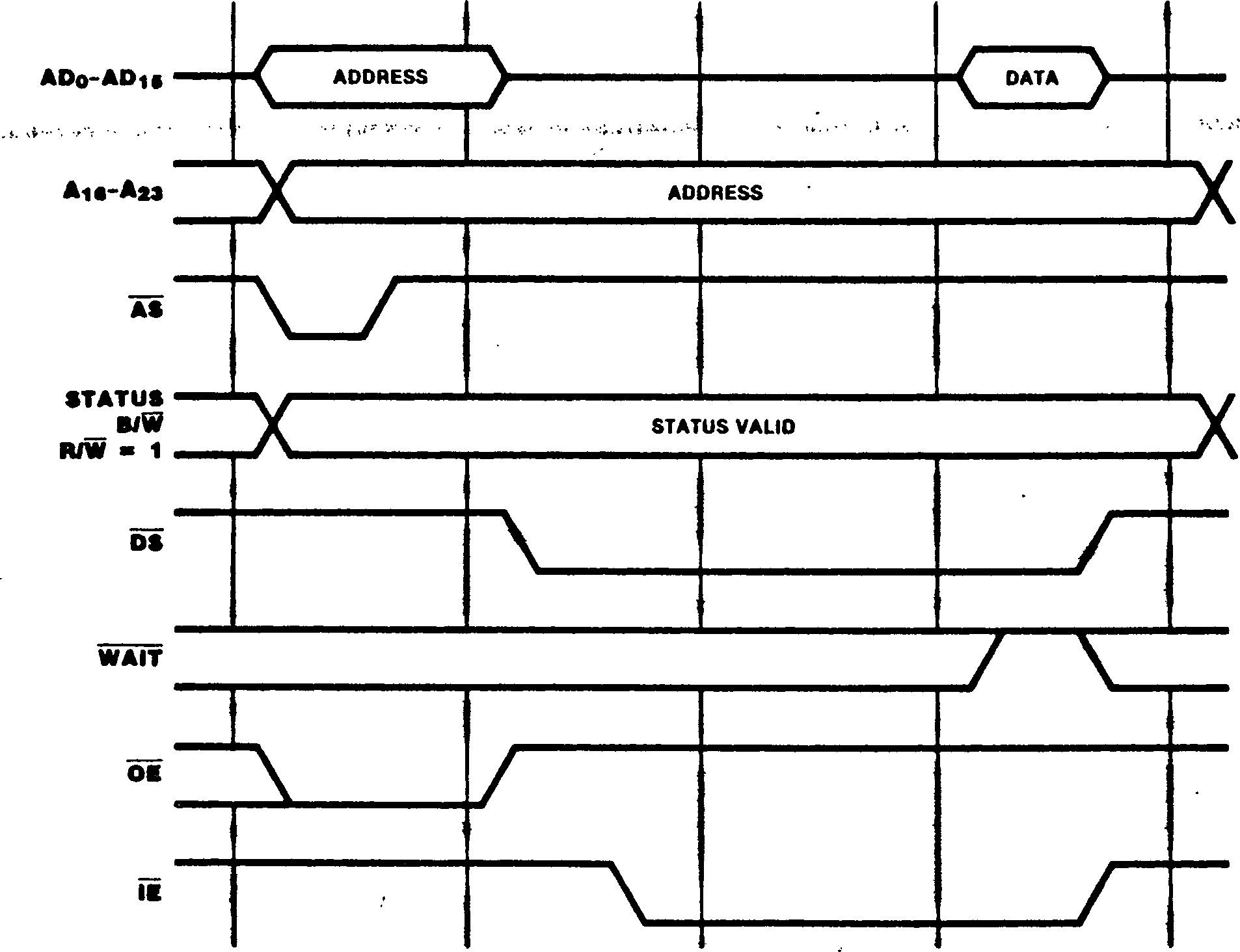


Figure 13-6. Memory Read Timing with Internal Wait Cycle

identical

The WAII input is also sampled on the falling edqe of the clock during the third clock cycle; if WAIT is low, another bus clock cycle is added before sampling the data. Wait states can also be added through programming of the Bus Timing and Initialization register and Bus Timing and Control register. For example, Figures 13-4, 13-5, and 13-6 illustrate memory transactions with one wait state.

* + - 1. Burst Meeory Transactions

The Z-BUS configuration of the Z280 MPU supports a special kind of memory transaction called a "burst memory transaction" for use in systems employing burst-mode memory devices. Control bits in the Cache Control register indicate whether portions of the memory system can support burst transactions; burst mode can be specified for either the upper half of memory (A23 = 1)> the lower half of memory (A23 = 0), or both. . /

Burst memory transactions are used only during instruction fetches to "prefetch" instructions into the on-chip cache. In a burst memory read, four consecutive words of memory are read. If a byte is to be read from a portion of external memory that supports burst transactions, and that read operation is cacheable, the CPU reads the four words that contain the desired byte of the instruction with a single burst transaction. The address of the first word read during a burst transaction has zeros in the three least significant bits. The CPU reads a total of eight bytes via four word transfers, where the last byte read has all ones in the three least significant bits of its address. This effectively increases the bus bandwidth by prefetching a cache block on a cache miss. Burst transactions are not used when fetching templates in extended instructions.

The timing of a burst transaction is illustrated in Figure 13-7. During burst transactions, four Data Strobes are generated with a single Address

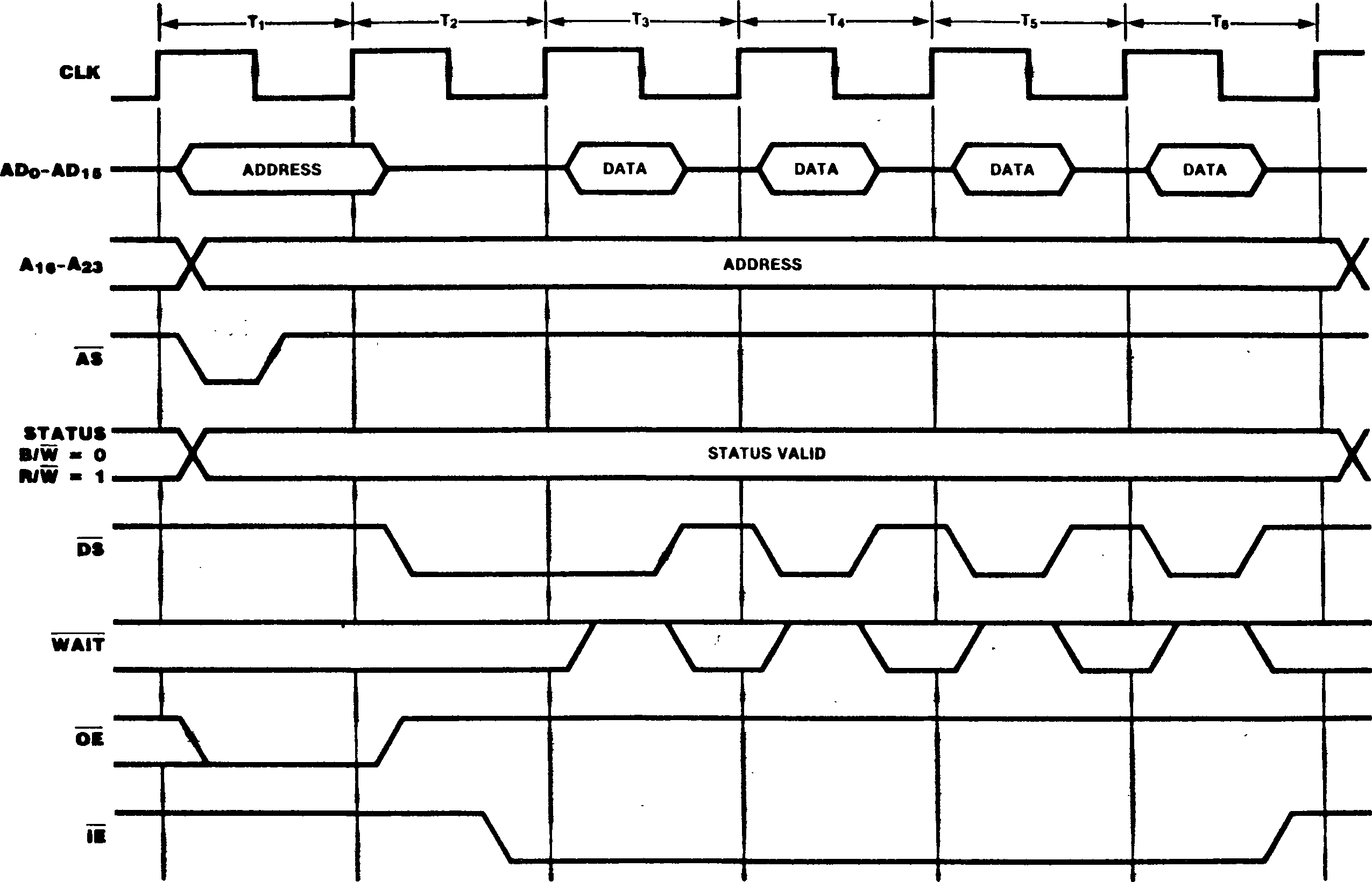
Strobe. Timing for the first data transfer is

to that for a single memory read,

including the insertion of automatic wait states.

This first transfer is immediately followed by three more transfers in the next three bus clock cycles. The WAIT input is sampled during each transfer and any resulting wait states, thereby allowing wait states to be added before any of the transfers. However, automatic wait states are added only before the first transfer.

Figure 13-7. Burst Memory Read Timing



* + - 1. Test and Set Memory Transactions
    1. Halt and Refresh Transactions

These transactions transactions, except WAIT input is not transferred.

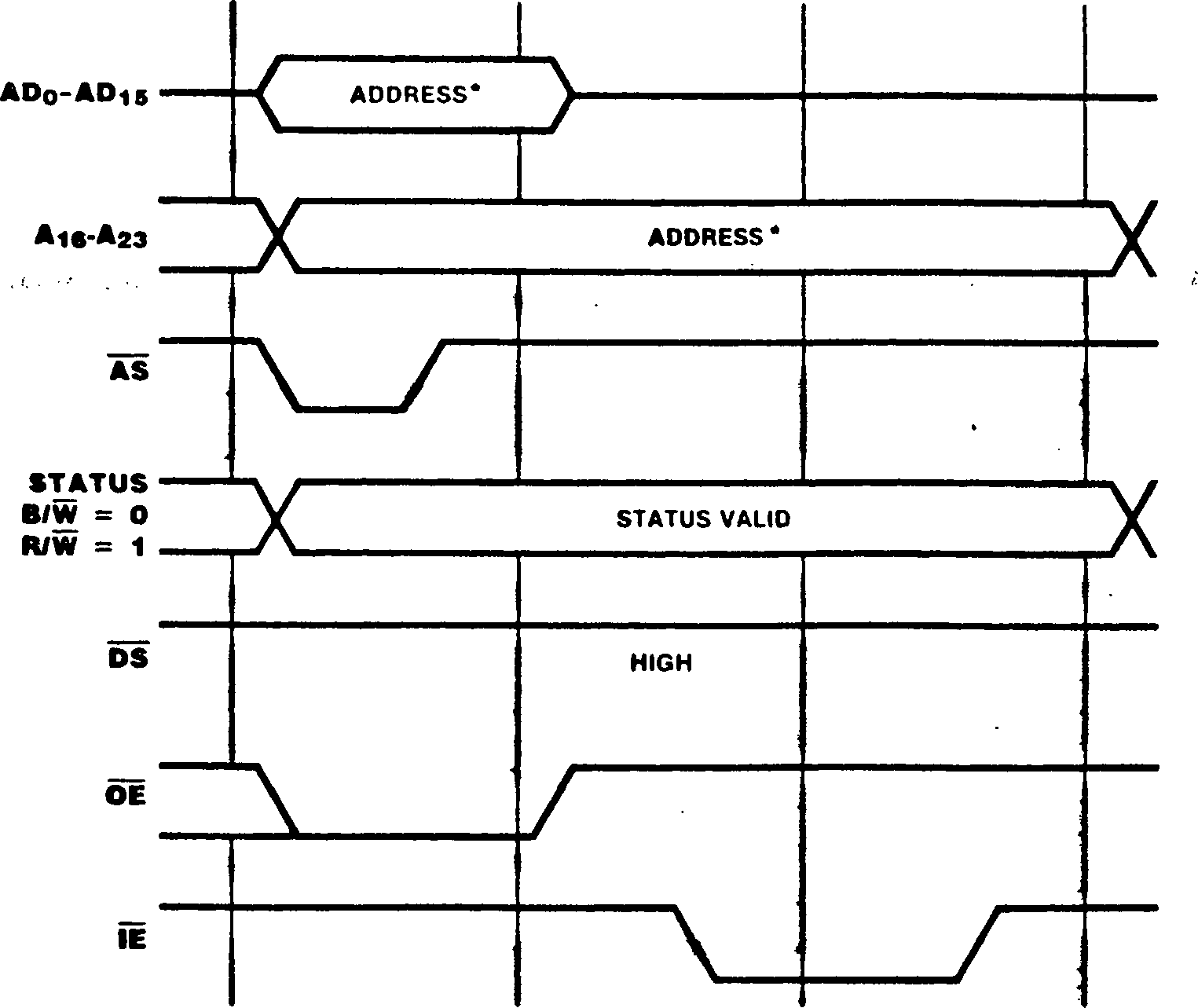
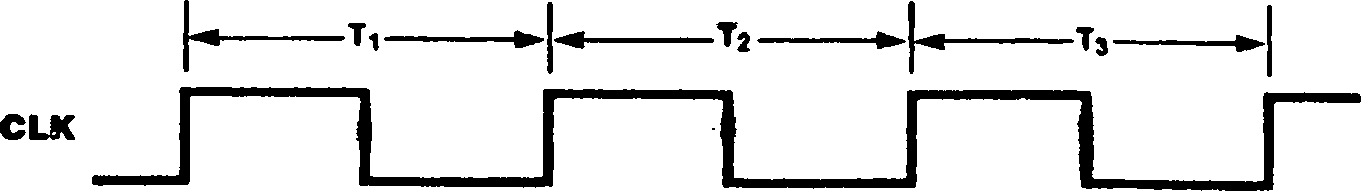
The lest and Set (TSET) instruction provides a locking mechanism that can be used to synchronize software processes in a multitasking or multi­processor system where exclusive access to certain resources is required. TSET tests and sets semaphores that control access to shared resources. Execution of TSET involves a memory read followed immediately by a memory write; the memory read followed by the memory write is one indivisible operation. The testing and setting of a semaphore requires the semaphore to be read from memory, modified, then written back into the same memory location. During the first of these two memory operations, the ”1111” status code is placed on the STj-STq status lines. This is particularly useful in a multiple microprocessor environment with semaphores in a shared memory area. The Test and Set status code can be used to control external circuitry that precludes memory access by another processor during the Test and Set semaphore operation. Furthermore, the BUSREQ input is disabled during a Test and Set operation to ensure that the semaphore is tested and set without any intervening accesses.

There are two kinds of bus transactions that do not transfer data: Halt and Refresh transactions.

are similar to memory that DS remains high, the sampled, and no data is

The Halt transaction (Figure 13-8) is generated when a HALT instruction is encountered or a fatal sequence of traps occurs. The ”0011” status code on the STj-STg lines identifies the Halt transaction. For Halt transactions generated by the HALT instruction, once the Halt transaction is executed, all subsequent CPU activity is suspended until an active interrupt request or reset is detected. After Halt transactions generated due to a fatal condition, all CPU activity is suspended until an active reset is detected (see section 6.6). However, Refresh transactions or DMA transfers may occur while the CPU is in the Halt state; also, the bus can be granted. The address emitted during the address phase of the Halt transaction is the address of the Halt instruction or the instruction that initiated the fatal sequence of traps. ?

• Address of Halt Instruction.



t

**Figure 13-8. Halt Timing**

A memory refresh transaction (Figure 13-9) is generated by the Z280 MPU refresh mechanism and can occur immediately after the final clock cycle of any other transaction. The memory refresh counter’s 1O-bit address is emitted on ADQ-AD9 when AS is asserted; the contents of the remaining address lines are undefined. The ”0001” status code on the STj-STq lines identifies the Refresh transaction. This transaction can be used to generate refreshes for dynamic RAMs. Refreshes may occur while the CPU is in the Halt state.

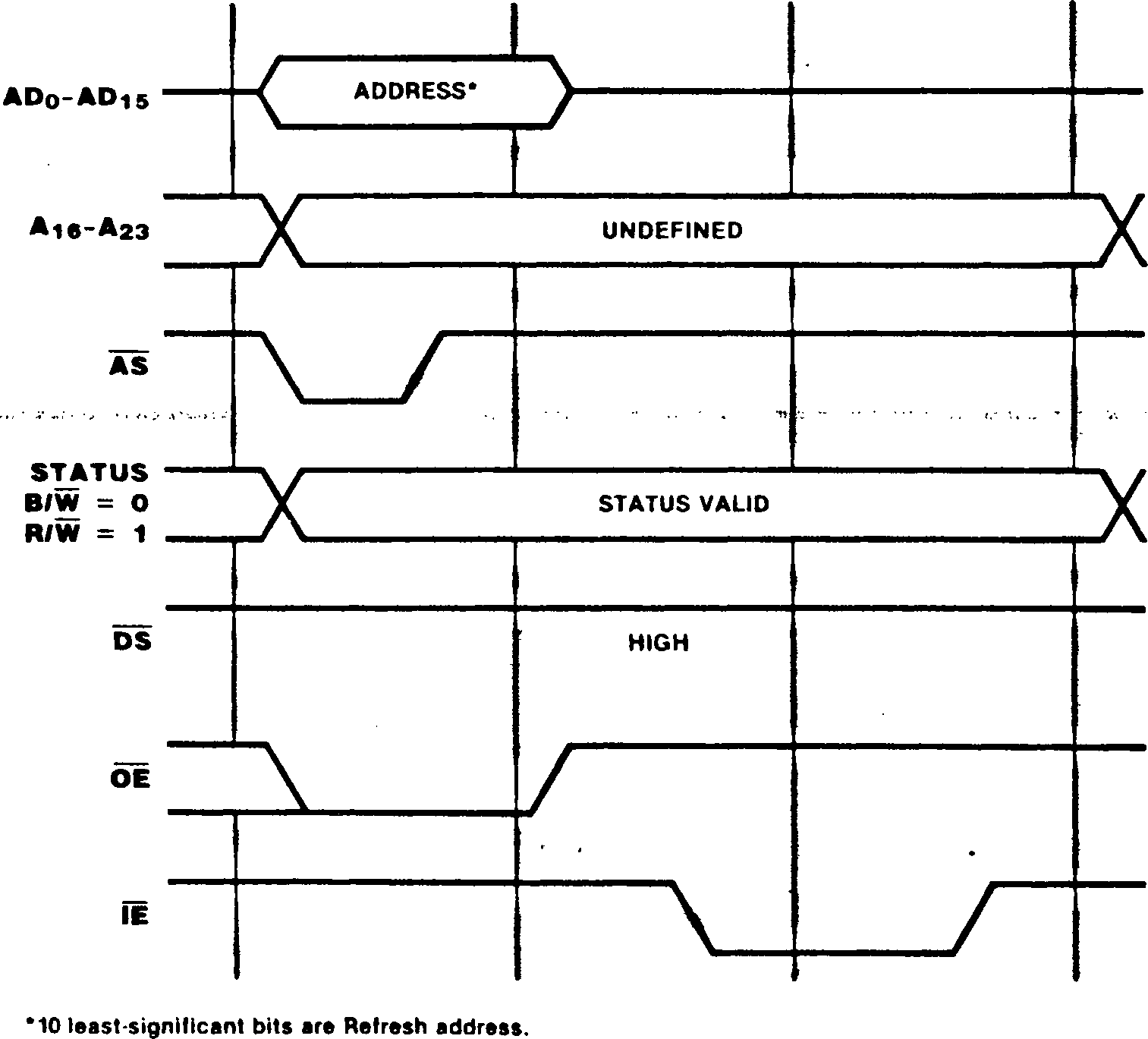
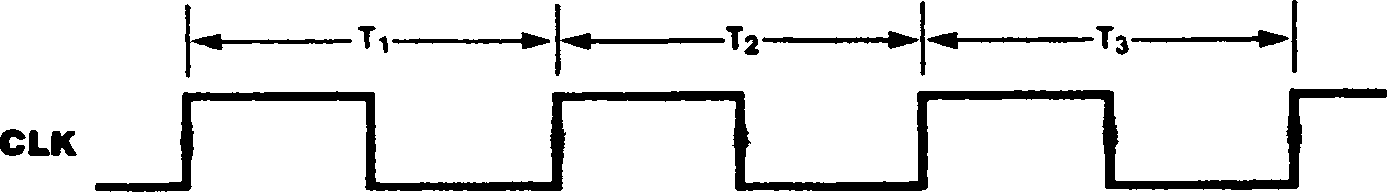
* + 1. 1/0 Transactions

I/O Transactions move data to or from peripherals and are generated during the execution of I/O instructions or during DMA-controlled transfers. I/O transactions to devices in I/O pages FEj^ and FF^ do not generate external bus transactions.

Figures 13-10 and 13-11 illustrate 1/0 transaction timing. 1/0 transactions are four clock cycles long at a minimum, and, like memory transactions, may be lengthened by the addition of wait cycles. I/O transaction timing is similar to memory transaction timing with one automatic wait state. The ”0010” status code on the STj-STq lines indicates that an 1/0 transaction is taking place, and the R/W line indicates the direction of the data transfer. The 1/0 address is found on ADq-AD^ and A16-A23 when AS rises. For read operations, DS and IE are asserted during the second clock cycle, and input data from the peripheral is sampled by the bus master during the fourth cycle (unless additional wait states are inserted in the transaction). Note that DS falls near the middle of T2 for 1/0 read transactions (as opposed to the beginning of T2 for memory reads); this provides peripheral control logic with additional time for address decoding. For write operations, DS is asserted during the second cycle with 0E remaining asserted; output data to the peripheral is placed on the bus at this time.

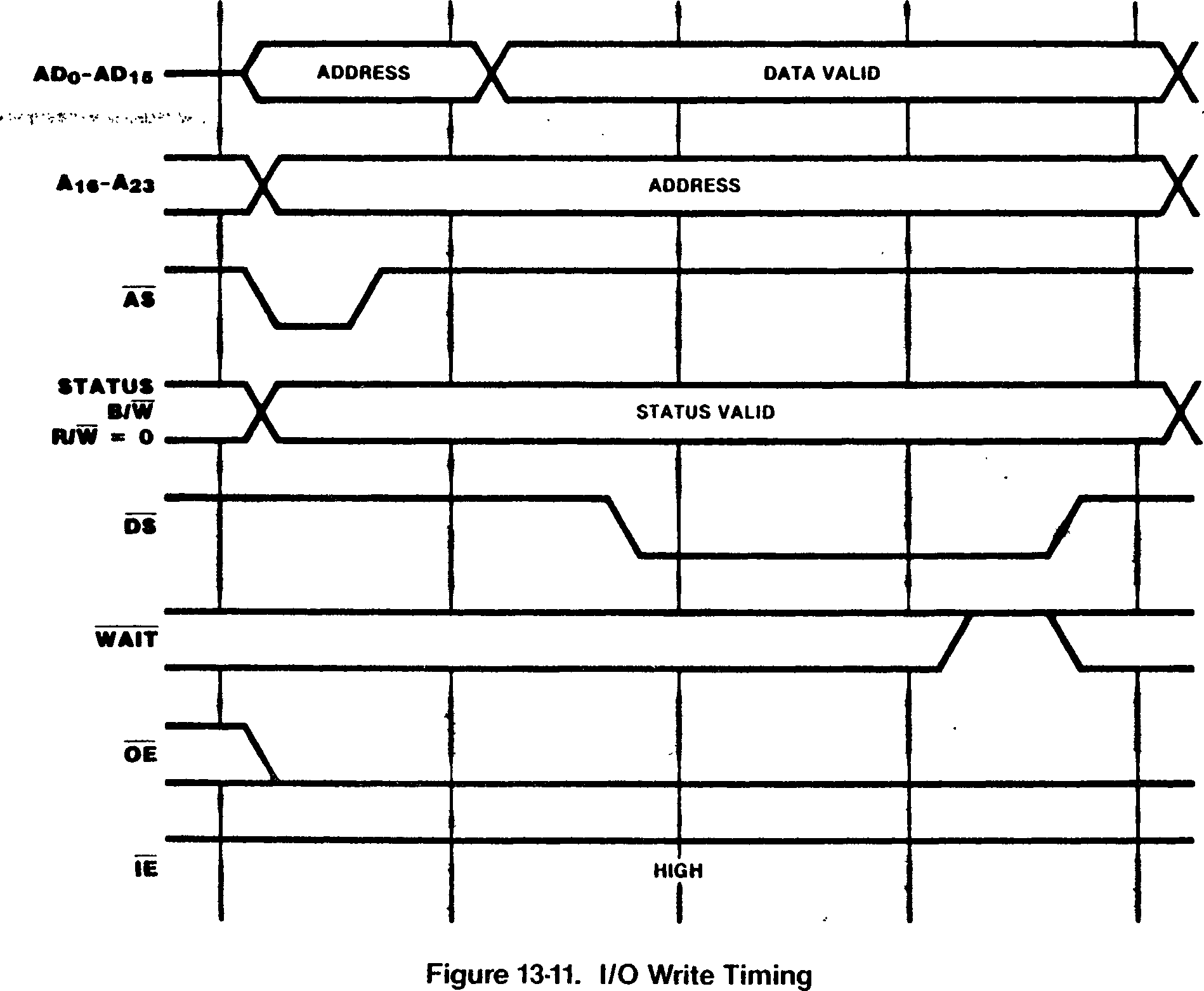
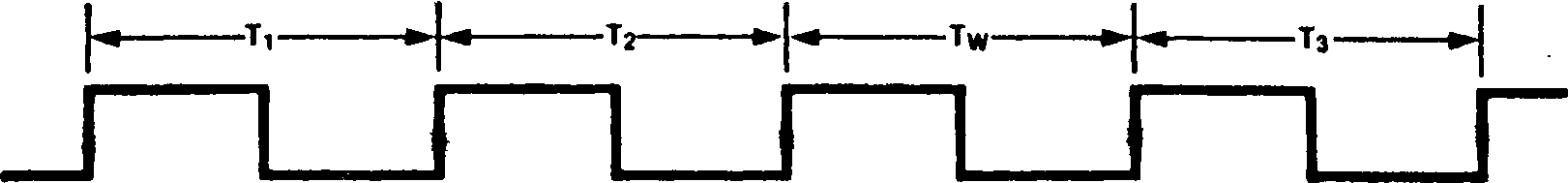
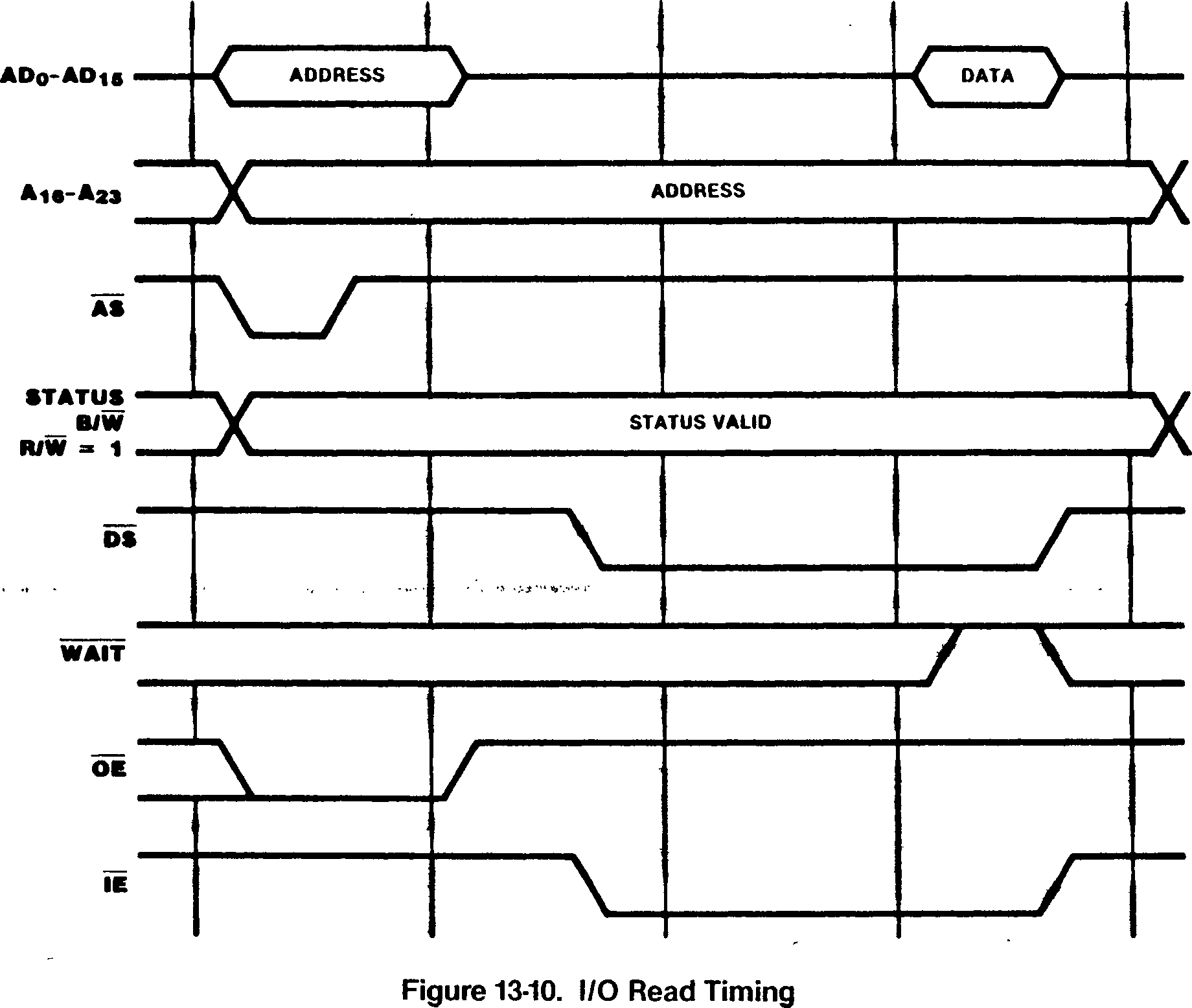
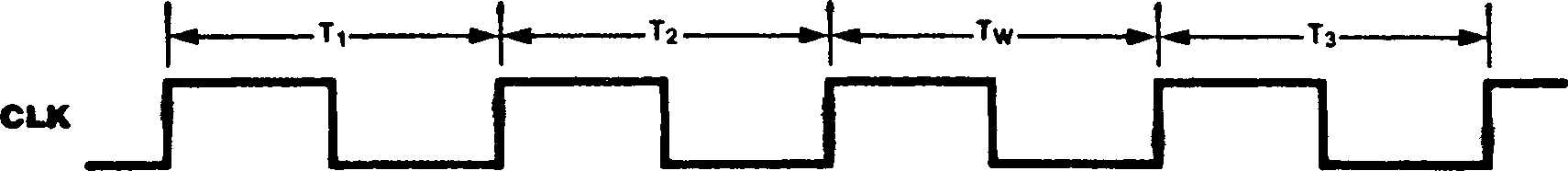
For byte I/O operations (B/W = high), the byte of data is always transferred on the ADg-ADy bus lines, regardless of the address of the peripheral device. For word 1/0 operations, the most significant byte of data is transferred on ADg-ADy and the least significant byte on ADg-AD^, as with word memory transactions.

Figure 13-9. Memory Refresh Timing



13-12

**cue**



* + 1. **Interrupt Acknowledge Transactions**

Interrupt Acknowledge transactions acknowledge an interrupt and read an identifier from the device that generated the interrupt. These transactions undefined when AS is asserted. The R/W line indicates read (high), and the B/W line indicates word (low). The identifier is sampled by the CPU on the AD lines at the falling clock edge before DS is raised high.

are generated automatically by the CPU when an

interrupt request is detected.

Interrupt Acknowledge transactions are five cycles long at a mimimum, with two automatic wait cycles (Figure 13-12). The wait cycles are used to give the interrupt priority daisy chain (or other priority resolution devices) time to settle before the identifier is read. Additional automatic wait states can be generated by programming the Bus

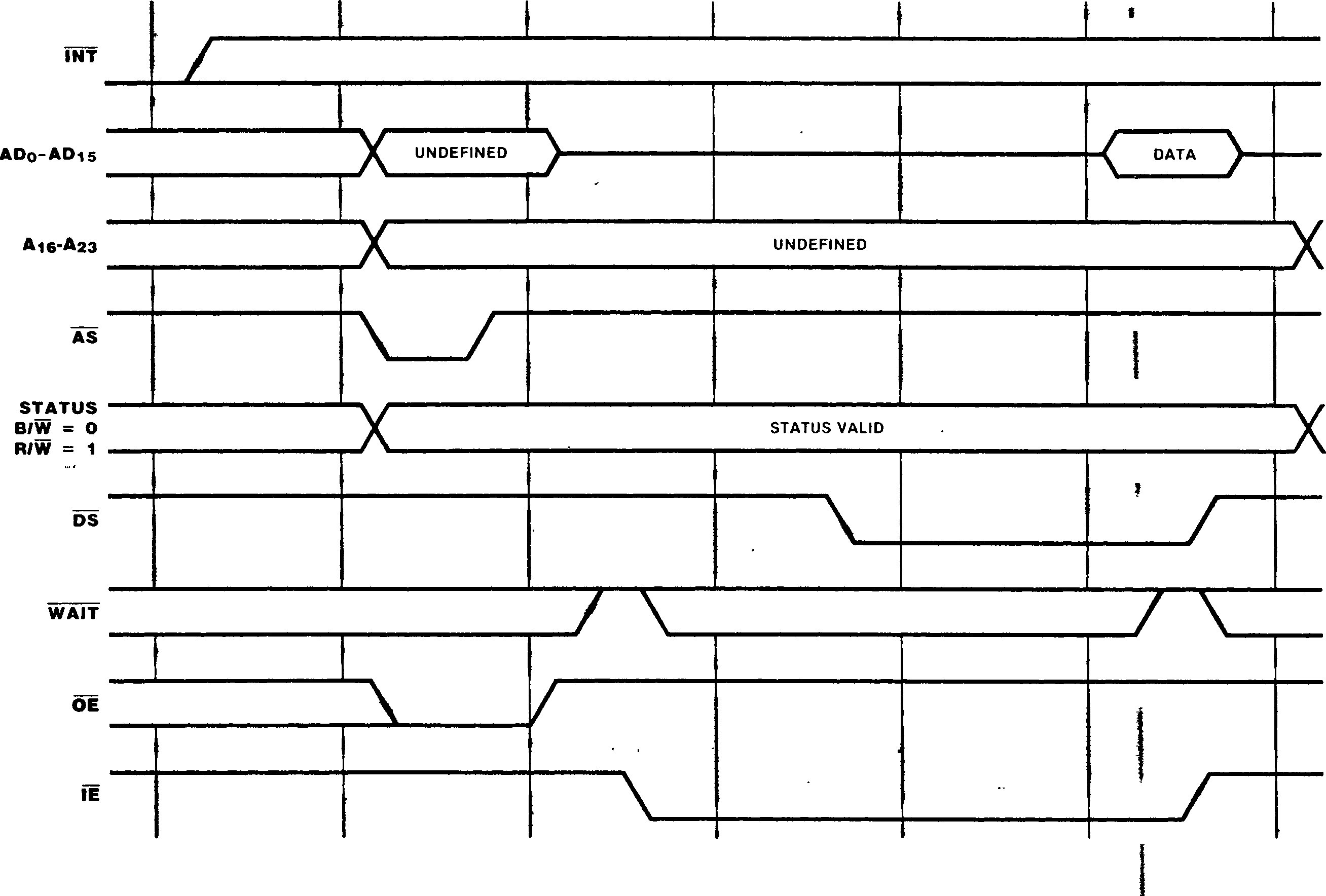
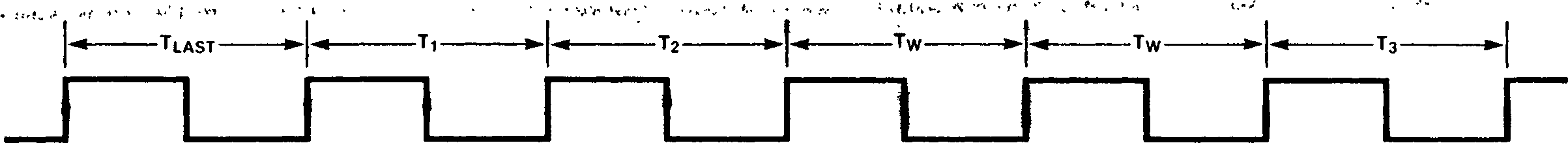
Timing and Control register.

The STj-STg status lines indicate the type of interrupt being acknowledged. No address is generated, so the contents of the address bus are

There are two places where the WAIT line is sampled and, thus, where wait states can be inserted by external circuitry. The first, during T2, serves to delay the falling edge of DS to allow the daisy chain a longer time to settle; the second, during T3, serves to delay the point at which the identifier is read. Software­generated wait states can also be added at either time via programming of the DC and I/O fields in the Bus Timing and Control register. As always, software-generated wait states are inserted into the transaction before the external WAIT signal is sampled. :

CLK

Figure 13-12. Interrupt Acknowledge Timing >



) j

* + 1. **Extended Processing Unit (EPU) Transactions**

Z280 MPUs in the Z-BUS configuration can operate in conjunction with one or more Extended Processing Units (EPUs). Functioning as a coprocessor, the EPU monitors the status and timing signals output by the CPU so that it knows when to participate in a transaction. The Z280 MPU provides the address, status, and timing signals while the EPU supplies or captures data. Each of the four possible types of transactions that require EPU participation are signalled by the Z280 MPU STj-SIg outputs. CPU and EPU interaction is fully described in section 10.5.

* + - 1. **EPU Instruction Fetch**

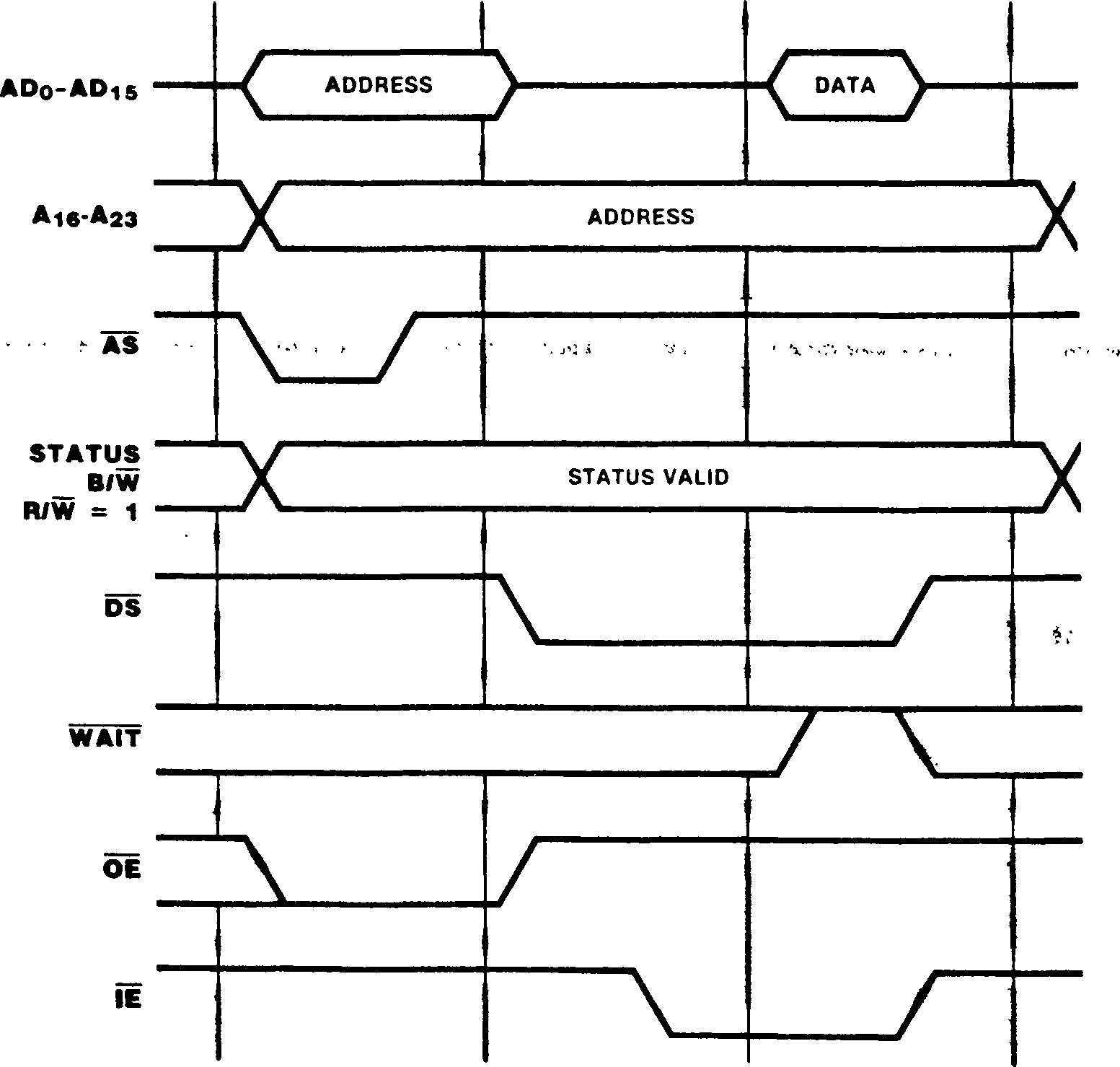
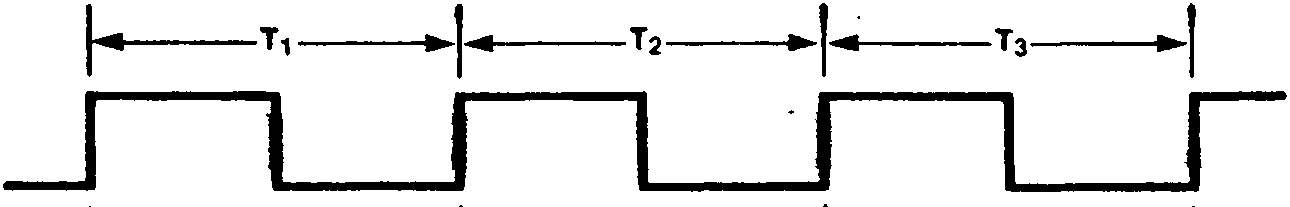
When the Z280 CPU encounters an extended instruction, the state of the EPU Enable bit in the Trap Control register is examined. If the EPU Enable bit is zero, the Z280 generates an Extended Instruction trap. If the EPU Enable bit is set to 1, then the four-byte EPU template is

fetched from memory using memor> transactions and captured by both the CPU and EPU. The ”1101" status code on the SI3-Sfg lines indicates when the first word of the template is fetched, and the ”1100” status code indicates fetches of the subsequent template word or words, depending on the alignment. The CPU fetches the template from external memory using two word transactions if the template is aligned (that is, starts on an even address) or a byte transaction followed by two word transactions if the template is unaligned. The opcode and addressing mode portion of the extended instruction may be executed from cache, but the template will always be fetched from external memory. ।

In a multiple EPU system, the EPU that is to participate in the execution of an extended instruction is selected implicitly by an identification code in the instruction template. Thus, there is no indication on the bus as to which EPU is cooperating with the CPU at any given time.

CLK

Figure 13-13. Memory to EPU Timing



* + - 1. Memory-EPU Transactions

**If** an extended instruction involves a read or write to memory, then the transfers of data between memory and the EPU are the next non-refresh transactions performed by the CPU following the fetch of the template. The timing of memory-EPU data transfers is shown in Figures 13-13 and 13-14. The EPU must supply the data during write operations (R/W = low) or capture the data during read operations (R/W = high), just as if it were part of the CPU. In both cases, the CPU 3-states its AD lines while data is being transferred (DS = low). EPU reads from memory are three cycles long unless extended by wait states.

EPU writes to memory are six cycles long unless extended by wait states.

* + - 1. **EPU-CPU Transactions**

If an extended instruction involves a transfer from the EPU to the Z280 CPU, the next non-refresh transaction following the fetch of the template is the EPU-to-CPU data transfer (Figure 13-15).

cycles long, unless extended by wait states.

Although AS is asserted, no address is generated ' and the contents of the address bus are

undefined. Ihe ”1110'’ status code on the STj-STq lines indicate an EPU-to-CPU transaction.

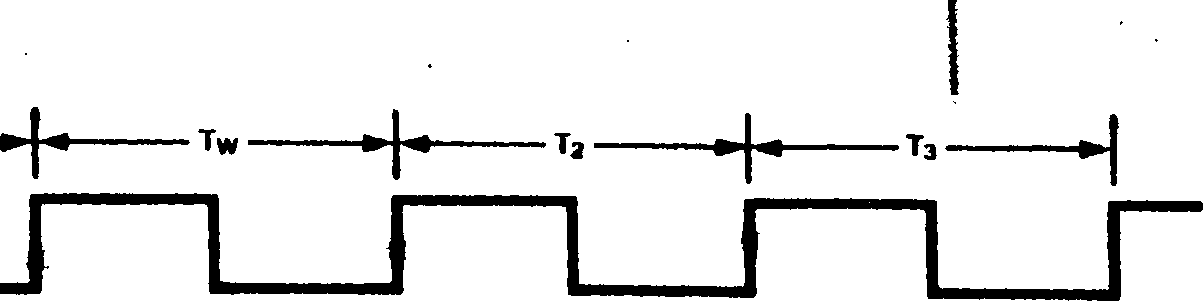
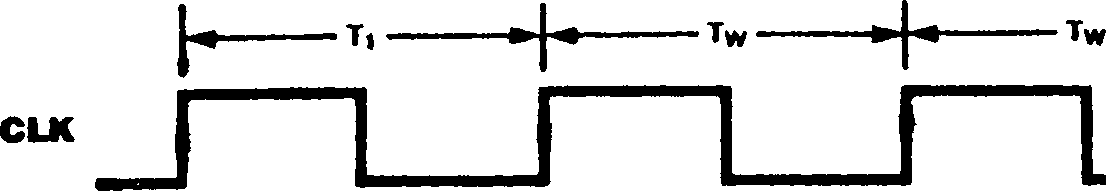
* + - 1. PAUSE Timing

The PAUSE signal is used to synchronize CPU-EPU activity in the case of overlapping extended instructions. The CPU samples the PAUSE signal within one bus clock period of the completion of

the fetch of an extended instruction’s template

(Figure 13-16). If PAUSE is active when sampled,

the CPU enters an idle state wherein all CPU activity is suspended. While in this idle state, the CPU samples the PAUSE input each processor clock cycle until PAUSE is deasserted. The CPU then resumes operation at the point at which it was suspended, either by executing the data transactions associated with the extended instruction (in the case of an extended instruction specifying an EPU-memory or CPU-EPU data transfer) or by starting the fetch of the



instruction specifying an internal EPU operation).

next instruction (in the case of an extended

EPU-to-CPU transactions have the same form as I/O read transactions and thus are four clock

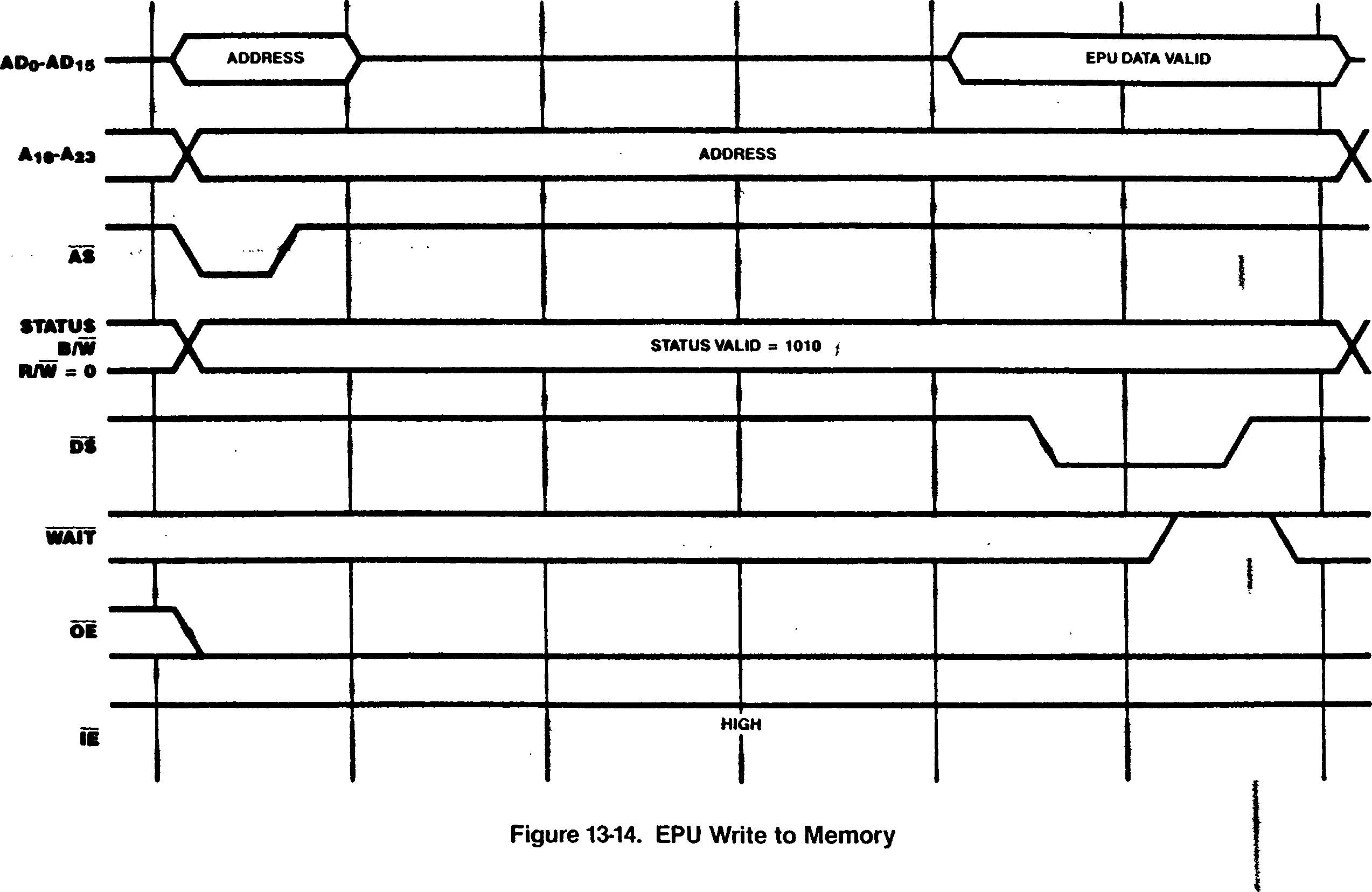


Figure 13-15. EPU to CPU Timing

I I

I

I

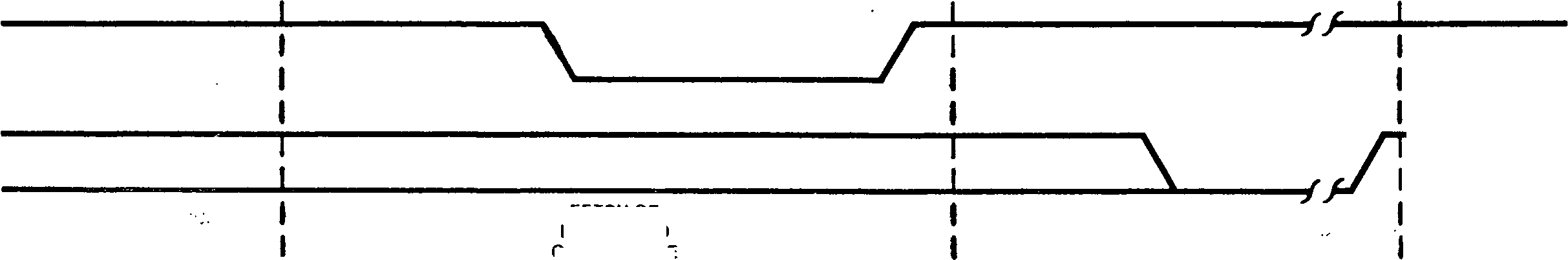
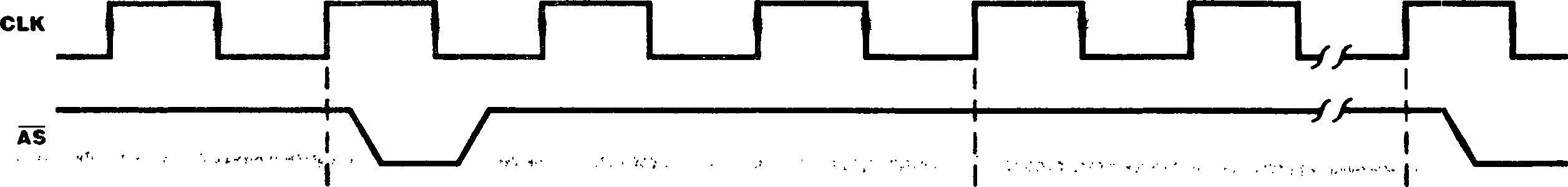
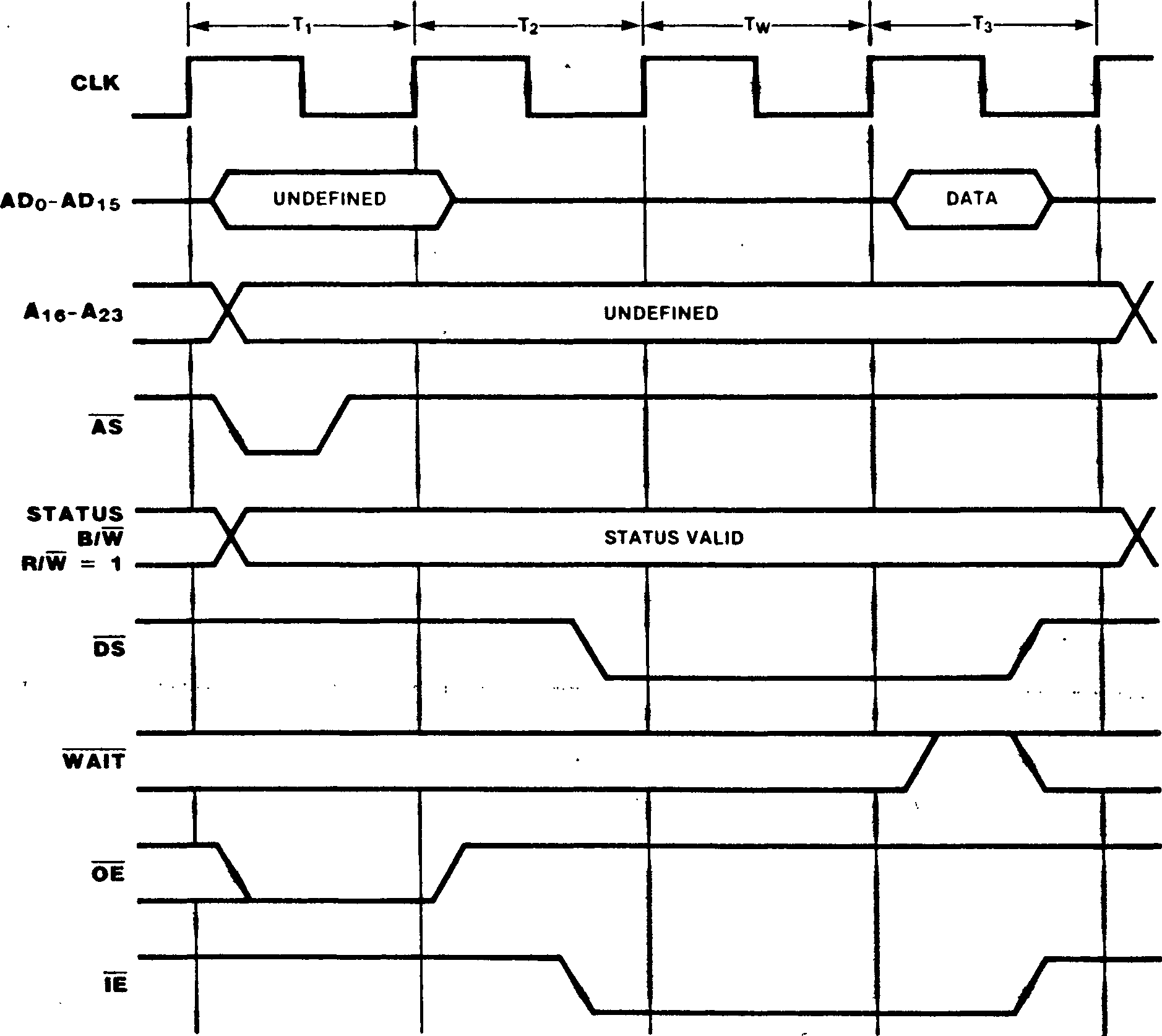
DS

PAUSE

FETCH OF LAST WORD OF TEMPLATE

NEXT BUS TRANSACTION

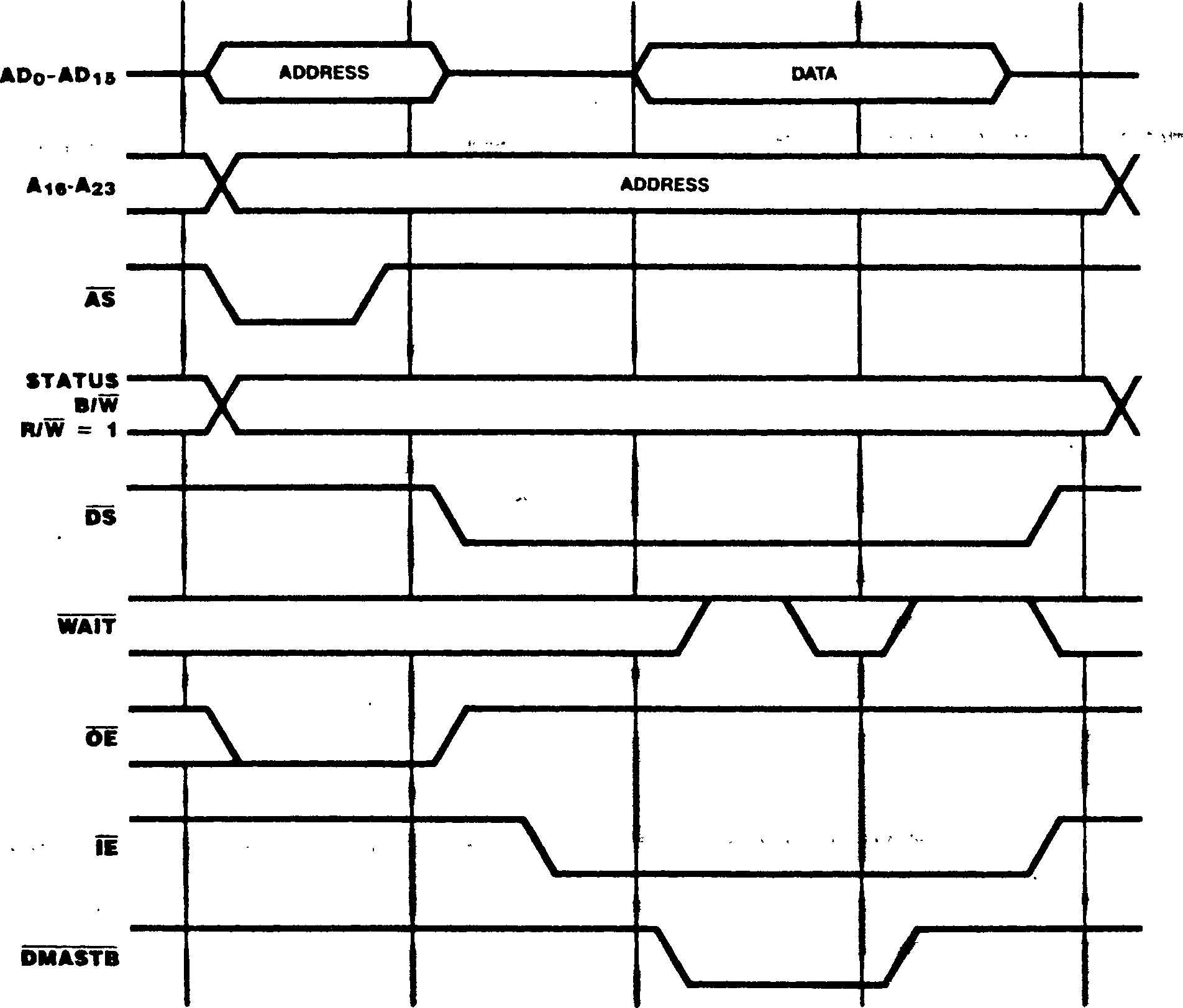
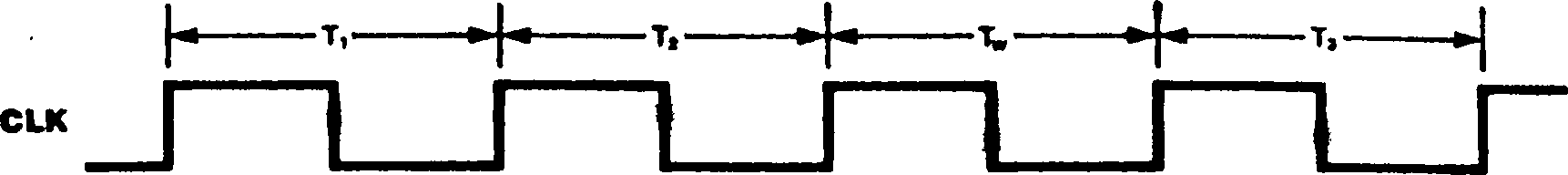
Figure 13-16. PAUSE Timing



On-chip DMA channels 0 and 1 can transfer data between memory and peripheral devices using flyby type transfers; external DMA controllers in Z280 MPU systems (such as the Z8016 DIG) may also have this capability. The timing of flyby transactions is similar to memory transaction timing, with the exception that the DMA Strobe (DMASTB) signal is activated; the DMASTB signal is used to select the participating I/O device that must capture or flyby transactions controlled by the on-chip DMA channels always include one automatic wait state (Figures 13-17 and 15-18). As with all memory transactions, other hardware- and software­generated wait states can be added to the transaction. The external WAIT signal is sampled at two different times: during the automatic wait state and during T3.

supply the data during the memory access.

Figure 13-17. On-Chip DMA Channel Flyby Memory Read Transaction



lor flyby transactions that read from memory and write to a peripheral (Figure 13-17), DMASfB is asserted during the automatic wait state and any subsequent wait states due to an active WAIT signal. Thus, if the WAI I input is asserted during the automatic wait state, the additional wait states extend the width of the DMASfB pulse. Wait states added via the assertion of WAIT during T3 (after DMASfB is deasserted) stretch the DS

For flyby transactions that read from a peripheral and write to memory (Figure 13-18), DMASfB is asserted at the beginning of T2 and remains asserted until the second half of T3. The DS signal is asserted only during the automatic wait state. Wait states added via the essertion of

WAIT stretch the DMASfB signal without affecting DS.

signal without affecting DMASTB.

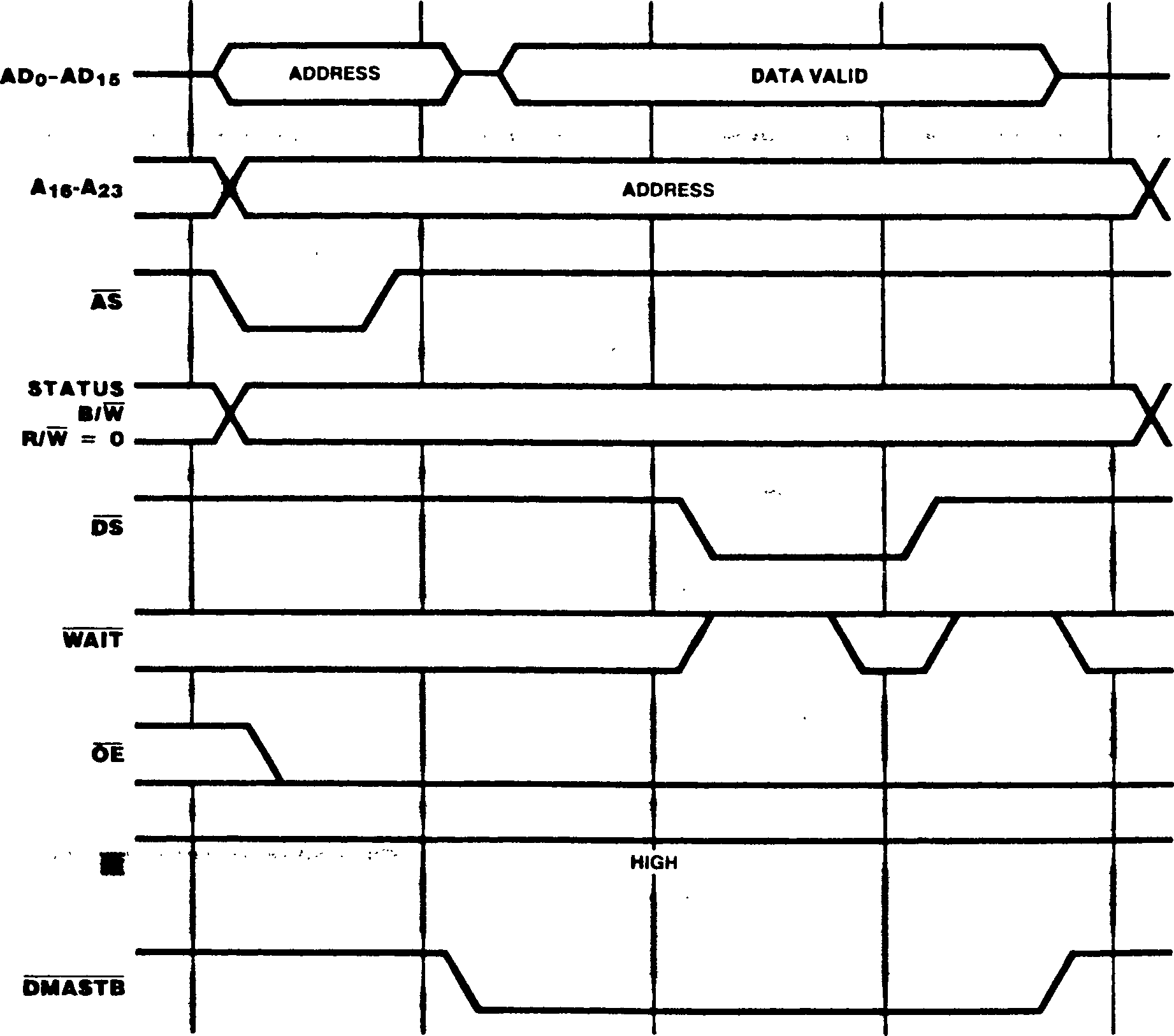
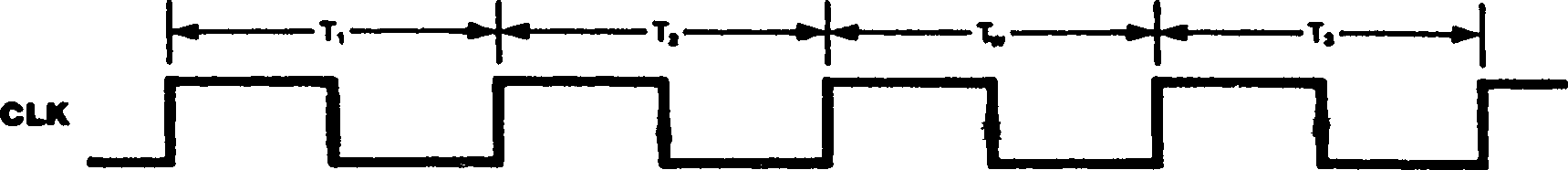


Figure 13-18. On-Chip DMA Channel Flyby Memory Write Transaction

13.6 REQUESTS

The Z280 MPU supports three types of request signals: interrupt requests, local bus requests, and global bus requests. A request is answered according to its type. Interrupt requests are generated by peripheral devices; the Z280 MPU responds with an Interrupt Acknowledge transaction. Local bus requests are initiated by an external potential bus master; the Z280 MPU responds by relinquishing the bus and generating an active Bus Acknowledge signal. Global bus requests are generated by the Z280 CPU or an on-chip DMA channel to access a global bus; the Z280 MPU receives a Global Bus Acknowledge signal in response to the request.

* + 1. Interrupt Requests

Ihe Z280 CPU supports two types of interrupts, maskable and nonmaskable (NMI). The interrupt request line from a device capable of generating interrupts can be tied to the Z280 MPU’s NMI or maskable interrupt request inputs; several devices can be connected to one interrupt request input, with interrupt priorities established via external logic or a priority daisy chain.

Nonmaskable interrupt requests are edge-triggered, but maskable interrupts are level-triggered. Any high-to-low transition on the NMI input is asynchronously edge-detected, and an internal NMI latch is set. At the beginning of the last clock cycle during execution of an instruction, the maskable interrupt inputs are sampled along with the state of the internal NMI latch. If an interrupt is detected, and that interrupt is enabled in the Master Status register, interrupt » processing proceeds in accordance with the current interrupt mode, as described in Chapter 6.

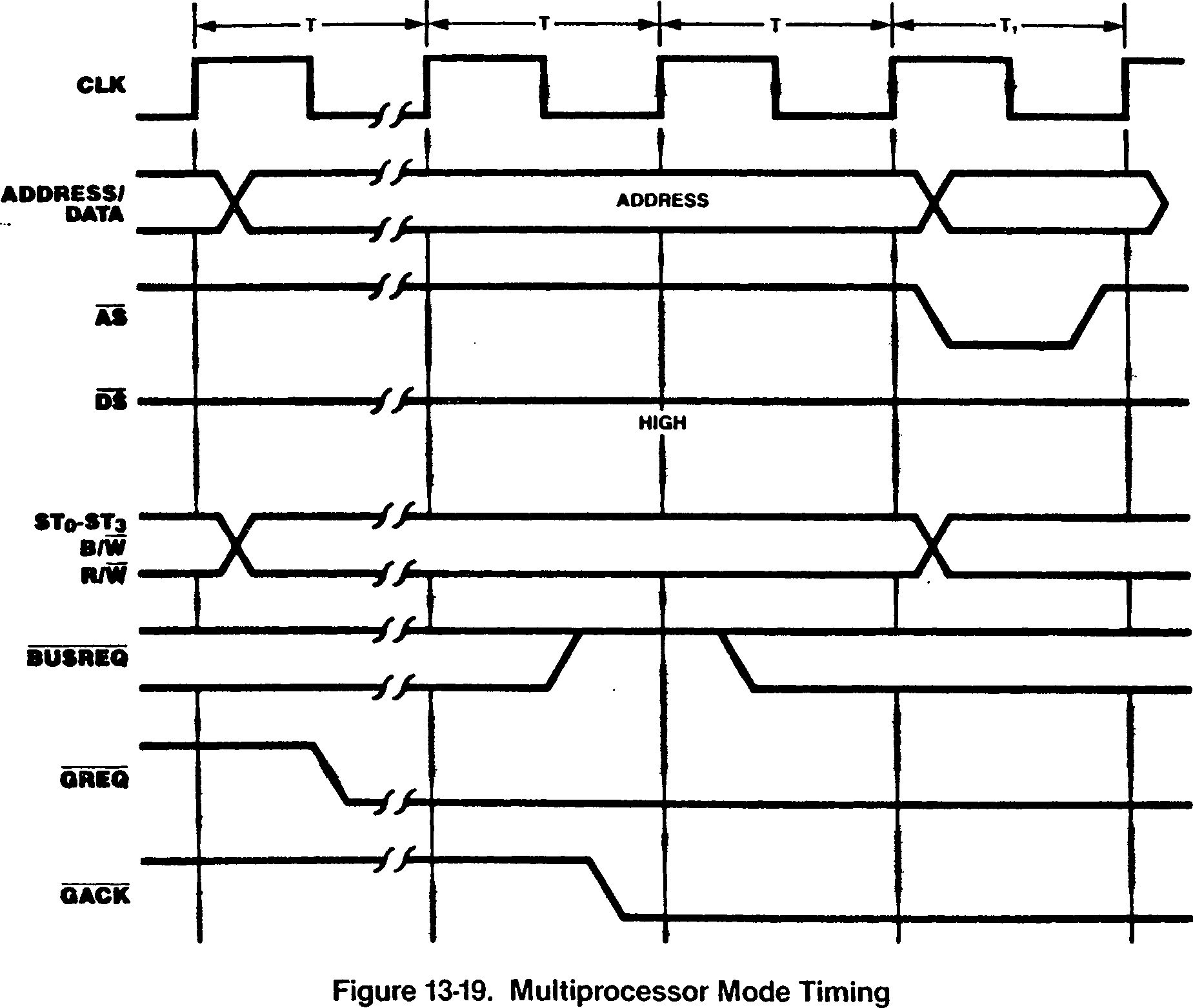
* + 1. Local Bus Requests

To generate transactions on the bus, a potential bus master (such as a DMA controller) must gain control of the bus by making a bus request. A bus request is initiated by pulling BUSREQ low; the Z280 MPU responds by 3-stating its address, data, bus control, and bus status outputs and asserting an active BUSACK, as described in section 10.2. The CPU regains control of the bus after BUSREQ rises. The on-chip DMA channels have higher priority than external devices requesting the bus via BUSREQ.

* + 1. **Global Bus Requests**

If the multiprocessor mode is specified in the Bus Timing and Initialization registET, then the contents of the Local Address register determine the range of memory addresses dedicated to the shared global bus. Before accessing an address on the global bus, the Z280 MPU must issue a Global Bus Request (GREQ) and receive an active Global Bus Acknowledge (GACK) signal, as described in Section 10.3.

Figure 13-19 illustrates the timing of the global bus request/acknowledge sequence. When the Z280 MPU needs to access a location on the global bus, GREQ is asserted in order to request use of the global bus. GACK is then sampled on each successive rising edge of the clock; when GACK becomes active (and if BUSREQ is not asserted), the memory transaction proceeds as described in section 13.5.1. GREQ is deasserted in the bus cycle immediately following the end of the memory transaction (except when executing the Test and Set instruction, where both the memory read and write operations are executed before deasserting GREQ).



Appendix A.

Z80/Z280 Compatibility

The Z280 MPU architecture is an upward-compatible extension of the Z80 CPU architecture. This compatibility extends to the instruction set, register architecture, interrupt structure, and bus structure of the Z280 MPU and Z80 CPU.

The Z80 CPU’s instruction set is a subset of the Z280 MPU’s instruction set. Thus, the Z280 MPU is completely binary-compatible with Z80 code. However, since some Z80 instructions, such as HALT, are privileged instructions in the Z280 MPU, complete compatibility is achieved only when the Z280 MPU is executing in the system mode. All Z80 software will execute successfully on a Z280 MPU running in system mode, provided that the software contains no timing dependencies, does not modify itself, and does not use any of the Z80\*s reserved instruction encodings.

Since the Z280 MPU is binary-code compatible with the Z80 CPU, the Z280 MPU’s general-purpose register set is the same as the Z80 CPU’s, with the exception of the Stack Pointer. The Z280 MPU contains both a System Stack Pointer and a User Stack Pointer, whereas the Z80 CPU has only one Stack Pointer register. In the Z80 CPU, the R register is used to indicate the next refresh address; in the Z280 MPU, the R register is not involved with the refresh logic and may be used by the programmer as a general-purpose storage register.

The Z280 MPU’s interrupt structure is also an upward-compatible extension of the Z80 CPU’s. The Z280 MPU supports all three interrupt modes found on the Z80 CPU, as well as a fourth interrupt mode new to the Z280 MPU.

The Z80 Bus configurations of the Z280 MPU are also bus-compatible with the Z80 CPU, generating the same RD, WR, IORQ, and MREQ bus control and status signals. However, M1 is asserted during every instruction fetch and interrupt acknowledge cycle in the Z80 CPU; for the Z280 MPU, Ml is asserted only during the special RETI bus transaction and interrupt acknowledge cycles. The Z8400 family of peripherals interface directly to

both Z80 CPUs and Z80 bus configuration of the Z280 MPUs.

Following a reset, the Z280 MPU takes on a configuration that is fully compatible with Z80 code. The Memory Management Unit is disabled, meaning that the 16-bit logical addresses from the Z280 CPU are routed directly to the 16 least significant address pins on the external bus. The User/System bit in the Master Status register specifies system-mode operation, allowing execution of privileged instructions and enabling the System Stack Pointer. The I/O Page register is cleared to all Os and Interrupt Mode 0 is selected. The Trap Control register is cleared to all zeros, disabling System Stack Overflow Warning traps and designating that I/O instructions are not privileged. All Z80 instructions can be successfully executed (and may execute from the on-chip memory that is enabled as an instruction- only cache upon reset). The Z280 MPU will remain in a Z80-compatible configuration as long as Z80 code is executed, since the Load Control instruction that acts on the Z280 MPU’s control registers is not part of the Z80 instruction set.

The software routine shown below can be used to determine if code is executing on a Z80 CPU or Z280 MPU. This facilitates development of programs that can execute on either processor, but contain special routines invoked only when executing on a Z280 MPU and, therefore, allowing use of Z280 MPU features not available on the Z80 CPU. The routine differentiates the Z80 CPU from the Z280 MPU by executing the instruction with machine code CB37^|. This instruction code is reserved in the Z80 CPU, and results in logically shifting the A register one bit to the left while shifting a 1 into the least significant bit. For the Z280 MPU, CB37^ is the code for the Test and Set instruction. If the A register holds a 40^ before executing this instruction code, the A register holds an 81^ and the Sign flag is set . to 1 after executing the instruction on a Z80 CPU;

the A register holds an FF^ and the Sign flag is cleared to 0 after executing the instruction on a Z280 MPU.

**Code to Distinguish Execution on a Z80 CPU and Z280 MPU**

; This instruction sequence exploits the difference when executing the CB37j\_| j machine code on the Z80 CPU and Z280 MPU, to allow a program to determine which : processor it is executing on. This instruction sets the S flag on the Z80 CPU ; and clears the S flag on the Z280 MPU. The A and F registers are used by the ; routine.

LD A,40^| ' ; Initialize the operand. ’

DEFB 0СВ|\_|,037ц ; This instruction will set the S flag on the

- ; Z80 CPU and clear the S flag on the Z280 MPU.

JP M,Z80 ; Now test the flag and jump,

or JP P,Z280

ft

Appendix В.

Z280 MPU Instruction Formats

Four formats are used to generate the machine­language bit encodings for the Z280 MPU instructions. Three formats are used for instructions that are executed solely by the Z280 CPU. (These same three formats are used for Z80 CPU instruction encoding.) A fourth format is dedicated to instructions that involve Extended Processing Units (EPUs).

encoding. In Formats 2 and 4, the opcode escape

byte immediately preceeds the opcode byte itself.

The bit encodings of the Z280 MPU instructions are partitioned into bytes. Every instruction encoding contains one byte dedicated to specifying the type of operation to be performed; this byte is referred to as the instruction’s operation code (opcode). Besides specifying a particular operation, opcodes typically include bit encodings specifying the operand addressing mode for the instruction and identifying any general-purpose registers used by the instruction. Along with the opcode, instruction encodings may include bytes that contain an address, displacement, and/or immediate value used by the instruction, and special bytes called ’’escape codes” that determine the meaning of the opcode itself.

In Format 3, a 1-byte displacement may be between the opcode escape byte and opcode itself. Opcode escape bytes are used to distinguish between two different instructions with the same opcode byte, thereby allowing more than 256 unique instructions. For example, the 01^ opcode, when alone, specifies a form of the Load Register Word instruction; when preceded by the CB|g escape byte, the opcode 01^ specifies a Rotate Left Circular instruction.

By themselves, one byte opcodes would allow the encoding of only 256 unigue instructions. Therefore, special ’’escape codes” that precede the opcode in the instruction encoding are used to expand the number of possible instructions. There are two types of escape codes: addressing mode escape codes and opcode escape codes. Escape codes are one byte in length.

Three of the instruction formats are differentiated by the opcode escape value used; the fourth format is for instructions that include an EPU template. Format 1 is for instructions without an opcode escape byte, Format 2 is for instructions whose opcode escape byte has the value ED^, and Format 3 is for instructions whose opcode escape byte has the value ЕВц\* Instructions that support EPUs use Format A and always have the opcode escape byte with value EDp| as the first byte of the instruction

Addressing mode escape codes are used to determine the type of encoding for the addressing mode field within an instruction’s opcode, and can be used in instructions with and without opcode escape values. An addressing mode escape byte can have the value DD|^ or FD^. The addressing mode escape byte, if present, is always the first byte of the instruction’s machine code, and is immedi­ately followed by either the opcode (Format 1) or the opcode escape byte (Formats 2 and 3). For example, the 79^ opcode, when alone, specifies a Load Accumulator instruction using Register addressing for the source operand; when preceded by the DD|\_| escape byte, the opcode 79^ specifies a Load Accumulator instruction using Base Index addressing for the source operand.

The four instruction formats are shown in Tables B-1 through B-4. Within each format, several different configurations are possible, depending on whether the instruction involves addressing mode escape bytes, addresses, displacements, or immediate data. In Tables B-1 through B-4, the symbol ”A.esc” is used to indicate the presence of an addressing mode escape byte, ”disp.” is an abbreviation for displacement, ”addr.” is an abbreviation for address, and ’’temp.” is an abbreviation for template. Templates in EPU instructions are four-byte fields that include the bit encodings that specify EPU operation.

**Table B-1. Format 11nstruction Encodings**

Example Instruction

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Instruction Format** | **Assembly** | **Machine Code (Hex)** |
|  | opcode | LD A,C | 79 |
|  | opcode 2-byte address | LD A,(addr) | ЗА addr(low) addr(high) |
|  | opcode Tbyte displacement | DJNZ addr | 10 disp |
|  | opcode immediate | LD E,n | lEn |
| A.esc | opcode . | LDA,(HL + IX) | DD79 |
| A.esc | opcode 2-byte address | LD IX,(addr) | DD 2A addr(low) addr(high) |
| A.esc | opcode Tbyte displacement | LD A,(IX + d) | DD 7E disp |
| A.esc | opcode 2-byte displacement | LD A,(IX + dd) | FD 79 d(low) d(high) |
| A.esc | opcode immediate | LD IX,nn | DD 21 n(low) n(high) |
| A.esc | opcode 2-byte address | immediate LD (addr),n | DD 3E addr(low) addr(high) n . |
| A.esc | opcode Tbyte displacement | immediate LD (IY + d),n | FD 36 d n . |
| A.esc | opcode 2-byte displacement | immediate LD<addr>,n | FD 06 disp(low) disp(high) n |

**Table B-2. Format 2 Instruction Encodings**

**Example Instruction**

**Instruction Format**

**Assembly**

**Machine Code (Hex)**

ED opcode ED opcode ED opcode ED opcode A.esc ED opcode

A.esc ED opcode

A.esc ED opcode

A.esc ED opcode

A.esc ED opcode

immediate

2-byte address

2-byte displacement

2-byte address

1 -byte displacement 2-byte displacement

2-byte immediate

MULT A,В

SC nn

LD BC.(addr)

LD (HL + dd),A MULTA.IY . MULT A,(addr) MULT A,(IY + d) LD IX,(IY + dd) MULTUW HL,nn

ED CO

ED 71 n(low) n(high)

ED 4B addr(low) addr(high)

ED 3B d(low) d(high)

FD ED E8

DD ED F8 addr(low) addr(high)

FD ED F8 d

DD ED 34 d(low) d(high)

FD ED F3 n(low) n(high)

**Table B-3. Format 3 Instruction Encodings**

**Example Instruction  
. ..... . Instruction Format ' Assembly ' Machine Code (Hex)**

CB opcode RLC(HL) CB06

A.esc CB 1-byte displacement opcode RCL(IX+d) DDCBdO6

**Table B-4. Format 4 Instruction Encodings '**

**' Example Instruction**

**Instruction Format Assembly Machine Code (Hex)**

ED opcode 4-byte template EPU \*- (HL) ED A6 tempi temp2 temp3 temp4

ED opcode 2-byte displacement 4-byte template EPU\*-(HL + dd) ED BC d(low) d(high) tempi temp2 temp3 temp4

ED opcode 2-byte address 4-bytetemplate EPU\*-(addr) ED A7 addr(low) addr(high) tempi ternp2 temp3 temp4

B-2

SOURCE CODE

OBJECT CODE

Appendix C.

Instructions in Alphabetic Order

SOURCE CODE

OBJECT CODE

ADC A,(HL) ADC A,(HL+IX) ADC A,(HL+IY) ADC A,(HL+1122H) ADC A,(IX+IY) ADC A,(IX+55H) ADC A,(IX+H22H) ADC A,(IY+55H) ADC A,(IY+1122H) ADC АДРС+1122Н) ADC A,(SP+U22H) ADC A,(3344H) ADC .■ A,A ■» ADC A,В ADC A,C ADC A,D ADC A,E ADC A,H ADC A,IXH ADC A,IXL ADC A,IYH ADC A,IYL ADC A,L

ADC A.66H ADC HL,BC ADC HL,DE

ADC HL,HL

**ADC HL,SP ADC IX,BC**

[**ADC IX,DE**](#bookmark158)

[**ADC IX,IX**](#bookmark155)

**ADC IX,SP**

ADC IY,BC ADC IY,DE ADC IY.IY ADC IY,SP ADD A,(HL) ADD A,(HL+IX) ADD A,(HL+IY) ADD A,(HL+1122H) ADD A,(IX+IY) ADD A,(IX+55H) ADD A,(IX+1122H) ADD A,(IY+55H) ADD A,(IY+1122H) ADD A,(PC+1122H) ADD A,(SP+H22H) ADD A,(3344H) ADD A,A ADD A,В ADD A,C

8E DD89 DD8A FD8B2211 DD8B DD8E55 FD892211 FD8E55 FD8A2211 FD882211 DD882211 DD8F4433 8F

88 89

8A 8B

8C DD8C DD8D FD8C FD8D

8D CE66 ED4A ED5A ED6A ED7A DDED4A DDED5A DDED6A DDED7A FDED4A FDED5A FDED6A FDED7A 86

DD81 DD82 FD832211 DD83 DD8655 FD812211 FD8655 FD822211 FD802211 DD802211 DD874433 87 80 81

ADD A,D ADD A,E ADD A,H ADD A,IXH ADD A,IXL ADD A,IYH ADD A.IYL ADD A,L ADD A.66H ADD HL,A ADD HL,BC ADD HL,DE ADD HL,HL \* • '

ADD HL.SP ADD IX,A ADD IX,BC ADD IX,DE ADD IX,IX ADD IX,SP ADD IY,A ADD IY,BC ADD IY.DE ADD IY,IY ADD IY,SP ADDW HL,(HL) ADDW HL,(IX+1122H) ADDW HL,(IY+1122H) ADDW HL,(PC+1122H) ADDW HL,(3344H) ADDW HL.BC ADDW HL,DE ADDW HL,HL ADDW HL,IX ADDW HL,IY ADDW HL,SP ADDW HL.3344H AND A,(HL)

AND A,(HL+IX) AND A,(HL+IY) AND A,(HL+1122H) AND A,(IX+IY) AND АД1Х+55Н) AND A,(IX+1122H) AND A,(IY+55H) AND A,(IY+U22H) AND A,(PC+1122H) AND A,(SP+1122H) AND АД3344Н) AND A,A AND A,В AND A,C

82 83 84 DD84 DD85 FD84 FD85 85 C666 ED6D 09 19 29 '

39 DDED6D DD09 DD19 DD29 DD39 FDED6D FD09 FD19 FD29 FD39 DDEDC6

FDEDC62211 FDEDD62211 DDEDF62211 DDEDD64433 EDC6 EDD6 EDE6 DDEDE6 FDEDE6 EDF6 FDEDF64433 A6 DDA1 DDA2 FDA322U DDA3 DDA655 FDA12211 FDA655 FDA22211 FDA02211 DDA02211 DDA74433 A7 AO Al

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SOURCE CODE** | | **OBJECT CODE** | **SOURCE CODE** | | **OBJECT CODE** |
| **AND** | **A.D** | **A2** | **BIT** | **5Д1Х+55Н)** | **DDCB556E** |
| **AND** | **A,E** | **A3** | **BIT** | **5,(IY+55H)** | **FDCB556E** |
| **AND** | **A,H** | **A4** | **BIT** | **5,A** | **CB€F** |
| **AND** | **A.IXH** | **DDA4** | **BIT** | **5,В** | **CB68** |
| **AND** | **A.IXL** | **DDA5** | **BIT** | **5,C** | **CB69** |
| **AND** | **A.IYH** | **FDA4** | **BIT** | **5.D** | **CB6A** |
| **AND** | **A.IYL** | **FDA5** | **BIT** | **5,E** | **CB6B** |
| **AND** | **A.L** | **A5** | **BIT** | **5,H** | **CB6C** |
| **AND** | **A.66H** | **E666** | **BIT** | **5,L** | **CB6D** |
| **BIT** | **0,(HL)** | **CB46** | **BIT** | **6,(HL)** | **CB76** |
| **BIT** | **0Д1Х+55Н)** | **DDCB5546** | **BIT** | **6,(IX+55H)** | **DDCB5576** |
| **BIT** | **0,(IY+55H)** | **FDCB5546** | **BIT** | **6,(IY+55H)** | **FDCB5576** |
| **BIT** | **0,A** | **CB47** | **BIT** | **6,A** | **CB77** |
| **BIT** | **0,B** | **CB40** | **BIT** | **6,B** | **CB70** |
| **BIT** | **0,C** | **CB41** | **BIT** | **6,C** | **CB71** |
| **BIT** | **0,D** | **CB42** | **BIT** | **6,D** | **CB72** |
| **BIT** | **O.E** | **CB43** | **BIT** | **6,E** | **CB73** |
| **BIT** | **0,H** | **CB44** | **BIT .** | **6,H** | **CB74** |
| **BIT** | **0,L** | **CB45** | **BIT** | **6,L** | **CB75** |
| **BIT** | **l.(HL)** | **CB4E** | **BIT** | **7,(HL)** | **CB7E** |
| **BIT** | **Ц1Х+55Н)** | **DDCB554E** | **BIT** | **7Д1Х+55Н)** | **DDCB557E** |

BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT BIT

BIT BIT BIT BIT BIT

1,(IY+55H)

2,(HL) 2.0X+55H) 2,(IY+55H) 2,A

... 2,D ...

2,H

3,(HL) 3,(IX+55H) 3,(IY+55H) 3,A

3.H

4,(HL) 4,(IX+55H) 4,(IY+55H) 4.A

5,(HL)

FDCB554E

CB4F

CB48

CB49

CB4A

CB4B

CB4C

CB4D

CB56 DDCB5556 FDCB5556

CB57 CB50 CB51

CB52

CB53

CB54 CB55 CB5E

i.

DDCB555E FDCB555E CB5F CB58 CB59 CB5A CB5B CB5C CB5D CB66

DDCB5566 FDCB5566 CB67 CB60 CB61 CB62 CB63 CB64 CB65 CB6E

BIT BIT BIT BIT BIT BIT BIT BIT CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL CALL

7,(IY+55H)

7,D

(HL) (PC+1122H) C,(HL) C,(PC+1122H) C.3344H M.(HL) M,(PC+1122H)

M.3344H NC,(HL)

l\IC,(PC+1122H) NC.3344H HZ,(HL) NZ,(PC+1122H) NZ.3344H P.(HL) РДРС+1122Н) P.3344H PE,(HL) РЕДРС+1122Н) PE.3344H PO,(HL) PO,(PC+1122H) PO.3344H Z,(HL) Z,(PC+1122H) Z.3344H 3344H

A,(HL) A,(HL+IX) A,(HL+IY)

FDCB557E CB7F CB78 CB79 CB7A CB7B CB7C CB7D DDCD FDCD2211 DDDC FDDC2211 DC4433 DDFC FDFC2211 FC4433 DDD4 FDD42211 D44433 DDC4 FDC42211 C44433 DDF4 FDF42211 F44433 □DEC FDEC2211 EC4433

DDE4 FDE42211 E44433 DDCC FDCC2211 CC4433 CD4433 3F BE DDB9 DDBA

OBJECT CODE FDBB2211 DDBB DDBE55 FDB92211 FDBE55 FDBA2211 FDB82211 DDB82211 DDBF4433 BF B8 B9 BA BB BC DDBC DDBD FDBC FDBD BD FE66

EDA9 v • EDB9 EDAI EDB1 2F DDEDC7 FDEDC72211 FDEDD72211 DDEDF72211 DDEDD74433 EDC7 EDD7 EDE7 DDEDE7 FDEDE7 EDF7 FDEDF74433 27 35 DDOD DD15 FD1D2211 DD1D DD3555 FD0D2211 FD3555 FD152211 FD052211 DD052211 DD3D4433 3D 05 OB 0D 15 IB ID 25 2B

OBJECT CODE

DD2B DD25 DD2D FD2B FD25 FD2D 2D 3B DDOB FD0B2211 FD1B2211 DD3B2211 DD1B4433

OB IB 2B DD2B FD2B 3B F3 ED7766 EDF4 DDEDCC DDEDD4 FDEDDC2211 DDEDDC DDEDF455 FDEDCC2211 FDEDF455 FDEDD42211 FDEDC42211 DDEDC42211 DDEDFC4433 EDFC

EDC4 EDCC EDD4 EDDC EDE4 DDEDE4 DDEDEC FDEDE4 FDEDEC EDEC FOEDFC66 EDF5 DDEDCD DDEDD5 FDEDDD2211 DDEDDD DDEDF555 FDEDCD2211 FDEDF555 FDEDD52211 FDEDC52211 DDEDC52211 DDEDFD4433 EDFD

EDC5 EDCD

SOURCE CODE CP A,(HL+1122H)

CP A,(IX+IY)

CP A,(IX+55H)

СР A,(IX+1122H)

CP A,(IY+55H)

СР A,(IY+1122H)

СР АДРС+1122Н)

СР A,(SP+1122H)

СР А,(3344Н)

СР А,А

СР А,В

СР А,С

**СР A.D**

**СР А,Е**

**СР А,Н**

**СР A.IXH**

**СР A.IXL**

**СР A.IYH**

**СР A.IYL**

**СР A.L**

СР А,66Н

CPD "

CPDR CPI CPIR CPL

CPW HL,(HL)

CPW HL,(IX+1122H)

CPW HL,(I Y+1122H)

CPW HL,(PC+1122H) CPW HL,(3344H) CPW HL,BC CPW HL,DE CPW HL,HL CPW HL,IX CPW HL.IY CPW HL,SP CPW HL.3344H DAA DEC (HL) DEC (HL+IX) DEC (HL+IY) DEC (HL+1122H) DEC (IX+IY) DEC (IX+55H) DEC (IX+1122H) DEC (IY+55H) DEC (IY+1122H) DEC (PC+1122H)

DEC (SP+1122H) DEC (3344H) DEC A DEC В DEC BC DEC C DEC t) DEC DE DEC E DEC H DEC HL

SOURCE CODE

DEC IX DEC IXH DEC IXL DEC IY DEC IYH DEC IYL DEC L DEC SP DECW (HL) DECW (IX+1122H) DECW (IY+1122H) DECW (PC+1122H) DECW (3344H) DECW BC DECW DE DECW HL DECW IX DECW IY DECW SP

DI DI 66H

DIV HL,(HL) DIV HL,(HL+IX)

DIV HL,(HL+IY) DIV HL,(HL+1122H) DIV HL,(IX+IY) DIV HL,(IX+55H) DIV HL,(IX+U22H) DIV HL,(IY+55H) DIV HL,(IY+1122H) DIV HL,(PC+1122H) DIV HL,(SP+1122H) DIV HL,(3344H) DIV HL,A DIV HL,В DIV HL,C DIV HL,D DIV HL,E DIV HL,H DIV HL,IXH

DIV HL,IXL DIV HL,IYH

DIV HL,IYL

DIV HL,L DIV HL.66H DIVU HL,(HL) DIVU HL,(HL+IX) DIVU HL,(HL+IY) DIVU HL,(HL+U22H) DIVU HL,(IX+IY) DIVU HL,(IX+55H) DIVU HL,(IX+1122H) DIVU HL,(IY+55H) DIVU HL,(IY+1122H) DIVU HL,(PC+1122H) DIVU HL,(SP+1122H) DIVU HL,(3344H) DIVU HL,A DIVU HL,В DIVU HL,C

OBJECT CODE EDD5 EDDD EDE5 DDEDE5 DDEDED FDEDE5 FDEDED EDED FDEDFD66 DDEDCB FDEDCB2211 FDEDDB2211 DDEDFB2211 DDEDDB4433 EDCB EDDB EDEB DDEDEB FDEDEB EDFB FDEDFB4433 DDEDCA

SOURCE CODE

EX АДРС+1122Н) EX A.(SP+1122H) EX АД3344Н)

EX A,A

EX A.B

**EX A.C**

**EX A.D**

**EX A.E**

**EX A.H**

**EX A.IXH**

**EX A,IXL**

**EX A.IYH**

**EX A.IYL**

**EX A.L**

**EX AF.AF’**

**EX DE,HL**

**EX H.L**

**EX IX,HL \_**

**EX IY.HL**

[**EXTS A**](#bookmark214)

[**EXTS HL**](#bookmark217)

EXX

OBJECT CODE FDED072211 DDED072211 DDED3F4433 ED3F ED07 EDOF ED17 EDIF ED27 DDED27 DDED2F FDED27 FDED2F ED2F 08 EB EDEF DDEB FDEB ED64 ED6C D9

FDEDCA2211 FDEDDA2211 DDEDFA2211 DDEDDA4433 EDCA EDDA EDEA DDEDEA FDEDEA EDFA FDEDFA4433 1075 FB ED7F66 ED97 - ED9F EDA6 ED8C ED94 EDBC2211 ED9C EDAC2211 EDB42211 EDA42211 ED842211 EDA74433 E3 DDE3 FDE3 ED37 DDEDOF DDED17 FDED1F2211 DDED1F DDED3755 FDED0F2211 FDED3755 FDED172211

HALT

IM  
IM  
IM

IM 3

IN IN IN IN IN IN IN IN IN IN IN IN IN

IN IN IN IN IN IN IN IN IN

INC INC INC INC INC INC INC INC INC INC INC

(HL+IX),(C) (HL+IY),(C) (HL+1122H),(C) (IX+IY),(C) (IX+1122H),(C) (IY+1122H),(C) (PC+1122H),(C) (SP+1122H) ,(C) (3344НЦС) A,(C) .....

АД66Н) B,(C) C,(C) D.(C) E.(C) НДС) HL.(C) IXH.(C) IXL,(C) IYH,(C) IYL,(C) L,(C) :

(HL) i

(HL+IX) (HL+IY) (HL+1122H) (IX+IY) . (IX+55H) (IX+1122H)

(IY+55H) (IY+1122H) (PC+1122H) (SP+1122H)

DIVU HL,D DIVU HL,E DIVU HL,H DIVU HL.IXH DIVU HL,IXL DIVU HL.IYH DIVU HL.IYL DIVU HL,L DIVU HL.66H DIVUW DEHL.(HL) DIVUW DEHL,(IX+1122H) DIVUW DEHL,(IY+1122H) DIVUW DEHL,(PC+1122H) DIVUW DEHL,(3344H) DIVUW DEHL.BC DIVUW DEHL.DE DIVUW DEHL.HL

DIVUW DEHL.IX DIVUW DEHL.IY DIVUW DEHL.SP DIVUW DEHL.3344H DIVW DEHL.(HL) DIVW DEHL,(IX+1122H) DIVW DEHL,(IY+1122H) DIVW DEHL,(PC+1122H) DIVW DEHL,(3344H) DIVW DEHL.BC DIVW DEHL.DE DIVW DEHL.HL DIVW DEHL.IX DIVW DEHL.IY DIVW DEHL.SP DIVW DEHL.3344H DJNZ 77H

El El 66H EPUF EPUI EPUM (HL) EPUM (HL+IX) EPUM (HL+IY) EPUM (HL+1122H) EPUM (IX+IY) EPUM (IX+1122H)' EPUM (IY+1122H) EPUM (PC+1122H) EPUM (SP+1122H) EPUM (3344H) EX (SP).HL EX (SP),IX EX (SP).IY EX A,(HL) EX A,(HL+IX) EX A,(HL+IY) EX A,(HL+1122H) EX A,(IX+IY) EX АД1Х+55Н) EX АД1Х+1122Н) EX A,(IY+55H) EX A.(IY+1122H)

76 ED46 ED56 ED5E ED4E DDED48 DDED50 FDED582211 DDED58 FDED482211 FDED502211 FDED402211 DDED402211 DDED784433

ED78 ..

DB66 ED40 ED48 ED50 ED58 ED60 EDB7 DDED60 DDED68 FDED60 FDED68

ED68 34 DDOC DD14 FD1C2211 DD1C DD3455 FDOC2211 FD3455 FD142211 FD042211 DD042211

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SOURCE CODE** | | **OBJECT CODE** | **SOURCE CODE** | |
| **INC** | **(3344H)** | **DD3C4433** | **JP** | **РЕДРС+1122Н)** |
| **INC** | **A** | **3C** | **JP** | **PE.3344H** |
| **INC** | **В** | **04** | **JP** | **PO,(HL)** |
| **INC** | **BC** | **03** | **JP** | **PO,(PC+1122H)** |
| **INC** | **c** | **ОС** | **JP** | **PO.3344H** |
| **INC** | **D** | **14** | **JP** | **Z,(HL)** |
| **INC** | **DE** | **13** | **JP** | **Z,(PC+1122H)** |
| **INC** | **E** | **IC** | **JP** | **Z.3344H** |
| **INC** | **H** | **24** | **JP** | **3344H** |
| **INC** | **HL** | **23** | **JR** | **C.77H** |
| **INC** | **IX** | **DD23** | **JR** | **NC,77H** |
| **INC** | **IXH** | **DD24** | **JR** | **NZ.77H** |
| **INC** | **IXL** | **DD2C** | **JR** | **Z.77H** |
| **INC** | **IY** | **FD23** | **JR** | **77H** |
| **INC** | **IYH** | **FD24** | **LD** | **(BC),A** |
| **INC** | **IYL** | **FD2C** | **LD** | **(DE),A** |
| **INC** | **L** | **2C** | **LD** | **(HL),A** |
| **INC** | **SP** | **33** | **LD** | **(HL),В** |
| **INCW** | **(HL)** | **DD03** | **LD** | **(HL),BC** |
| **INCW** | **(IX+1122H)** | **FD032211** | **LD** | **(HL),C** |
| **INCW** | **(IY+1122H)** | **FD132211** | **LD** | **(HL),D** |
| **INCW** | **(PC+1122H)** | **DD332211** |  | **(HL),DE ■'** |
| **INCW** | **(3344H)** | **DD134433** | **LD** | **(HL),E** |
| **INCW** | **BC** | **03** | **LD** | **(HL),H** |
| **INCW** | **DE** | **13** | **LD** | **(HL),HL** |
| **INCW** | **HL** | **23** | **LD** | **(HL),L** |
| **INCW** | **IX** | **DD23** | **LD** | **(HL),SP** |
| **INCW** | **IY** | **FD23** | **LD** | **(HL),66H** |
| **INCW** | **SP** | **33** | **LD** | **(HL+IX),A** |

IND INDR INDRW INDW INI INIR INIRW INIW INW JAF JAR JP JP JP JP JP JP JP JP JP JP JP JP JP JP JP JP JP JP JP JP

HL,(C) 77H 77H (HL) (IX) (IY) (PC+1122H) C,(HL) C,(PC+H22H) C.3344H M,(HL) МДРС+1122Н) M.3344H NC,(HL) NC,(PC+1122H) NC.3344H NZ,(HL) NZ,(PC+1122H) NZ.3344H P.(HL) РДРС+1122Н) P.3344H PE,(HL)

EDAA EDBA ED9A ED8A EDA2 EDB2 ED92 ED82 EDB7 DD2874 DD2074 E9

DDE9 FDE9 FDC32211 DDDA FDDA2211 DA4433 DDFA FDFA2211 FA4433 DDD2 FDD22211 D24433 DDC2 FDC22211 C24433 DDF2 FDF22211 F24433 DDEA

LD LD

LD LD

LD LD LD

LD LD LD LD

LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD

LD LD LD LD

(HL+IX),HL (HL+IX),IX (HL+IX),IY (HL+IX),66H (HL+IY),A (HL+IY),HL (HL+IY),IX (HL+IY),IY (HL+IY),66H (HL+U22H),A (HL+1122H),HL (HL-t-l 122H) ,IX (HL+1122H),IY (HL+U22H),66H (IX+IY),A (IX+IY),HL (IX+IY),IX (IX+IY),IY (IX+IY),66H (IX+55H),A (IX+55H),B (IX+55H),BC (IX+55H),C (IX+55H),D (IX+55H),DE (IX+55H),E (IX+55H),H (IX+55H),HL (IX+55H),L (IX+55H),SP (IX+55H),66H

OBJECT CODE FDEA2211 EA4433 DDE2 FDE22211 E24433 DDCA FDCA2211 CA4433

C34433 3875 3075 2075 2875 1875 02 12 77 70 EDOE 71 72 EDIE 73 74 ED2E 75 ED3E 3666 EDOB EDOD DDEDOD FDEDOD DD0E66 ED13 ED15 DDED15 FDED15 DD1666 ED3B2211 ED3D2211 DDED3D2211 FDED3D2211 FD1E221166 ED1B ED1D DDED1D FDED1D DD1E66 DD7755 DD7055 CDED0E55 DD7155 DD7255 DDED1E55 DD7355 DD7455 DDED2E55 DD7555 DDED3E55 DD365566

LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD

LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD

(IX+1122H),A (IX+1122H),HL (IX+1122H),IX (IX+1122H) ,IY (IX+1122H),66H (IY+55H),A (IY+55H),B (IY+55H),BC (IY+55H),C (IY+55H),D (IY+55H),DE (IY+55H),E (IY+55H),H (IY+55H),HL (IY+55H),L (IY+55H),SP (IY+55H),66H (IY+1122H),A (IY+1122H),HL (IY+1122H),IX (IY+1122H),IY (IY+1122H),66H (PC+1122H),A (PC+1122H),HL (PC+1122H),IX (PC+1122H),IY (PC+1122H),66H (SP+1122H),A (SP+1122H),HL (SP+1122H),IX (SP+1122H),IY (SP+1122H),66H (3344H),A (3344H),BC (3344H),DE (3344H),HL

- (3344H)JX (3344H),IY (3344H),SP (3344H),66H A,(BC) A,(DE) A,(HL) A,(HL+IX) A,(HL+IY) A,(HL+1122H) A,(IX+IY) A,(IX+55H) A,(IX+1122H) A,(IY+55H) A,(IY+1122H) A,(PC+1122H) A,(SP+1122H) A,(3344H) A,A A,В A,C A,D A.E A,H

|  |  |  |  |
| --- | --- | --- | --- |
| **OBJECT CODE** | **SOURCE CODE** | | **OBJECT CODE** |
| **ED2B2211** | **LD** | **A,l** | **ED57** |
| **ED2D2211** | **LD** | **A,IXH** | **DD7C** |
| **DDED2D2211** | **LD** | **A.IXL** | **DD7D** |
| **FDED2D2211** | **LD** | **A,IYH** | **FD7C** |
| **FD0E221166** | **LD** | **A.IYL** | **FC7D** |
| **FD7755** | **LD** | **A,L** | **7D** |
| **FD7055** | **LD** | **A.R** | **ED5F** |
| **FDED0E55** | **LD** | **A,66H** | **3E66** |
| **FD7155** | **LD** | **B,(HL)** | **46** |
| **FD7255** | **LD** | **B,(IX+55H)** | **DD4655** |
| **FDED1E55** | **LD** | **B,(IY+55H)** | **FD4655** |
| **FD7355** | **LD** | **B,A** | **47** |
| **FD7455** | **LD** | **B,B** | **40** |
| **FDED2E55** | **LD** | **B,C** | **41** |
| **FD7555** | **LD** | **B,D** | **42** |
| **FDED3E55** | **LD** | **B.E ,** | **43** |
| **FD365566** | **LD** | **B.H** | **44** |
| **ED332211** | **LD** | **B.IXH** | **DD44** |
| **ED352211** | **LD** | **B.IXL** | **DD45** |
| **DDED352211** | **LD** | **B.IYH** | **FD44** |
| **FDED352211** | **LD** | **B,IYL** | **FD45** |
| **FD16221166** | **LD** | **B.L** | **45** |
| **ED232211** | **LD** | **B.66H** | **0666** |
| **ED252211** | **LD** | **BC,(HL)** | **ED06** |
| **DDED252211** | **LD** | **ВСД1Х+55Н)** | **DDED0655** |
| **FDED252211** | **LD** | **BC,(IY+55H)** | **FDED0655** |
| **FD06221166** | **LD** | **ВСД3344Н)** | **ED4B4433 ।** |
| **ED032211** | **LD** | **BC.3344H** | **014433** |
| **ED052211** | **LD** | **C,(HL)** | **4E** |
| **DDED052211** | **LD** | **СД1Х+55Н)** | **DD4E55** |
| **FDED052211** | **LD** | **C,(IY+55H)** | **FD4E55** |
| **DD06221166** | **LD** | **C.A** | **4F** |
| **324433** | **LD** | **C,B** | **43** |
| **ED434433** | **LD** | **C.C** | **49** |
| **ED534433** | **LD** | **C.D** | **4A** |
| **224433** | **LD** | **C,E** | **4B** |
| **DD224433** | **LD .** |  | **ЛС** |
| **FD224433** | **LD** | **C,IXH** | **DD4C** |
| **ED734433** | **LD** | **C,IXL** | **DD4D** |
| **DD3E443366** | **LD** | **C,IYH** | **FD4C** |
| **OA** | **LD** | **C,IYL** | **FD4D** |
| **1A** | **LD** | **C.L** | **4D ।** |
| **7E** | **LD** | **C.66H** | **0E66** |
| **DD79** | **LD** | **D,(HL)** | **56 1** |
| **DD7A** | **LD** | **D,(IX+55H)** | **CD5655** |
| **FD7B2211** | **LD** | **D,(IY+55H)** | **FD5666** |
| **DD7B** | **LD** | **D,A** | **57** |
| **DD7E55** | **LD** | **D,B** | **50** |
| **FD792211** | **LD** | **D,C** | **51** |
| **FD7E55** | **LD** | **D,D** | **52** |
| **FD7A2211** | **LD** | **D,E** | **53** |
| **FD782211** | **LD** | **D(H** | **54** |
| **DD782211** | **LD** | **D,IXH** | **DD54** |
| **3A4433** | **LD** | **D,IXL** | **DD55** |
| **7F** | **LD** | **D,IYH** | **FD54** |
| **78** | **LD** | **D,IYL** | **FD55** |
| **79** | **LD** | **D.L** | **55** |
| **7A** | **LD** | **D,66H** | **1666** |
| **7B** | **LD** | **DE,(HL)** | **ED16** |
| **7C** | **LD** | **DE,(IX+55H)** | **DDED1655** |

**эпкгчаа**

SOURCE CODE

LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD

. LD LD

’ LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD

DE,(IY+55H) DE,(3344H) DE.3344H E,(HL) ЕД1Х+55Н) E,(IY+55H) E,A

OBJECT CODE FDED1655 ED5B4433 114433

SOURCE CODE

E,D E.E E,H E,IXH E,IXL E,IYH E.IYL

E,66H H.(HL) H,(IX+55H) H,(IY+55H) H,A

H,D H.E H,H

H.66H HL,(HL) HL,(HL+IX) HL,(HL+IY) HL,(IX+IY) HL,(IX+55H) HL,(IX+1122H) HL,(IY+55H) HL,(IY+H22H) HL,(PC+1122H)

HL,(SP+1122H) HL,(3344H) HL.3344H

I, A IX,(HL+IX) IX,(HL+IY) IX,(HL+1122H) IX.(IX-hlY) IX,(IX+1122H) IX,(IY+1122H) IX,(PC+1122H) IX,(SP+1122H) 1ХД3344Н) IX.3344H IXH,A

IXH.B IXH,C IXH,D IXH,E IXH.IXH IXH.IXL IXH.66H

DD5E55 FD5E55 5F 58 59 5A

5B 5C DD5C DD5D FD5C FD5D 5D 1E66 66 DD6655 FD6655 ’ 67

60

61

62

63

64 65 2666 ED26 EDOC ED14 ED1C DDED2655 ED2C2211 FDED2655 ED342211 ED242211 ED042211 2A4433 214433

ED47 DDEDOC DDED14 DDED3C2211 DDED1C DDED2C2211 DDED342211 DDED242211 DDED042211 DD2A4433 DD214433 DD67 DD60 DD61 DD62

DD63 DD64 DD65 DD2666

LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD

vr'\*.

IXL,A IXL,В IXL.C IXL.D IXL,E IXL.IXH IXL,IXL IXL.66H IY,(HL+IX) IY,(HL+IY) IY,(HL+1122H) IY,(IX+IY) IY,(IX+1122H) IY,(IY+1122H) IY,(PC+1122H) IY,(SP+1122H) IY.3344H IYH,A IYH,B IYH.C IYH.D

IYH.E "

IYH.IYH IYH.IYL IYH.66H IYL,A IYL,B IYL.C IYL,D IYL,E IYL.IYH IYL.IYL IYL.66H l.(hl) L,(IX+55H) L,(IY+55H)

LD LD LDA LDA LDA LDA LDA LDA LDA

L,66H R,A SP.(HL) SP,(IX+55H) SP,(IY+55H) SP,(3344H) SP.HL SP.IX SP.IY

SP.3344H HL,(HL+IX) HL,(HL+IY) HL,(HL+1122H) HL,(IX+IY) HL,(IX+1122H) HL,(IY+1122H) HL,(PC+1122H)

OBJECT CODE DD6F

DD68 DD69 DD6A DD6B DD6C DD6D DD2E66 FDEDOC FDED14 FDED3C2211

FDED1C FDED2C2211 FDED342211 FDED242211 FDED042211 FD214433 FD67

FD60 FD61 FD62 FD63 FD64 FD65 FD2666 FD6F FD68 FD69 FD6A FD6B FD6C FD6D FD2E66

6E

DD6E55 FD6E55 6F 68 69

6 A

6B 6C 6D 2E66 ED4F ED36 DDED3655 FDED3655

ED7B4433 F9 DDF9 FDF9 314433 EDOA ED12

ED3A2211 EDI A ED2A2211 ED322211 ED222211

OBJECT CODE ED022211 214433 DDEDOA DDED12 DDED3A2211 DDED1A DDED2A2211 DDED322211 DDED222211 DDED022211 DD214433 FDEDOA FDED12 FDED3A2211 FDED1A FDED2A2211 FDED322211 FDED222211 FDED022211 FD214433 ED6E DDED6E FDED6E ED66 ED87 DDED66 DDED87 FDED66 FDED87 ED8F DDED8F FDED8F EDA8 EDB8 EDAO EDBO ED8E DDED8E55 FDED8E55 ED86 DDED8655 FDED8655 ED9E DDED9E55 FDED9E55 ED96 DDED9655 FDED9655 EDOE EDIE ED2E ED3E DD014433 EDOD DDEDOD FDEDOD ED15 DDED15 FDED15 ED3D2211

OEJECT CODE DDED3D2211 FDED3D2211 ED1D DDED1D FDED1D DDED0E55 DDED1E55 DCED2E55 DDED3E55 ED2D2211 DCiED2D22U FDED2D2211 FDED0E55 FDED1E55 FDED2E55 FCED3E55 ED352211 DDED352211 ’ FDED352211

ED252211 DDED252211 FDED252211 DD3122114433 ED052211 DDED052211 FDED052211 EO434433 ED534433 224433 DD224433 FD224433 ED734433 DD1144339988 ED06 DDED0655 FDED0655 ED4B4433 014433 ED16 DDED1655 FDED1655 ED5B4433 114433 ED26 EDOC ED14 ED3C2211 ED1C DDED2655 ED2C2211 FDED2655 ED342211 ED242211 ED042211 2A4433 214433 DDEDOC DDED14 DDED3C2211 DDED1C

SOURCE CODE LDA HL,(SP+1122H)

LDA HL,(3344H) LDA IX,(HL+IX) LDA IX,(HL+IY) LDA IX,(HL+1122H) LDA IX,(IX+IY) LDA IX,(IX+1122H) LDA IX,(IY4-1122H) LDA 1ХДРС+1122Н) LDA IX,(SP+1122H)

LDA IX,(3344H) LDA IY,(HL+IX) LDA IY,(HL+IY)

LDA IY,(HL+1122H) LDA IY,(IX+IY) LDA IY,(IX+1122H) LDA IY,(IY+1122H) LDA IY,(PC+1122H) LDA IY,(SP+1122H)

LDA IY,(3344H) LDCTL (C),HL LDCTL (C),IX LDCTL (C),IY LDCTL HL,(C) LDCTL HL.USP LDCTL IX,(C) LDCTL IX.USP LDCTL IY,(C) LDCTL IY.IISP LDCTL USP.HL LDCTL USP.IX LDCTL USP.IY LDD

LDDR LDI LDIR

LDUD (HL),A ...

LDUD (IX+55H),A ‘

LDUD (IY+55H),A LDUD A,(HL) LDUD A,(IX+55H) LDUD A,(IY+55H) LDUP (HL),A LDUP (IX+55H),A LDUP (IY+55H),A LDUP A,(HL) LDUP A,(IX+55H) LDUP A,(IY+55H) LDW (HL),BC LDW (HL),DE LDW (HL),HL LDW (HL),SP LDW (HL),3344H LDW (HL+IX),HL LDW (HL+IX),IX LDW (HL+IX),IY

LDW (HL+IY),HL LDW (HL+IY),IX LDW (HL+IY),IY

LDW (HL+1122H),HL

SOURCE CODE

LDW (HL+U22H),IX LDW (HL+1122H),IY LDW (IX+IY),HL LDW (IXH-IY).IX LDW (IX+IY),IY LDW (IX+55H),BC LDW (IX+55H),DE LDW (IX+55H),HL LDW (IX+55H),SP LDW (IX+1122H),HL LDW (IX+1122H),IX LDW (IX+1122H),IY LDW (IY+55H),BC LDW (IY+55H),DE LDW (IY+55H),HL LDW (IY+55H),SP LDW (IY+1122H),HL LDW (IY+1122H),IX LDW (IY+1122H),IY LDW (PC+1122H),HL LDW (PC+1122H),IX LDW (PC+1122H),IY LDW (PC+1122H),3344H LDW (SP+1122H),HL LDW (SP+1122H),IX LDW (SP+1122H),IY LDW (3344H),BC LDW (3344H),DE LDW (3344H),HL LDW (3344H),IX LDW (3344H),IY LDW (3344H),SP LDW (3344H),8899H LDW BC,(HL) LDW BC,(IX+55H) LDW BC,(IY+55H) LDW BC,(3344H)

LDW BC.3344H \*K’

LDW DE,(HL) LDW DE,(IX+55H) LDW DE,(IY+55H) LDW DE,(3344H) LDW DE,3344H LDW HL,(HL)

LDW HL,(HL+IX) LDW HL,(HL+IY) LDW HL,(HL+1122H) LDW HL,(IX+IY) LDW HL,(IX+55H) LDW HL,(IX+1122H) LDW HL,(IY+55H) LDW HL,(IY+1122H) LDW HL,(PC+H22H) LDW HL,(SP+1122H) LDW HL,(3344H) LDW HL.3344H LDW IX,(HL+IX) LDW IX,(HL+IY) LDW IX,(HL+1122H) LDW IX,(IX+IY)

SOURCE CODE

LDW IX,(IX+1122H) LDW IX,(I Y+1122H)

LDW IX,(PC+1122H) LDW IX,(SP+1122H) LDW 1ХД3344Н) LDW IX.3344H LDW IY,(HL+IX) LDW IY,(HL+IY) LDW IY,(HL+1122H) LDW IY,(IX+IY) LDW IY,(IX+1122H) LDW IY,(IY+1122H) LDW IY,(PC+1122H) LDW IY, (SP+1122H) LDW IY,(3344H) LDW IY.3344H LDW SP,(HL) LDW SP,(IX+55H) LDW SP,(IY+55H) LDW SP,(3344H) LDW SP.HL LDW SP.IX LDW SP,IY LDW SP,3344H MEPU (HL) MEPU (HL+IX) MEPU (HL+IY) MEPU (HL+1122H) MEPU (IX+IY) MEPU (IX+1122H) MEPU (IY+1122H) MEPU (PC+1122H) MEPU (SP+1122H) MEPU (3344H) MULT A,(HL) MULT A,(HL+IX) MULT A,(HL+IY) MULT A,(HL+1122H) MULT A,(IX+IY) MULT A,(IX+55H) MULT A,(IX+1122H) MULT A,(IY+55H) MULT A,(IY+1122H) MULT A,(PC+1122H) MULT A,(SP+1122H) MULT АД3344Н) MULT A,A MULT A,В MULT A,C MULT A,D MULT A,E MULT A,H MULT A,IXH MULT A.IXL MULT A,IYH MULT A,IYL MULT A,L MULT A.66H MULTU A,(HL) MULTU A,(HL+IX)

MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTU MULTUW MULTUW MULTUW MULTUW MULTUW MULTUW MULTUW MULTUW MULTUW MULTUW MULTUW MULTUW MULTW MULTW MULTW MULTW MULTW MULTW MULTW MULTW MULTW MULTW MULTW MULTW

NEG

NEG NOP OR OR OR OR OR OR OR OR OR OR OR

A,(HL+IY) A,(HL+1122H) A,(IX+IY) A,(IX+55H) АД1Х+1122Н) A,(IY+55H) A,(IY+1122H) A,(PC+1122H) A,(SP+1122H) A,(3344H) A,A A,В A,C A.D A,E A.H

A,IXH A.IXL A,IYH A,IYL A.L

*A*,66H ! "■’\*:\*

HL,(HL) HL,(IX+1122H) HL,(IY+1122H) HL,(PC+1122H) HL,(3344H) HL.BC

HL,DE HL,HL HL,IX HL.IY HL.SP HL.3344H HL,(HL) HL,(IX+1122H) HL,(IY+1122H)

HL,(PC+1122H) HL,(3344H) HL.BC

HL,DE HL,HL HL,IX HL.IY HL.SP HL.3344H A

OBJECT CODE DDED2C2211 DDED342211 DDED242211 DDED042211 DD2A4433 DD214433 FDEDOC FDED14 FDED3C2211 FDED1C FDED2C2211 FDED342211 FDED242211 FDED042211 FD2A4433 FD214433 ED36 DDED3655 FDED3655 ED7B4433 F9

DDF9 a-

FDF9 314433 EDAE ED8D ED95 EDBD2211 ED9D

EDAD2211 EDB52211 EDA52211 ED852211 EDAF4433 EDFO DDEDC8 DDEDDO FDEDD82211 DDEDD8 DDEDF055 FDEDC82211 FDEDF055 FDEDD02211 FDEDC02211 DDEDC02211 DDEDF84433 EDF8 EDCO EDC8 EDDO EDD8 EDEO DDEDEO DDEDE8 FDEDEO FDEDE8 EDE8 FDEDF866 EDF1 DDEDC9

HL

A,(HL)

A,(HL+IX) A,(HL+IY) A,(HL+U22H) A,(IX+IY) A,(IX+55H) A,(IX+1122H) A,(IY+55H) A,(IY+1122H) A,(PC+1122H)

A,(SP+1122H)

DDEDD1 FDEDD92211 DDEDD9 DDEDF155 FDEDC92211 FDEDF155 FDEDD12211 FDEDC12211 DDEDC12211 DDEDF94433 EDF9 EDC1 EDC9 EDD1

EDD9 f, EDEI DDEDE1 DDEDE9 FDEDE1 FDEDE9 EDE9 FDEDF966 DDEDC3 FDEDC32211 FDEDD32211 DDEDF32211

DDEDD34433 EDC3 EDD3 EDE3 DDEDE3 FDEDE3 EDF3

FDEDF34433 DDEDC2 FDEDC22211 FDEDD22211 DDEDF22211 DDEDD24433 EDC2 EDD2 EDE2 DDEDE2 FDEDE2 EDF2 FDEDF24433 ED44

ED4C 00

B6

DDB1 DDB2 FDB32211 DDB3 DDB655 FDB12211

FDB655 FDB22211 FDB02211 DDB02211

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SOURCE CODE** | | **OBJECT CODE** | **SOURCE CODE** | | **OBJECT CODE** |
| **OR** | **АД3344Н)** | **DDB74433** | **PUSH** | **HL** | **E5** |
| **OR** | **A,A** | **B7** | **PUSH** | **IX** | **DDE5** |
| **OR** | **A,В** | **BO** | **PUSH** | **IY** | **FDE5** |
| **OR** | **A,C** | **Bl** | **PUSH** | **3344H** | **FDF54433** |
| **OR** | **A,D** | **B2** | **RES** | **0,(HL)** | **CB86** |
| **OR** | **A.E** | **B3** | **RES** | **ОД1Х+55Н)** | **DDCB5586** |
| **OR** | **A.H** | **B4** | **RES** | **0,(IY+55H)** | **FDCB5586** |
| **OR** | **A.IXH** | **DDB4** | **RES** | **0,A** | **CB87** |
| **OR** | **A,IXL** | **DDB5** | **RES** | **0,B** | **CB80** |
| **OR** | **A,IYH** | **FDB4** | **RES** | **o.c** | **CB81** |
| **OR** | **A.IYL** | **FDB5** | **RES** | **O.D** | **CB82** |
| **OR** | **A.L** | **B5** | **RES** | **O.E** | **CB83** |
| **OR** | **A.66H** | **F666** | **RES** | o.h | **CB84** |
| **OTDR** |  | **EDBB** | **RES** | **0,L •** | **CB85** |
| **OTDRW** | | **ED9B** | **RES** | **1,(HL)** | **CB8E** |
| **OTIR** |  | **EDB3** | **RES** | **1,(IX+55H)** | **DDCB558E** |
| **OTIRW** | | **ED93** | **RES** | **1,(IY+55H)** | **FDCB558E** |
| **OUT** | **(C),(HL+IX)** | **DDED49** | **RES** | **l.A** | **CB8F** |

OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT OUT

(C),(HL+IY) (C),(HL+1122H) (C),(IX+IY) (C),(IX+1122H) (C),(IY+1122H) (C),(PC+1122H) (C),(SP+1122H) (СЦ3344Н) (C),A (C),B (C).C (<=).D (C).E (C),H (C),HL (C),IXH (C),IXL (C),IYH (C),IYL

(C),L

(66Н),А

OUTD OUTDW OUTI OUTIW OUTW (C),HL PCACHE

POP POP POP POP POP POP POP POP POP

PUSH PUSH PUSH PUSH PUSH PUSH

(HL) (PC+1122H) (3344H)

AF

BC

DE

HL

IX IY (HL) (PC+1122H) (3344H) AF

BC DE

DDED51

FDED592211 DDED59 FDED492211

FDED512211 FDED412211 DDED412211 DDED794433

ED79 ED41 ED49 ED51

ED59 ED61 EDBF DDED61 DDED69 FDED61 FDED69 ED69 D366 EDAB ED8B EDA3 ED83 EDBF ED65 DDC1 DDF12211 DDD14433

Fl Cl DI El DDE1 FDE1 DDC5 DDF52211 DDD54433 F5

C5 D5

/w».

RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES RES

2,(HL) 2Д1Х+55Н) 2,(IY+55H) 2,A

2.D

2,H

3.(HL) ЗД1Х+55Н) 3,(IY+55H) 3.A

3.D

з.н

4,(HL) 4Д1Х+55Н) 4,(IY+55H)

5,(HL) 5Д1Х+55Н) 5,(IY+55H)

5,A

CB88 CB89 CB8A CB8B CB8C CB8D CB96 DDCB5596 FDCB5596 CB97 CB90

CB91 CB92

CB93 CB94 CB95 CB9E DDCB559E FDCB559E CB9F

CB98 CB99 CB9A CB9B CB9C CB9D CBA6 DDCB55A6 FDCB55A6 CBA7 CBAO CBA1 CBA2 CBA3 CBA4 CBA5 СВАЕ

DDCB55AE FDCB55AE CBAF CBA8 CBA9

■i-.-

|  |  |  |  |
| --- | --- | --- | --- |
| **SOURCE CODE** | **OBJECT CODE** | **SOURCE CODE** | **OBJECT CODE** |
| **RES 5,D** | **CBAA** | **RR (IX+55H)** | **DDCB551E** |
| **RES 5,E** | **CBAB** | **RR (IY+55H)** | **FDCB551E** |
| **RES 5,H** | **CBAC** | **RR A** | **CB1F** |
| **RES 5,L** | **CBAD** | **RR В** | **CB18** |
| **RES 6,(HL)** | **CBB6** | **RR C** | **CB19** |
| **RES 6Д1Х+55Н)** | **DDCB55B6** | **RR D** | **CB1A** |
| **RES 6,(IY+55H)** | **FDCB55B6** | **RR E** | **CB1B** |
| **RES 6,A** | **CBB7** | **RR H** | **CB1C** |
| **RES 6,В** | **CBBO** | **RR L** | **CB1D** |
| **RES 6,C** | **CBB1** | **RRA** | **IF** |
| **RES 6,D** | **CBB2** | **RRC (HL)** | **СВОЕ** |
| **RES 6,E** | **CBB3** | **RRC (IX+55H)** | **DDCB55OE** |
| **RES 6,H** | **CBB4** | **RRC (IY+55H)** | **FDCB550E** |
| **RES 6,L** | **CBB5** | **RRC A** | **CBOF** |
| **RES 7,(HL)** | **CBBE** | **RRC В** | **CB08** |
| **RES 7Д1Х+55Н)** | **DDCB55BE** | **RRC C** | **CB09** |
| **RES 7,(IY+55H)** | **FDCB55BE** | **RRC D** | **CBOA** |
| **RES 7,A** | **CBBF** | **RRC E** | **CBOB** |
| **RES 7,В** | **CBB8** | **RRC H** | **CBOC** |
| **RES 7,C** | **CBB9** | **RRC L** | **CBOD** |
| **RES 7,D** | **CBBA** | **RRCA** | **OF** |
| **RES 7,E** | **■ CBBB** | **RRD** | **ED67** |
| **RES 7,H** | **CBBC** | **RST OOH** | **C7** |
| **RES 7.L** | **CBBD** | **RST 08H** | **CF** |
| **RET** | **C9** | **RST 10H** | **D7** |
| **RET C** | **D8** | **RST 18H** | **DF** |
| **RET M** | **F8** | **RST 20H** | **E7** |
| **RET NC** | **DO** | **RST 28H** | **EF** |
| **RET NZ** | **CO** | **RST ЗОН** | **F7** |
| **RET P** | **FO** | **RST 38H** | **FF** |
| **RET PE** | **E8** | **SBC A,(HL)** | **9E** |
| **RET PO** | **EO** | **SBC A,(HL+IX)** | **DD99** |
| **RET Z** | **C8** | **SBC A,(HL+IY)** | **DO9A** |
| **RETI** | **ED4D** | **SBC A,(HL+1122H)** | **FD9B2211** |
| **RETIL** | **ED55** | **SBC A,(IX+IY)** | **DD9B** |
| **RETN** | **ED45** | **SBC A,(IX+55H)** | **DD9E55** |
| **RL (HL)** | **CB16** | **SBC A,(IX+1122H)** | **FD992211** |
| **RL (IX+55H)** | **DDCB5516** | **SBC A,(IY+55H)** | **FD9E55** |
| **RL (IY+55H)** | **FDCB5516** | **SBC A,(IY+1122H)** | **FD9A2211** |
| **RL A** | **CB17** | **SBC A,(PC+1122H)** | **FD982211** |
| **RL В** | **CB10** | **SBC A,(SP+1122H)** | **DD982211** |
| **RL C** | **CB11** | **SBC АД3344Н)** | **DD9F4433** |
| **RL D** | **CB12** | **SBC A,A** | **9F** |
| **RL E** | **CB13** | **SBC A,В** | **98** |
| **RL H** | **CB14** | **SBC A,C** | **99** |
| **RL L** | **CB15** | **SBC A,D** | **9A** |
| **RLA** | **17** | **SBC A,E** | **9B** |
| **RLC (HL)** | **CB06** | **SBC A,H** | **9C** |
| **RLC (IX+55H)** | **DDCB5506** | **SBC A,IXH** | **DD9C** |
| **RLC (IY+55H)** | **FDCB5506** | **SBC A.IXL** | **DD9D** |
| **RLC A** | **CB07** | **SBC A.IYH** | **FD9C** |
| **RLC В** | **CBOO** | **SBC A,IYL** | **FD9D** |
| **RLC C** | **CB01** | **SBC A,L** | **9D** |
| **RLC D** | **CB02** | **SBC A.66H** | **DE66** |
| **RLC E** | **СВОЗ** | **SBC HL.BC** | **ED42** |
| **RLC H** | **CB04** | **SBC HL,DE** | **ED52** |
| **RLC L** | **CB05** | **SBC HL,HL** | **ED62** |
| **RLCA** | **07** | **SBC HL,SP** | **ED72** |
| **RLD** | **ED6F** | **SBC IX,BC** | **DDED42** |
| **RR (HL)** | **CB1E** | **SBC IX,DE** | **DDED52** |

OBJECT CODE DDED62 DDED72 FDED42 FDED52 FDED62 FDED72 ED714433 37 CBC6 DDCB55C6 FDCB55C6 CBC7 CBCO CBC1

CBC2 CBC3 CBC4 CBC5 CBCE DDCB55CE FDCB55CE CBCF

CBC8 CBC9 CBCA CBCB CBCC CBCD CBD6 DDCB55D6 FDCB55D6 CBD7

CBDO CBD1 CBD2 CBD3

CBD4 .... ,,,,л

CBD5 CBDE DDCB55DE FDCB55DE CBDF CBD8

CBD9 CBDA CBDB CBDC CBDD

CBE6 DDCB55E6 FDCB55E6 CBE7 CBEO CBE1

CBE2 СВЕЗ

CBE4 CBE5 CBEE DDCB55EE

SOURCE CODE SBC IX,IX

SBC IX.SP SBC IY,BC SBC IY,DE SBC IY,IY SBC IY.SP SC 3344H SCF SET 0,(HL) SET 0,(IX+55H)

SET 0,(IY+55H)

SET 0,A SET 0,B SET 0,C SET 0,D SET 0,E SET 0,H SET 0,L SET 1,(HL)

SET 1Д1Х+55Н)

SET 1,(IY+55H)

SET 1,A SET 1,B SET l.C SET 1,D SET l.E SET l.H SET l.L SET 2,(HL)

SET 2,(IX+55H)

SET 2,(IY+55H)

SET 2,A SET 2,В SET 2,C SET 2,D SET 2,E

SET, 2.H ......., .

SET 2.L

SET 3,(HL)

SET ЗД1Х+55Н)

SET 3,(IY+55H)

SET 3,A SET 3,B SET 3,C SET 3,D SET 3,E SET 3,H SET 3,L SET 4,(HL)

SET 4,(IX+55H)

SET 4,(IY+55H)

SET 4,A SET 4,В SET 4,C SET 4,D SET 4,E SET 4.H SET 4,L SET 5,(HL)

SET 5,(IX+55H)

SOURCE CODE

SET 5,(IY+55H)

SET 5,A

SET 5,В SET 5,C

SET 5,D SET 5.E SET 5,H SET 5,L SET 6,(HL) SET 6Д1Х+55Н) SET 6,(IY+55H) SET 6,A

SET 6,В SET 6,C SET 6,D SET 6,E SET 6,H SET 6.L SET 7,(HL)

SET 7,(IX+55H)

SET 7,(IY+55H) SET 7,A

SET 7,В

SET 7,C SET 7,D SET 7,E SET 7,H SET 7,L SLA (HL) SLA (IX+55H)

SLA (IY+55H) SLA A SLA В SLA C SLA D SLA E

**SLA . ... H .... , , SLA L**

SRA (HL)

SRA (IX+55H)

SRA (IY+55H) SRA A SRA В

SRA C SRA D SRA E SRA H . SRA L SRL (HL) SRL (IX+55H) SRL (IY+55H) SRL A SRL В SRL C SRL D SRL E SRL H SRL I SUB A,(HL)

SUB A,(HL+IX)

OBJECT CODE FDCB55EE CBEF CBE8

CEE9 CBEA CBEB СВЕС CEED CBF6 DDCB55F6 FDCB55F6 CBF7

CEFO CBF1 CBF2 CEF3 CEF4 CBF5 CBFE DDCB55FE FDCB55FE CEFF

CBF8 CBF9 CBFA CBFB CBFC CBFD CB26 DDCB5526 FDCB5526 CE27 CB20 CE21 CB22

CB23 CE24 v Хч' • >

CB25 CE2E DDCB552E FCCB552E CE2F

CB28 CE29 CB2A CB2B CE2C CB2D CE3E DDCB553E FDCB553E CB3F CB38

CB39 CB3A CB3B CB3C CE3D 96 DD91

OBJECT CODE DD92 FD932211 DD93 DD9655 FD912211 FD9655 FD922211 FD902211 DD902211 DD974433 97 90 91 92 93 94 DD94 DD95 FD94 FD95 95

D666 :

DDEDCE FDEDCE2211 FDEDDE2211 DDEDFE2211 DDEDDE4433 EDCE EDDE EDEE DDEDEE FDEDEE EDFE FDEDFE4433 CB36

OBJECT CODE DDCB5536 FDCB5536

CB37 CB30

CB31 CB32 CB33 CB34 CB35 ED70

AE

DDA9 DDAA FDAB2211 DDAB DDAE55 FDA92211 FDAE55 FDAA2211 FDA82211 DDA82211 DDAF4433 AF A8 A9 AA AB AC DDAC DDAD FDAC FDAD AD

EE66

SOURCE CODE SUB A,(HL+IY) SUB A,(HL+1122H) SUB A,(IX+IY) SUB АД1Х+55Н) SUB A,(IX+1122H) SUB A,(IY+55H) SUB A,(IY+1122H) SUB АДРС+1122Н) SUB A,(SP+1122H) SUB АД3344Н) SUB A,A

SUB A,В

SUB A,C SUB A,D SUB A,E SUB A,H SUB A.IXH SUB A,IXL

SUB A.IYH SUB A.IYL SUB A.L SUB A.66H SUBW HL,(HL) SUBW HL,(IX+1122H) SUBW HL,(IY+1122H) SUBW HL,(PC+1122H) SUBW HL,(3344H) SUBW HL.BC SUBW HL,DE SUBW HL,HL SUBW HL,IX SUBW HL.IY SUBW HL.SP SUBW HL.3344H TSET (HL)

SOURCE CODE TSET (IX+55H) TSET (IY+55H) TSET A

TSET В TSET C TSET D TSET E TSET H TSET L TSTI (C) XOR A,(HL) XOR A,(HL+IX) XOR A,(HL+IY) XOR A,(HL+1122H) XOR A,(IX+IY) < XOR АД1Х+55Н) XOR A,(IX+1122H) XOR A,(IY+55H) XOR A,(IY+1122H) XOR A,(PC+1122H) XOR A,(SP+1122H) XOR A,(3344H) XOR A,A

XOR A,В

XOR A.C XOR A.D XOR A.E XOR A.H XOR A.IXH XOR A,IXL

XOR A.IYH XOR A.IYL XOR A.L XOR A.66H



. Appendix D.

Instructions in Numeric Order

мп—~~еи—миным~~тя—^—————■ЕШЯНЯ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **OBJECT CODE 00 014433 014433** | **SOURCE CODE NOP** | | **OBJECT CODE 29 2A4433 2A4433** | **SOURCE CODE** | |
| **ADD LD LDW** | **HL,HL HL,(3344H) HL,(3344H)** |
| **LD**  **LDW** | **BC.3344H**  **BC.3344H** |
| **02** | **LD** | **(BC),A** | **2B** | **DEC** | **HL** |
| **03** | **INCW** | **BC** | **2B** | **DECW** | **HL** |
| **03** | **INC** | **BC** | **2C** | **INC** | **L** |
| **04** | **INC** | **В** | **2D** | **DEC** | **L** |
| **05** | **DEC** | **В** | **2E66** | **LD** | **L,66H** |
| **0666** | **LD** | **B,66H** | **2F** | **CPL** |  |
| **07** | **RLCA** |  | **3075** | **JR** | **NC,77H** |
| **08** | **EX** | **AF,AF’** | **314433** | **LD** | **SP,3344H** |
| **09** | **ADD** | **HL,ВС** | **314433** | **LDW** | **SP.3344H** |
| **•' 0A** | **LD** | **А,(ВС) >** |  | **LD** | **(3344H),A** |
| **0B** | **DEC** | **ВС** | **33** | **INC** | **SP** |
| **0B** | **DECW** | **ВС** | **33** | **INCW** | **SP** |
| **ОС** | **INC** | **С** | **34** | **INC** | **(HL)** |
| **0D** | **DEC** | **С** | **35** | **DEC** | **(HL)** |
| **0E66** | **LD** | **С.66Н** | **3666** | **LD** | **(HL),66H** |
| **OF** | **RRCA** |  | **37** | **SCF** |  |
| **1075** | **DJNZ** | **77Н** | **3875** | **JR** | **C,77H** |
| **114433** | **LD** | **DE.3344H** | **39** | **ADD** | **HL,SP** |
| **114433** | **LDW** | **DE.3344H** | **3A4433** | **LD** | **АД3344Н)** |
| **12** | **LD** | **(BE),А** | **3B** | **DEC** | **SP** |
| **13** | **INC** | **DE** | **3B** | **DECW** | **SP** |
| **13** | **INCW** | **DE** | **3C** | **INC** | **A** |
| **14** | **INC** | **D** | **3D** | **DEC** | **A** |
| **15** | **DEC** | **D** | **3E66** | **LD** | **A,66H** |
| **1666** | **LD** | **D,66H** | **3F** | **CCF** |  |
| **17** | **RLA** |  | **40** | **LD** | **B.B** |
| **1875** | **JR** | **77H** | **41** | **LD** | **B,C** |
| **19** | **ADD** | **HL,DE** | **42** | **LD** | **B,D** |
| **1A** | **LD** | **A,(DE)** | **• • ■ 43 ■ ■** | **LD** | **B,E** |
| **IB** | **DEC** | **DE** | **44** | **LD** | **B,H** |
| **IB** | **DECW** | **DE** | **45** | **LD** | **B.L** |
| **IC** | **INC** | **E** | **46** | **LD** | **B,(HL)** |
| **ID** | **DEC** | **E** | **47** | **LD** | **B,A** |
| **1E66** | **LD** | **E,66H** | **48** | **LD** | **C,B** |
| **IF** | **RRA** |  | **49** | **LD** | **C,C** |
| **2075** | **JR** | **NZ.77H** | **4A** | **LD** | **C,D** |
| **214433** | **LD** | **HL.3344H** | **4B** | **LD** | **C,E** |
| **214433** | **LDA** | **HL,(3344H)** | **4C** | **LD** | **C.H** |
| **214433** | **LDW** | **HL.3344H** | **4D** | **LD** | **C.L** |
| **224433** | **LD** | **(3344H),HL** | **4E** | **LD** | **C,(HL)** |
| **224433** | **LDW** | **(3344H),HL** | **4F** | **LD** | **C.A** |
| **23** | **INCW** | **HL** | **50** | **LD** | **D,B** |
| **23** | **INC** | **HL** | **51** | **LD** | **D,C** |
| **24** | **INC** | **H** | **52** | **LD** | **D,D** |
| **25** | **DEC** | **H** | **53** | **LD** | **D,E** |
| **2666** | **LD** | **H,66H** | **54** | **LD** | **D,H** |
| **27** | **DAA** |  | **55** | **LD** | **D,L** |
| **2875** | **JR** | **Z,77H** | **56** | **LD** | **D,(HL)** |

|  |  |  |
| --- | --- | --- |
| **OBJECT CODE** | **SOURCE CODE** | |
| **57** | **LD** | **D,A** |
| **58** | **LD** | **E.B** |
| **59** | **LD** | **E,C** |
| **5A** | **LD** | **E,D** |
| **5B** | **LD** | **E,E** |
| **5C** | **LD** | **E,H** |
| **5D** | **LD** | **E,L** |
| **5E** | **LD** | **E,(HL)** |
| **5F** | **LD** | **E,A** |
| **60** | **LD** | **H.B** |
| **61** | **LD** | **H.C** |
| **62** | **LD** | **H.D** |
| **63** | **LD** | **H.E** |
| **64** | **LD** | **H.H** |
| **65** | **LD** | **H.L** |
| **66** | **LD** | **H,(HL)** |
| **67** | **LD** | **H.A** |
| **68** | **LD** | **L.B** |
| **69 ■■** | **LD** | **L.C** |
| **6A** | **LD** | **L,D** |
| **6B** | **LD** | **L,E** |
| **6C** | **LD** | **L,H** |
| **6D** | **LD** | **L,L** |
| **6E** | **LD** | **L,(HL)** |
| **6F** | **LD** | **L.A** |
| **70** | **LD** | **(HL),В** |
| **71** | **LD** | **(HL),C** |
| **72** | **LD** | **(HL),D** |
| **73** | **LD** | **(HL),E** |
| **74** | **LD** | **(HL),H** |
| **75** | **LD** | **(HL),L** |
| **76** | **HALT** |  |
| **77** | **LD** | **(HL),A** |
| **78** | **LD** | **A,В** |
| **79** | **LD** | **A,C** |
| **7A** | **LD** | **A.D** |
| **7B** | **LD** | **A,E ,** |
|  | **LD** | **A,H** |
| **7D** | **LD** | **A.L** |
| **7E** | **LD** | **A.(HL)** |
| **7F** | **LD** | **A,A** |
| **80** | **ADD** | **A.B** |
| **81 '** | **ADD** | **A,C** |
| **82** | **ADD** | **A,D** |
| **83** | **ADD** | **A.E** |
| **84** | **ADD** | **A.H** |
| **85** | **ADD** | **A,L** |
| **86** | **ADD** | **A,(HL)** |
| **87** | **ADD** | **A,A** |
| **88** | **ADC** | **A,В** |
| **89** | **ADC** | **A,C** |
| **8A** | **ADC** | **A,D** |
| **8B** | **ADC** | **A,E** |
| **8C** | **ADC** | **A,H** |
| **8D** | **ADC** | **A,L** |
| **8E** | **ADC** | **A,(HL)** |
| **8F** | **ADC** | **A,A** |
| **90** | **SUB** | **A,В** |
| **91** | **SUB** | **A,C** |
| **92** | **SUB** | **A,D** |

|  |  |  |
| --- | --- | --- |
| **OBJECT CODE** | **SOURCE CODE** | |
| **93** | **SUB** | **A,E** |
| **94** | **SUB** | **A.H** |
| **95** | **SUB** | **A,L** |
| **96** | **SUB** | **A,(Hl)** |
| **97** | **SUB** | **A,A** |
| **98** | **SBC** | **A,В** |
| **99** | **SBC** | **A,C** |
| **9A** | **SBC** | **A.D** |
| **9B** | **SBC** | **A,E** |
| **9C** | **SBC** | **A,H** |
| **9D** | **SBC** | **A.L** |
| **9E** | **SBC** | **A,(HL)** |
| **9F** | **SBC** | **A,A** |
| **AO** | **AND** | **A.B** |
| **Al** | **AND** | **A,C** |
| **A2** | **AND** | **A.D** |
| **A3** | **AND** | **A.E** |
| **A4** | **AND** | **A,H** |
| **A5** | **AND** | **A.L** |
| **A6** | **AND** | **A,(HL)** |
| **A7.** | **AND** | **A .A** |
| **A8** | **XOR** | **A,В** |
| **A9** | **XOR** | **A.C** |
| **AA** | **XOR** | **A.D** |
| **AB** | **XOR** | **A.E** |
| **AC** | **XOR** | **A.H** |
| **AD** | **XOR** | **A,L** |
| **AE** | **XOR** | **A,(HL)** |
| **AF** | **XOR** | **A .A** |
| **BO** | **OR** | **A,В** |
| **Bl** | **OR** | **A.C** |
| **B2** | **OR** | **AD** |
| **B3** | **OR** | **A.E** |
| **B4** | **OR** | **A.H** |
| **B5** | **OR** | **A.L** |
| **B6** | **OR** | **A,(HL)** |
| **B7** | **OR** | **A,A** |
| **B8** | **CP** | **A,В** |
| **B9** | **CP** | **A.C** |
| **BA** | **CP** | **A,D** |
| **BB** | **CP** | **A,E** |
| **BC** | **CP** | **A.H** |
| **BD** | **CP** | **A,L** |
| **BE** | **CP** | **A,(HL)** |
| **BF** | **CP** | **A,A** |
| **CO** | **RET** | **NZ** |
| **Cl** | **POP** | **BC** |
| **C24433** | **JP** | **N2.3344H** |
| **C34433** | **JP** | **3344H** |
| **C44433** | **CALL** | **NZ.3344H** |
| **C5** | **PUSH** | **BC** |
| **C666** | **ADD** | **A.66H** |
| **C7** | **RST** | **OOH** |
| **C8** | **RET** | **z** |
| **C9** | **RET** |  |
| **CA4433** | **JP** | **Z.3344H** |
| **CBOO** | **RLC** | **В** |
| **CB01** | **RLC** | **c** |
| **CB02** | **RLC** | **D** |
| **СВОЗ** | **RLC** | **E** |

|  |  |  |
| --- | --- | --- |
| **OBJECT CODE** | **SOURCE CODE** | |
| **CB7C** | **BIT** | **7.H** |
| **CB7D** | **BIT** | **7,L** |
| **CB7E** | **BIT** | **7,(HL)** |
| **CB7F** | **BIT** | **7,A** |
| **CB80** | **RES** | **O.B** |
| **CB81** | **RES** | **0,C** |
| **CB82** | **RES** | **O,D** |
| **CB83** | **RES** | **O.E** |
| **CB84** | **RES** | **O.H** |
| **CB85** | **RES** | **O.L** |
| **CB86** | **RES** | **O.(HL)** |
| **CB87** | **RES** | **O,A** |
| **CB88** | **RES** | **1,B** |
| **CB89** | **RES** | **i.c** |
| **CB8A** | **RES** | **1,D** |
| **CB8B** | **RES** | **l.E** |
| **CB8C** | **RES** | **l.H** |
| **CB8D .......** | **RES** | **l.L** |
| **CB8E** | **RES** | **l.(HL)** |
| **CB8F** | **RES** | **l.A** |
| **CB90** | **RES** | **2.B** |
| **CB91** | **RES** | **2.C** |
| **CB92** | **RES** | **2.D** |
| **CB93** | **RES** | **2.E** |
| **CB94** | **RES** | **2.H** |
| **CB95** | **RES** | **2.L** |
| **CB96** | **RES** | **2.(HL)** |
| **CB97** | **RES** | **2.A** |
| **CB98** | **RES** | **з.в** |
| **CB99** | **RES** | **з.с** |
| **CB9A** | **RES** | **3.D** |
| **CB9B** | **RES** | **3.E** |
| **CB9C** | **RES** | **з.н** |
| **CB9D** | **RES** | **ЗД** |
| **CB9E** | **RES** | **3,(HL)** |
| **CB9F** | **RES** | **3.A** |
|  | **RES** | **4,В \*** |
| **CBA1** | **RES** | **4.C** |
| **CBA2** | **RES** | **4,D** |
| **CBA3** | **RES** | **4.E** |
| **CBA4** | **RES** | **4.H** |
| **CBA5** | **RES** | **4Д** |
| **CBA6** | **RES** | **4,(HL)** |
| **CBA7** | **RES** | **4.A** |
| **CBA8** | **RES** | **5,В** |
| **CBA9** | **RES** | **5(C** |
| **CBAA** | **RES** | **5.D** |
| **CBAB** | **RES** | **5.E** |
| **CBAC** | **RES** | **5,H** |
| **CBAD** | **RES** | **5.L** |
| **СВАЕ** | **RES** | **5,(HL)** |
| **CBAF** | **RES** | **5,A** |
| **CBBO** | **RES** | **б,В** |
| **CBB1** | **RES** | **б.с** |
| **CBB2** | **RES** | **6.D** |
| **CBB3** | **RES** | **б.Е** |
| **CBB4** | **RES** | **б.Н** |
| **CBB5** | **RES** | **6.L** |
| **CBB6** | **RES** | **6,(HL)** |
| **CBB7** | **RES** | **6.А** |

OBJECT CODE CBB8 CBB9 CBBA CBBB CBBC CBBD CBBE CBBF CBCO CBC1 CBC2 CBC3 CBC4 CBC5

CBC6 CBC7 CBC8 CBC9 CBCA CBCB CBCC CBCD CBCE CBCF CBDO CBD1 CBD2 CBD3 CBD4 CBD5 CBD6 CBD7 CBD8 CBD9 CBDA CBDB CBDC CBDD CBDE CBDF CBEO CBE1 CBE2 СВЕЗ CBE4 CBE5 CBE6 CBE7 CBE8 CBE9 CBEA CBEB СВЕС CBED CBEE CBEF CBFO CBF1 CBF2 CBF3

RES RES RES RES RES RES RES RES SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET SET

7.B 7,C 7,D 7,E

7.H 7,L

7,(HL) 7,A 0,B O.C O.D 0,E O.H D.L D,(HL) O.A t.B l.C l.D l.E l.H l.L [.(HL) 1.A 2,В 2.C 2.D 2.E 2,H 2,L

2,(HL) 2,A 3,B 3,C 3.D 3.E 3.H 3.L 3,(HL) 3,A 4,В 4.C 4,D 4.E 4,H 4.L

4,(HL) 4,A 5,В 5,C 5.D 5.E 5.H 5Д

5,(HL) 5,A 6.B 6,C 6.D 6.E

|  |  |  |  |
| --- | --- | --- | --- |
| **OBJECT CODE** | **SOURCE CODE** | | **OBJECT CODE** |
| **CBF4** | **SET** | **6.H** | **DD2874** |
| **CBF5** | **SET** | **6.L** | **DD29** |
| **CBF6** | **SET** | **6,(HL)** | **DD2A4433** |
| **CBF7** | **SET** | **6,A** | **DD2A4433** |
| **CBF8** | **SET** | **7, В** | **DD2B** |
| **CBF9** | **SET** | **7,C** | **DD2B** |
| **CBFA** | **SET** | **7,D** | **DD2C** |
| **CBFB** | **SET** | **7.E** | **DD2D** |
| **CBFC** | **SET** | **7,H** | **DD2E66** |
| **CBFD** | **SET** | **7,L** | **DD31221144** |
| **CBFE** | **SET** | **7,(HL)** | **DD332211** |
| **CBFF** | **SET** | **7,A** | **DD3455** |
| **CC4433** | **CALL** | **Z.3344H** | **DD3555** |
| **CD4433** | **CALL** | **3344H** | **DD365566** |
| **CE66** | **. ADC** | **A.66H** | **DD39** |
| **CF** | **RST** | **08H** | **DD3B2211** |
| **DO** | **RET** | **NC** | **DD3C4433** |
| **DI** | **POP** | **DE** | **DD3D4433** |
| **D24433** | **JP** | **NC.3344H** | **DD3E443366** |
| **D366** | **OUT** | **(66H),A** | **DD44** |
| **D44433** | **CALL** | **NC.3344H** | **DD45** |
| f \* » » \*• r  **D5** | **PUSH** | **DE** | **DD4655** |
| **D666** | **SUB** | **A,66H** | **DD4C** |
| **D7** | **RST** | **10H** | **DD4D** |
| **D8** | **RET** | **C** | **DD4E55** |
| **D9** | **EXX** |  | **DD54** |
| **DA4433** | **JP** | **C.3344H** | **DD55** |
| **DB66** | **IN** | **A,(66H)** | **DD5655** |
| **DC4433** | **CALL** | **C.3344H** | **DD5C** |
| **DD014433** | **LDW** | **(HL),3344H** | **DD5D** |
| **DD03** | **INCW** | **(HL)** | **DD5E55** |
| **DD042211** | **INC** | **(SP+1122H)** | **DD60** |
| **DD052211** | **DEC** | **(SP+1122H)** | **DD61** |
| **DD06221166** | **LD** | **(SP+1122H),66H** | **DD62** |
| **DD09** | **ADD** | **IX,BC** | **DD63** |
| **DDOB** | **DECW** | **(HL)** | **DD64** |
| **DDOC** | **INC** | **(HL+IX)** | **DD65** |
| **DDOD** | **DEC** | **(HL+IX)** | **DD6655** |
| **DD0E66** | **LD** | **(HL+IX), 66H** | **DD67** |
| **DD1144339988** | **LDW** | **(3344H),8899H** | **DD68 » \* 4 •** |
| **DD134433** | **INCW** | **(3344H)** | **DD69** |
| **DD14** | **INC** | **(HL+IY)** | **DD6A** |
| **DD15** | **DEC** | **(HL+IY)** | **DD6B** |
| **DD1666** | **LD** | **(HL+IY),66H** | **DD6C** |
| **DD19** | **ADD** | **IX,DE** | **DD6D** |
| **DD1B4433** | **DECW** | **(3344H)** | **DD6E55** |
| **DD1C** | **INC** | **(IX+IY)** | **DD6F** |
| **DD1D** | **DEC** | **(IX+IY)** | **DD7055** |
| **DD1E66** | **LD** | **(IX+IY),66H** | **DD7155** |
| **DD2074** | **JAR** | **77H** | **DD7255** |
| **DD214433** | **LD** | **IX,3344H** | **DD7355** |
| **DD214433** | **LDA** | **IX,(3344H)** | **DD7455** |
| **DD214433** | **LDW** | **IX.3344H** | **DD7555** |
| **DD224433** | **LD** | **(3344H),IX** | **DD7755** |
| **DD224433** | **LDW** | **(3344H ,IX** | **DD782211** |
| **DD23** | **INC** | **IX** | **DD79** |
| **DD23** | **INCW** | **IX** | **DD7A** |
| **DD24** | **INC** | **IXH** | **DD7B** |
| **DD25** | **DEC** | **IXH** | **DD7C** |
| **DD2666** | **LD** | **IXH.66H** | **DD7D** |

«Mi

SOURCE CODE

JAF ADD LD LDW DEC DECW INC DEC LD LDW INCW INC DEC LD ADD DECW INC DEC LD LD . LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD

... LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD LD

77H IX,IX IX,(3344H) IX,(3344H) IX

IX IXL IXL IXL.66H (PC+1122H),3344H. (PC+1122H) (IX+55H) (IX+55H) (IX+55H),66H IX,SP (PC+1122H) (3344H) (3344H) (3344H),66H B.IXH B.IXL ВД1Х+55Н) C.IXH C,IXL C,(IX+55H) D.IXH D,IXL D,(IX+55H) E.IXH E,IXL E,(IX+55H) IXH.B IXH.C

IXH.D IXH.E IXH.IXH IXH,IXL

H,(IX+55H) IXH,A IXL,В IXL,C

IXL.D IXL,E IXL.IXH IXL,IXL L,(IX+55H) IXL,A (IX+55H),B (IX+55H),C (IX+55H),D (IX+55H),E (IX+55H),H (IX+55H),L (IX+55H),A A,(SP+1122H) A,(HL+IX) A,(HL+IY) A,(IX+IY) A,IXH A,IXL

OBJECT CODE DD7E55 DD802211 DD81 DD82 DD83 DD84 DD85 DD8655 DD874433 DD882211 DD89 DD8A DD8B DD8C DD8D DD8E55 DD8F4433 DD902211 DD91 DD92 DD93 DD94 DD95 DD9655 DD974433 DD982211 DD99 DD9A DD9B DD9C DD9D DD9E55 DD9F4433 DDA02211 DDA1 DDA2 DDA3 DDA4 DDA5 DDA655 DDA74433 DDA82211 DDA9 DDAA DDAB DDAC DDAD DDAE55 DDAF4433 DDB02211 DDB1 DDB2 DDB3 DDB4 DDB5 DDB655 DDB74433 DDB82211 DDB9 DDBA

OBJECT CODE DDBB DDBC DDBD DDBE55 DDBF4433 DDC1 DDC2 DDC4 DDC5 DDCA DDCB5506 DDCB55OE DDCB5516 DDCB551E DDCB5526 DDCB552E DDCB5536 DDCB553E DDCB5546 DDCB554E DDCB5556 DDCB555E DDCB5566 DDCB556E DDCB5576 DDCB557E DDCB5586 DDCB558E DDCB5596 DDCB559E DDCB55A6 DDCB55AE DDCB55B6 DDCB55BE DDCB55C6 DDCB55CE DDCB55D6 DDCB55DE DDCB55E6 DDCB55EE DDCB55F6 DDCB55FE DDCC DDCD DDD14433 DDD2 DDD4 DDD54433 DDDA DDDC DDE1 DDE2 DDE3 DDE4 DDE5 DDE9 DDEA DDEB DDEC DDED022211

SOURCE CODE LD АД1Х+55Н)

ADD A,(SP+1122H) ADD A,(HL+IX) ADD AJHL+IY) ADD A,(IX+IY) ADD AJXH ADD AJXL ADD АД1Х+55Н) ADD АД3344Н) ADC A,(SP+1122H) ADC A,(HL+IX) ADC A,(HL+IY) ADC A,(IX+IY) ADC AJXH ADC A,IXL

ADC АД1Х+55Н) ADC АД3344Н)

SUB A,(SP+1122H) SUB A,(HL+IX) SUB A,(HL+IY) SUB A,(IX+IY) SUB AJXH SUB A,IXL

SUB АД1Х+55Н) SUB АД3344Н) SBC A,(SP+1122H) SBC A,(HL+IX) SBC A,(HL+IY) SBC A,(IX+IY) SBC AJXH SBC AJXL SBC АД1Х+55Н) SBC АД3344Н)

AND A,(SP+1122H) AND A,(HL+IX) AND A,(HL+IY) AND A,(IX+IY) AND AJXH AND AJXL AND АД1Х+55Н) AND АД3344Н)

XOR A,(SP+1122H) XOR A,(HL+IX) XOR AJHL+IY) XOR A,(IX+IY) XOR AJXH XOR AJXL XOR АД1Х+55Н) XOR АД3344Н) OR A,(SP+1122H) OR A,(HL+IX) OR A,(HL+IY) OR A,(IX+IY) OR AJXH OR AJXL OR АД1Х+55Н) OR АД3344Н)

CP A,(SP+1122H) CP A,(HL+IX) CP A,(HL+IY)

SOURCE CODE CP A,(IX+IY)

CP AJXH

CP AJXL

СР АД1Х+55Н)

СР АД3344Н)

POP (HL)

JP NZ,(HL)

CALL NZ.(HL)

PUSH (HL)

JP Z,(HL)

RLC (IX+55H)

RRC (IX+55H)

RL (IX+55H)

RR (IX+55H)

SLA (IX+55H)

SRA (IX+55H)

TSET (IX+55H)

SRL (IX+55H)

BIT 0Д1Х+55Н)

BIT 1Д1Х+55Н)

BIT 2Д1Х+55Н)

BIT ЗД1Х+55Н)

BIT 4Д1Х+55Н)

IT 5Д1Х+55Н)

BIT 6Д1Х+55Н)

BIT 7Д1Х+55Н)

RES 0Д1Х+55Н)

RES 1Д1Х+55Н)

RES 2Д1Х+55Н)

RES ЗД1Х+55Н)

RES 4Д1Х+55Н)

RES 5Д1Х+55Н)

RES 6Д1Х+55Н) RES 7Д1Х+55Н)

SET 0Д1Х+55Н)

SET 1Д1Х+55Н)

SET 2Д1Х+55Н)

SET ЗД1Х+55Н) SET 4Д1Х+55Н)

SET 5Д1Х+55Н) SET 6Д1Х+55Н)

SET 7Д1Х+55Н) CALL Z,(HL) CALL (HL) POP (3344H) JP NC,(HL) CALL NC,(HL) PUSH (3344H) JP C,(HL) CALL C,(HL) POP IX JP PO,(HL) EX (SP)JX CALL PO,(HL) PUSH IX

JP (IX) JP PE,(HL) EX IX,HL CALL PE.(HL) LDA IX,(SP+1122H)

OBJECT CODE DDED042211 DDED042211 DDED052211 DDED052211 DDED0655 DDED0655 DDED072211 DDEDOA DDEDOC DDEDOC DDEDOD DDEDOD DDED0E55 DDED0E55 DDEDOF DDED12 DDED14 DDED14 DDED15 DDED15 DDED1655 DDED1655 DDED17 DDED1A DDED1C DDED1C DDED1D DDED1D DDED1E55 DDED1E55 DDED1F DDED222211 DDED242211 DDED242211 DDED252211 DDED252211 DDED2655 DDED2655 DDED27 DDED2A2211 DDED2C2211 DDED2C2211 DDED2D2211 DDED2D2211 DDED2E55 DDED2E55 DDED2F DDED322211 DDED342211 DDED342211 DDED352211 DDED352211 DDED3655 DDED3655 DDED3755 DDED3A2211 DDED3C2211 DDED3C2211 DDED3D2211 DDED3D2211

OBJECT CODE DDED3E55 DDED3E55 DDED3F4433 DDED402211 DDED412211 DDED42 DDED48 DDED49 DDED4A DDED50 DDED51 DDED52 DDED58 DDED59 DDED5A DDED60 DDED61 DDED62 DDED66 DDED68

DDED69

DDED6A DDED6D DDED6E DDED72 DDED784433 DDED794433 DDED7A DDED8655 DDED87 DDED8E55 DDED8F DDED9655 DDED9E55 DDEDC02211 DDEDC12211 DDEDC2 DDEDC3 DDEDC42211 DDEDC52211 DDEDC6 DDEDC7 DDEDC8 DDEDC9 DDEDCA DDEDCB DDEDCC DDEDCD DDEDCE DDEDDO DDEDD1 DDEDD24433 DDEDD34433 DDEDD4 DDEDD5 DDEDD64433 DDEDD74433 DDEDD8 DDEDD9 DDEDDA4433

SOURCE CODE

LD IX,(SP+1122H)

LDW IX,(SP+1122H)

LD (SP+1122H),IX

LDW (SP+1122H),IX

LD ВСД1Х+55Н)

LDW ВСД1Х+55Н)

EX A,(SP+H22H)

LDA IXJHL+IX)

LD IX,(HL+IX)

LDW IX,(HL+IX)

LD (HL+IX),IX

LDW (HL+lxj.lX

LD (IX+55H),BC

LDW (IX+55H),BC

EX A,(HL+IX)

LDA IX,(HL+IY)

LD IX,(HL+IY)

LDW IX,(HL+IY)

LD (HL+IY),IX

LDW (HL+IY),IX

LD DE,(IX+55H)

LDW DE,(IX+55H) "

EX A,(HL+IY)

LDA IX,(IX+IY)

LD IX,(IX+IY)

LDW IX,(IX+IY)

LD (IX+IY),IX

LDW (IX+IY),IX

LD (IX+55H),DE

LDW (IX+55H),DE

EX A,(IX+IY)

LDA 1ХДРС+1122Н)

LD 1ХДРС+1122Н)

LDW 1ХДРС+1122Н)

LD (PC+1122H),IX

LDW (PC+1122H),IX

LD HL,(IX+55H)

LDW HL,(iX+55H) EX A,IXH

LDA 1ХД1Х+1122Н)

LD 1ХД1Х+1122Н)

LDW 1ХД1Х+1122Н)

LD (IX+1122H),IX

LDW (IX+1122H),IX

LD (IX+55H),HL

LDW (IX+55H),HL

EX A,IXL

LDA IX,(IY+1122H)

LD IX,(IY+1122H)

LDW IX,(IY+1122H)

LD (IY+1122H),IX

LDW (IY+1122H),IX

LD SP,(IX+55H)

LDW SP,(IX+55H)

EX АД1Х+55Н)

LDA IX,(HL+1122H)

LD IX,(HL+1122H)

LDW IX,(HL+1122H)

LD (HL+1122H),IX

LDW (HL+1122H),IX

SOURCE CODE LD (IX+55H),SP LDW (IX+55H),SP EX АД3344Н) IN (SP+1122H),(C) OUT (C),(SP+1122H) SBC IX.BC

IN (HL+IX),(C) OUT (C),(HL+IX) ADC IX,BC IN (HL+IY),(C) OUT (C),(HL+IY) SBC IX,DE IN (IX+IY),(C) OUT (C),(IX+IY) ADC IX,DE IN 1ХНДС) OUT (C),IXH SBC IX,IX LDCTL IX,(C) IN IXL,(C) OUT (C),IXL ADC IX,IX ADD IX,A LDCTL (C),IX SBC IX,SP IN (3344H),(C) OUT (C),(3344H) ADC IX,SP LDUD АД1Х+55Н) LDCTL IX.USP LDUD (IX+55H),A LDCTL USP,IX LDUP АД1Х+55Н) LDUP (IX+55H),A MULT A,(SP+1122H) MULTU A,(SP+H22H) MULTW HL,(HL) MULTUW HL,(HL) DIV HL,(SP+1122H) DIVU HL,(SP+1122H) ADDW HL,(HL) CPW HL,(HL) MULT A,(HL+IX) MULTU A,(HL+IX) DIVW DEHL,(HL) DIVUW DEHL,(HL) DIV HL,(HL+IX) DIVU HL,(HL+IX) SUBW HL,(HL) MULT A,(HL+IY) MULTU A,(HL+IY) MULTW HL,(3344H)

MULTUW HL,(3344H)

DIV HL,(HL+IY) DIVU HL,(HL+IY) ADDW HL,(3344H) CPW HL,(3344H) MULT A,(IX+IY) MULTU A,(IX+IY) DIVW DEHL,(3344H)

OBJECT CODE DDEDDB4433 DDEDDC DDEDDD DDEDDE4433 DDEDEO DDEDE1 DDEDE2 DDEDE3 DDEDE4 DDEDE5 DDEDE6 DDEDE7 DDEDE8 DDEDE9 DDEDEA DDEDEB DDEDEC DDEDED DDEDEE DDEDF055 DDEDF155 DDEDF22211 DDEDF32211 DDEDF455 DDEDF555 DDEDF62211 DDEDF72211 DDEDF84433 DDEDF94433 DDEDFA2211 DDEDFB2211 DDEDFC4433 DDEDFD4433 DDEDFE2211 DDF12211 DDF2 DDF4 DDF52211 DDF9 DDF9 DDFA DDFC DE66 DF EO El E24433 E3 E44433 E5 E666 E7 E8 E9 EA4433 EB EC4433 ED022211 ED032211 ED042211

SOURCE CODE DIVUW DEHL,(3344H) DIV HL,(IX+IY) DIVU HL,(IX+IY) SUBW HL,(3344H) MULT A,IXH MULTUAJXH MULTW HL,IX

MULTUW HL,IX

DIV HL.IXH DIVU HL.IXH ADDW HL,IX CPW HL,IX MULT A,IXL

MULTU A,IXL DIVW DEHL,IX DIVUW DEHL,IX DIV HL,IXL DIVU HL,IXL SUBW HL,IX MULT A,(IX+55H) MULTU A,(IX+55H)

MULTW HL,(PC+1122H)

MULTUW HL,(PC+1122H)

DIV HL,(IX+55H) DIVU HL,(IX+55H) ADDW HL,(PC+1122H) CPW HL,(PC+1122H) MULT A,(3344H) MULTU A,(3344H)

DIVW DEHL,(PC+1122H) DIVUW DEHL,(PC+1122H)

DIV HL,(3344H) DIVU HL.(3344H)

SUBW HL,(PC+1122H) POP (PC+1122H) JP P.(HL) CALL P,(HL) 1 PUSH (PC+1122H) LDW SP.IX LD SP,IX JP M,(HL) CALL M,(HL) SBC A.66H RST 18H RET PO POP HL JP PO.3344H EX (SP),HL CALL PO.3344H PUSH HL AND A.66H RST 20H RET PE JP (HL) JP PE.3344H EX DE,HL CALL PE.3344H LDA HL,(SP+1122H)

LD (SP+1122H),A LD HL,(SP+1122H)

OBJECT CODE ED042211 ED052211 ED052211 ED06 ED06 ED07 . EDOA EDOB EDOC EDOC EDOD EDOD EDOE EDOE EDOF ED12 ED13 EDU .... ED14 ED15 ED15 ED16 ED16 ED17 ED1A ED1B ED1C ED1C ED1D ED1D EDIE EDIE EDIF ED222211 ED232211 ED242211 ED242211 ” \*

ED252211 ED252211 ED26 ED26 ED27 ED2A2211 ED2B2211 ED2C2211 ED2C2211 ED2D2211 ED2D2211 ED2E ED2E ED2F ED322211 ED332211 ED342211 ED342211 ED352211 ED352211 ED36 ED36 ED37

LDW LD LDW LD LDW EX LDA LD LD LDW LD LDW LD LDW EX LDA LD LD LDW LD LDW LD LDW EX LDA LD LD LDW LD LDW LD LDW EX LDA LD LD LDW LD LDW LD LDW EX LDA LD LD LDW LD LDW LD LDW EX LDA LD LD LDW LD LDW LD LDW EX

HL,(SP+1122H) (SP+1122H),HL (SP+1122H),HL BC.(HL) BC,(HL) A,В HL,(HL+IX) (HL+IX),A HL,(HL+IX) HL,(HL+IX) (HL+IX),HL (HL+IX),HL (HL).BC (HL).BC A,C HL,(HL+IY) (HL+IY),A HL,(HL+IY) HL,(HL+IY) (HL+IY),HL (HL+IY),HL DE,(HL) DE,(HL) A.D HL,(IX+IY) (IX-t-IY).A HL,(IX+IY) HL,(IX+IY) (IX+IY),HL (IX+IY),HL (HL),DE (HL),DE A,E HL,(PC+1122H) (PC+1122H),A HL,(PC+1122H) HL,(PC+U22H) (PC+1122H),HL (PC+1122HJ.HL HL,(HL) HL,(HL) A.H HL,(IX+1122H) (IX+1122H),A HL,(IX+1122H) HL,(IX+1122H) (IX+1122H),HL (IX+1122H),HL (HL).HL (HL),HL A,L

HL,(IY+1122H) (IY+1122H),A HL,(IY+U22H) HL,(IY+1122H) (IY+1122H),HL (IY+1122H),HL SP,(HL) SP.(HL) A,(HL)

OBJECT CODE ED3A2211 ED3B2211 ED3C2211 ED3D2211 ED3D2211 ED3E ED3E ED3F ED40 ED41 ED42 ED434433 ED434433 ED44 ED45 ED46 ED47 ED48 ED49 ED4A ED4B4433 ED4B4433 ED4C ED4D ED4E ED4F ED50 ED51 ED52 ED534433 ED534433 ED55 ED56 ED57 ED58 ED59 ED5A ED5B4433 ED5B4433 ED5E ED5F ED60 ED61 ED62 ED64 ED65 ED66 ED67 ED68 ED69 ED6A ED6C ED6D ED6E ED6F ED70 ED714433 ED72 ED734433 ED734433

OBJECT CODE ED7766 ED78 ED79 ED7A ED7B4433 ED7B4433 ED7F66 ED82 ED83 ED842211 ED852211 ED86 ED87 ED8A ED8B ED8C ED8D ED8E ED8F ED92 ED93 ED94 ED95 ED96

ED97 ED9A ED9B ED9C ED9D ED9E ED9F EDAO EDAI

EDA2 EDA3 EDA42211 EDA52211

EDA6 EDA74433 EDA8

EDA9 EDAA EDAB EDAC2211 EDAD2211 EDAE EDAF4433

EDBO EDB1 EDB2 EDB3 EDB42211 EDB52211

EDB7 EDB7 EDB8 EDB9 EDBA EDBB EDBC2211

SOURCE CODE LDA HL,(HL+1122H) LD (HL+1122H),A

LDW HL,(HL+1122H) LD (HL+1122H),HL

LDW (HL+1122H),HL LD (HL),SP

LDW (HL),SP

EX A,A

IN B,(C)

OUT (C),B

SBC HL,BC

LD (3344H),BC LDW (3344H),BC NEG A RETN IM 0 LD I,A IN C,(C) OUT (C),C ADC HL,BC LD ВСД3344Н) LDW ВСД3344Н) NEG HL RETI IM 3 LD R,A IN D,(C) OUT (C),D SBC HL,DE LD (3344H),DE LDW (3344H),DE RETIL IM 1 LD A,I IN E,(C) OUT (C),E ADC HL,DE LD DE,(3344H) LDW DE,(3344H) IM .... 2 LD A,R IN НДС) OUT (C),H SBC HL,HL EXTS A PCACHE LDCTL HL,(C) RRD IN L,(C) OUT (C),L ADC HL,HL EXTS HL ADD HL,A

LDCTL (C),HL RLD TSTI (C) SC 3344H

SBC HL,SP LD (3344H),SP

LDW (3344H),SP

**SOURCE CODE DI 66H**

[**IN A,(C)**](#bookmark227)

**OUT (C),A ADC HL.SP LD SP,(3344H)**

**LDW SP,(3344H) Ei 66H**

INIV7 OUTIW

EPUM (SP+1122H) MEPU (SP+1122H) LDUD A,(HL) LDCTL HL.USP INDW OUTDW EPUM (HL+IX) MEPU (HL+IX) LDUD (HL),A LDCTL USP,HL INIRW OTIRW

EPUM (HL+IY) MEPU (HL+IY) LDUP A,(HL) EPLF INDRW OTDRW EPUM (IX+IY) MEPU (IX+IY) LDUP (HL),A EPLI LDI CPI INI OUTI

EPUM (PC+1122H) MEPU (PC+1122H) EPUM (HL) EPUM (3344H) LDD CPD IND OUTD

EPUM (IX+1122H) MEPU (IX+1122H) MEPU (HL) MEPU (3344H) LDIR CPIR INIR OTIR

EPUM (IY+1122H) MEPU (IY+1122H) IN HL,(C) INW HL,(C) LDDR

CPDR INDR OTDR

EPUM (HL+1122H)

OBJECT CODE EDFB EDFC EDFD EDFE EE66 EF FO Fl F24433 F3 F44433 F5 F666 F7

F8 F9 F9 FA4433 FB FC4433 FD032211 FD042211 FD052211 FD06221166 FD09 FD0B2211 FD0C2211 FD0D2211 FD0E221166 FD132211 FD142211 FD152211 FD16221166 FD19 FD1B2211 FD1C2211 FD1D2211 FD1E221166 FD214433 FD214433 FD214433 FD224433 FD224433 FD23 FD23 FD24 FD25 FD2666 FD29 FD2A4433 FD2B FD2B FD2C FD2D FD2E66 FD3455 FD3555 FD365566 FD39 FD44

OBJECT CODE EDBD2211 EDBF EDBF EDCO EDC1 EDC2 EDC3 EDC4 EDC5 EDC6 EDC7 EDC8 EDC9 EDCA EDCB EDCC EDCD

EDCE .......

EDDO EDD1 EDD2 EDD3 EDD4 EDD5 EDD6 EDD7 EDD8 EDDO EDDA EDDB EDDC EDDD EDDE EDEO EDEI EDE2

EDE3 ■ r

EDE4 EDE5

EDE6 EDE7 EDE8 EDE9 EDEA EDEB EDEC EDED EDEE EDEF EDFO EDF1 EDF2 EDF3 EDF4 EDF5 EDF6 EDF7 EDF8 EDF9 EDFA

SOURCE CODE MEPU (HL+1122H) OUT (C),HL OUTW (C),HL MULT A,В MULTUA.B MULTW HL.BC

MULTUW HL.BC

DIV HL,В

DIVU HL,В ADDW HL.BC CPW HL.BC MULT A.C MULTUA.C DIVW DEHL.BC DIVUW DEHL.BC DIV HL,C DIVU HL,C SUBW HL.BC MULT A,D MULTUA.D

MULTW HL,DE

MULTUW HL,DE

DIV HL,D DIVU HL.D ADDW HL,DE CPW HL,DE MULT A.E MULTUA.E DIVW DEHL.DE DIVUW DEHL.DE DIV HL,E DIVU HL,E SUBW HL.DE MULT A.H MULTUA.H

MULTW HL,HL

MULTUW HL.HL

DIV HL.H DIVU HL.H ADDW HL,HL CPW HL,HL MULT A.L MULTUA.L DIVW DEHL.HL DIVUW DEHL.HL DIV HL,L DIVU HL.L SUBW HL.HL EX H.L MULT A,(HL) MULTUA.(HL) MULTW HL.SP

MULTUW HL.SP

DIV HL.(HL) DIVU HL,(HL) ADDW HL.SP CPW HL.SP MULT A.A MULTUA.A DIVW DEHL.SP

DIVUW DEHL.SP DIV HL,A

DIVU HL,A SUBW HL.SP XOR A.66H RST 28H RET P POP AF JP P.3344H DI

CALL P.3344H PUSH AF

OR A.66H RST ЗОН RET M LDW SP.HL LD SP.HL

JP M.3344H

El CALL M.3344H INCW (IX+1122H) INC (PC+1122H) DEC (PC+1122H) LD (PC+1122H),66H ADD IY.BC DECW (IX+1122H) INC (IX+1122H) DEC (IX+1122H) LD (IX+1122H),66H INCW (IY+1122H) INC (IY+1122H) DEC (IY+1122H) LD (IY+1122H),66H ADD IY,DE DECW (IY+1122H) INC (HL+1122H) DEC (HL+1122H) LD (HL+1122H),66H

LD IY.3344H

LDA IY,(3344H) LDW IY.3344H LD (3344H),IY LDW (3344H),IY INC IY INCW IY INC IYH DEC IYH LD IYH.66H ADD IY.IY LDW IY,(3344H) DEC IY DECW IY INC IYL DEC IYL LD IYL.66H INC (IY+55H) DEC (IY+55H) LD (IY+55H),66H

ADD IY,SP LD B.1YH

|  |  |  |
| --- | --- | --- |
| **OBJECT CODE** | **SOURCE CODE** | |
| **FD45** | **LD** | **B.IYL** |
| **FD4655** | **LD** | **B,(IY+55H)** |
| **FD4C** | **LD** | **C,IYH** |
| **FD4D** | **LD** | **C,IYL** |
| **FD4E55** | **LD** | **C,(IY+55H)** |
| **FD54** | **LD** | **D,IYH** |
| **FD55** | **LD** | **D,IYL** |
| **FD5666** | **LD** | **D,(IY+55H)** |
| **FD5C** | **LD** | **E.IYH** |
| **FD5D** | **LD** | **E.IYL** |
| **FD5E55** | **LD** | **E,(IY+55H)** |
| **FD60** | **LD** | **IYH,В** |
| **FD61** | **LD** | **IYH,C** |
| **FD62** | **LD** | **IYH.D** |
| **FD63** | **LD** | **IYH,E** |
| **FD64** | **LD** | **IYH,IYH** |
| **FD65** | **LD** | **IYH,IYL** |
| **FD6655** | **LD** | **H,(IY+55H)** |
| **FD67** | **LD** | **IYH,A** |
| **FD68** | **LD** | **IYL,В** |
| **FD69** | **LD** | **IYL.C** |
| **FD6A** | **LD** | **IYL,D** |
| **FD6B** | **LD** | **IYL,E** |
| **FD6C** | **LD** | **IYL,IYH** |
| **FD6D** | **LD** | **IYL,IYL** |
| **FD6E55** | **LD** | **L,(IY+55H)** |
| **FD6F** | **LD** | **IYL,A** |
| **FD7055** | **LD** | **(IY+55H),B** |
| **FD7155** | **LD** | **(IY+55H),C** |
| **FD7255** | **LD** | **(IY+55H),D** |
| **FD7355** | **LD** | **(IY+55H),E** |
| **FD7455** | **LD** | **(IY+55H),H** |
| **FD7555** | **LD** | **(IY+55H),L** |
| **FD7755** | **LD** | **(IY+55H),A** |
| **FD782211** | **LD** | **A,(PC+U22H)** |
| **FD792211** | **LD** | **АД1Х+1122Н)** |
| **FD7A2211** | **LD** | **A,(IY+1122H)** |
| **FD7B2211** | **LD** | **A,(HL+1122H)** |
| **FD7C** | **LD** | **A,IYH** |
| **FD7D** | **LD** | **A,IYL** |
| **FD7E55** | **LD** | **A,(IY+55H)** |
| **FD802211** | **ADD** | **A,(PC+1122H)** |
| **FD812211** | **ADD** | **A,(IX+1122H)** |
| **FD822211** | **ADD** | **A,(IY+1122H)** |
| **FD832211** | **ADD** | **A,(HL+1122H)** |
| **FD84** | **ADD** | **A,IYH** |
| **FD85** | **ADD** | **A,IYL** |
| **FD8655** | **ADD** | **AJIY+55H)** |
| **FD882211** | **ADC** | **A,(PC+1122H)** |
| **FD892211** | **ADC** | **АД1Х+1122Н)** |
| **FD8A2211** | **ADC** | **A,(IY+U22H)** |
| **FD8B2211** | **ADC** | **A,(HL+1122H)** |
| **FD8C** | **ADC** | **A,IYH** |
| **FD8D** | **ADC** | **A,IYL** |
| **FD8E55** | **ADC** | **A,(IY+55H)** |
| **FD902211** | **SUB** | **АДРС+1122Н)** |
| **FD912211** | **SUB** | **A,(IX+1122H)** |
| **FD922211** | **SUB** | **A,(IY+1122H)** |
| **FD932211** | **SUB** | **A,(HL+1122H)** |
| **FD94** | **SUB** | **A,IYH** |

FD95 FD9655 FD982211 FD992211 FD9A2211 FD9B2211 FD9C FD9D FD9E55 FDA02211 FDA12211 FDA22211 FDA32211 FDA4 FDA5 FDA655 FDA82211 FDA92211 FDAA2211 FDAB2211 FDAC FDAD FDAE55 FDB02211 FDB12211 FDB22211 FDB32211 FDB4 FDB5 FDB655 FDB82211 FDB92211 FDBA2211 FDBB2211 FDBC FDBD FDBE55 FDC22211 FDC32211 FDC42211 FDCA2211 FDCB5506 FDCB550E FDCB5516 FDCB551E FDCB5526 FDCB552E FDCB5536 FDCB553E FDCB5546 FDCB554E FDCB5556 FDCB555E FDCB5566 FDCB556E FDCB5576 FDCB557E FDCB5586 FDCB558E FDCB5596

SUB SUB SBC SBC SBC SBC SBC SBC SBC AND AND AND AND AND AND AND XOR XOR XOR XOR ., XOR

XOR XOR OR OR OR OR OR OR OR CP CP CP CP CP CP CP JP JP CALL JP RLC RRC RL RR SLA SRA TSET SRL BIT BIT BIT BIT BIT BIT BIT BIT RES RES RES

A,IYL A,(IY+55H) A,(PC+U22H) A,(IX+U22H) A,(IY+1122H) A,(HL+1122H) A,IYH A,IYL A,(IY+55H) АДРС+1122Н) A,(IX+1122H) A,(IY+1122H) A,(HL+1122H) A,IYH A,IYL A,(IY+55H) АДРС+1122Н) АД1Х+1122Н) A,(IY+1122H) A,(HL+1122H) A,IYH A,IYL A,(IY+55H) АДРС+1122Н) A,(IX+1122H) A,(IY+1122H) A,(HL+1122H) A,IYH A,IYL A,(IY+55H) АДРС+1122Н) АД1Х+1122Н) A,(IY+U22H) A,(HL+1122H) A,IYH A,IYL A,(IY+55H) NZ,(PC+U22H) (PC+1122H) NZ,(PC+1122H) Z,(PC+U22H) (IY+55H) (IY+55H) (IY+55H) (IY+55H) (IY+55H) (IY+55H) (IY+55H) (IY+55H) 0,(IY+55H) 1,(IY+55H) 2,(IY+55H) 3,(IY+55H) 4,(IY+55H) 5,(IY+55H) 6,(IY+55H) 7,(IY+55H) 0,(IY+55H) 1,(IY+55H) 2,(IY+55H)

OBJECT CODE FDCB559E FDCB55A6 FDCB55AE FDCB55B6 FDCB55BE FDCB55C6 FDCB55CE FDCB55D6 FDCB55DE FDCB55E6 FDCB55EE FDCB55F6 FDCB55FE FDCC2211 FDCD2211 FDD22211 FDD42211 FDDA2211 FDDC2211 FDE1 FDE22211 FDE3 FDE42211 FDE5

FDE9 FDEA2211 FDEB FDEC2211 FDED022211 FDED042211 FDED042211 FDED052211 FDED052211 FDED0655 FDED0655 FDED072211 FDEDOA FDEDOC FDEDOC FDEDOD FDEDOD FDEDOE55 FDEDOE55 FDED0F2211 FDED12 FDED14 FDED14 FDED15 FDED15 FDED1655 FDED1655 FDED172211 FDED1A FDED1C FDED1C FDED1D FDED1D FDED1E55 FDED1E55 FDED1F2211

OBJECT CODE FDED222211 FDED242211 FDED242211 FDED252211 FDED252211 FDED2655 FDED2655 FDED27 FDED2A2211 FDED2C2211 FDED2C2211 FDED2D2211 FDED2D2211 FDED2E55 FDED2E55 FDED2F FDED322211 FDED342211 FDED342211 FDED352211 FDED352211 FDED3655 FDED3655 FDED3755

FDED3A2211 FDED3C2211 FDED3C2211 FDED3D2211 FDED3D2211 FDED3E55 FDED3E55

FDED402211 FDED412211 FDED42 FDED482211 FDED492211 FDED4A FDED502211 FDED512211 FDED52 FDED582211 FDED592211 FDED5A FDED60 FDED61 FDED62 FDED66 FDED68 FDED69 FDED6A FDED6D FDED6E FDED72 FDED7A FDED8655 FDED87 FDED8E55 FDED8F FDED9655 FDED9E55

SOURCE CODE

RES 3,(IY+55H)

RES 4,(IY+55H)

RES 5,(IY+55H)

RES 6,(IY+55H)

RES 7,(IY+55H)

SET 0,(IY+55H)

SET 1,(IY+55H)

SET 2,(IY+55H)

SET 3,(IY+55H)

SET 4,(IY+55H)

SET 5,(IY+55H)

SET 6,(IY+55H)

SET 7,(IY+55H)

CALL Z,(PC+1122H)

CALL (PC+1122H)

JP NC,(PC+1122H)

CALL NC,(PC+U22H)

JP C,(PC+1122H)

CALL C,(PC+1122H) POP IY

JP PO,(PC+1122H)

EX (SP)JY

CALL PO,(PC+1122H) PUSH IY

JP (IY)

JP PE,(PC+1122H)

EX IY,HL

CALL PE,(PC+1122H)

LDA IY,(SP+1122H)

LD IY,(SP+1122H)

LDW IY,(SP+U22H)

LD (SP+1122H),IY

LDW (SP+1122H),IY

LD BC,(IY+55H)

LDW BC,(IY+55H)

EX АДРС+1122Н)

LDA IY,(HL+IX)

LD IY,(HL+IX)

LDW IY,(HL+IX) LD (HL+IX),IY

LDW (HL+IX),IY

LD (IY+55H),BC

LDW (IY+55H),BC

EX АД1Х+1122Н)

LDA IY,(HL+IY)

LD IY,(HL+IY)

LDW IY,(HL+IY)

LD (HL+IY),IY

LDW (HL+IY),IY

LD DE,(IY+55H)

LDW DE,(IY+55H)

EX A,(IY+1122H)

LDA IY,(IX+IY)

LD IY,(IX+IY)

LDW IY,(IX+IY)

LD (IX+IY),IY

LDW (IX+IY),IY

LD (IY+55H),DE

LDW (IY+55H),DE

EX A,(HL+1122H)

LDA IY,(PC+1122H)

LD IY,(PC+1122H)

LDW IY,(PC+1122H)

LD (PC+1122H) ,1Y

LDW (PC+1122H),IY

LD HL,(IY+55H)

LDW HL,(IY+55H)

EX A.IYH

LDA IY,(IX+1122H)

LD IY,(IX+1122H)

LDW IY,(IX+1122H)

LD (IX+1122H),IY

LDW (IX+1122H),IY

LD (IY+55H),HL

LDW (IY+55H),HL

EX A.IYL

LDA IY,(IY+1122H)

LD IY,(IY+1122H)

LDW I Y,(IY+1122H)

LD (IY+1122H),IY

LDW jlY+1122H),IY

LD SP,(IY+55H)

LDW SP,(IY+55H)

EX A,(IY+55H)

LDA IY,(HL+1122H)

LD IY,(HL+1122H)

LDW IY,(HL+1122H)

LD (HL+1122H),IY

LDW (HL+1122H),IY

LD (IY+55H),SP

LDW (IY+55H),SP

IN (PC+1122H) ,(C)

OUT (C),(PC+1122H)

SBC IY,BC

IN (1X+1122H) ,(C)

OUT (C),(IX+1122H)

ADC IV,BC IN (I Y+1122H) ,(C)

OUT (C),(l Y+1122H)

SBC IV,DE

IN (HL+1122H) ,(C)

OUT (C),(HL+1122H)

ADC IY,DE

IN IYH,(C)

OUT (C),IYH

SBC IY,IY

LDCTL IY,(C)

IN IYL,(C)

OUT (C),IYL

ADC IY,IY

ADD IY,A

LDCTL (C),IY SBC IY,SP ADC IY,SP LDUD A,(IY+55H)

LDCTL IY.USP LDUD (I7+55H),A

LDCTL USP.IY LDUP A (IY+55H) LDUP (IY+55H),A

OBJECT CODE FDEDC02211 FDEDC12211 FDEDC22211 FDEDC32211 FDEDC42211 FDEDC52211 FDEDC62211 FDEDC72211 FDEDC82211 FDEDC92211 FDEDCA2211 FDEDCB2211 FDEDCC2211 FDEDCD2211 FDEDCE2211 FDEDD02211 FDEDD12211 FDEDD22211 FDEDD32211 FDEDD42211 FDEDD52211 FDEDD62211 FDEDD72211 FDEDD82211 FDEDD92211 FDEDDA2211 FDEDDB2211 FDEDDC2211 FDEDDD2211 FDEDDE2211 FDEDEO FDEDE1 FDEDE2 FDEDE3 FDEDE4

SOURCE CODE

OBJECT CODE

SOURCE CODE

MULT A,(PC+1122H) MULTU АДРС+И22Н) MULTW HL,(IX+1122H)

MULTUW HLI(IX+1122H)

DIV HL,(PC+1122H)

DIVU HL.(PC+U22H) ADDW HL,(IX+1122H) CPW HL,(IX+1122H) MULT A,(IX+1122H) MULTU A,(IX+1122H) DIVW DEHL,(IX+1122H) DIVUW DEHL,(IX+1122H)

DIV HL,(IX+U22H) DIVU HL,(IX+1122H) SUBW HL.(IX+1122H) MULT A,(IY+1122H) MULTU A.(IY+1122H) MULTW HL,(IY+1122H)

MULTUW HL,(IY+1122H)

DIV HL,(IY+H22H) ....... DIVU HL,(IY4-1122H)

ADDW HL,(IY+1122H) CPW HL,(IY+1122H)

MULT A,(HL+1122H) MULTU A,(HL+1122H) DIVW DEHL,(IY+1122H) DIVUW DEHL,(IY+1122H) , DIV HL,(HL+U22H) DIVU HL,(HL+1122H) SUBW HL,(IY+1122H) MULT A.IYH MULTU A.IYH MULTW HL.IY

MULTUW HL.IY

DIV HL.IYH

FDEDE5 FDEDE6 FDEDE7 FDEDE8 FDEDE9 FDEDEA FDEDEB FDEDEC FDEDED FDEDEE FDEDF055 FDEDF155 FDEDF24433 FDEDF34433

' FDEDF455 FDEDF555 FDEDF64433 FDEDF74433 FDEDF866 FDEDF966 FDEDFA4433 : >

FDEDFB4433 FDEDFC66 FDEDFD66 FDEDFE4433 FDF22211 FDF42211 FDF54433 FDF9 FDF9 FDFA2211 FDFC2211 FE66 FF

DIVU HL.IYH ADDW HL.IY CPW HL.IY MULT A.IYL MULTU A.IYL DIVW DEHL.IY DIVUW DEHL.IY DIV HL.IYL DIVU HL.IYL SUBW HL.IY MULT A.(IY+55H) MULTU A,(IY+55H) MULTW HL.3344H

MULTUW HL.3344H

DIV HL,(IY+55H) DIVU HL,(IY+55H)

ADDW HL.3344H CPW HL.3344H MULT A.66H MULTUA.66H DIVW DEHL.3344H DIVUW DEHL.3344H DIV HL.66H DIVU HL.66H SUBW HL.3344H JP P,(PC+1122H)

CALL P,(PC+1122H) PUSH 3344H

LD SP.IY LDW SP.IY

JP M,(PC+1122H)

CALL M,(PC+1122H) СР A.66H

RST 38H

Appendix E.

Instruction Timing

The Z280 CPU processes instructions using a three- stage pipeline consisting of an instruction prefetch unit, an instruction decoder, and an instruction execution unit. Each section of the pipeline operates autonomously, communicating with the other stages of the pipeline via handshakes and local buses. The pipelined architecture of the Z280 MPU greatly increases program throughput; as one instruction is being executed, the next instruction can be decoded, and the instruction after that can be fetched.

, , The autonomous operation of the three stages in,

the Z280 CPU instruction pipeline makes it difficult to calculate exact instruction execution times. Furthermore, execution times are affected by cache activity; the current cache contents determine the number of external memory transactions made during the fetch and execution of a given instruction. In this appendix, three types of tables are provided for calculation of instruction timings: instruction execution timing, instruction fetch and decode timing, and bus transaction timing. All tables list execution and transaction timings in terms of CPU clock cycles.

Tables E-1, E-2, and E-3 show the execution times for all instructions and interrupt and trap processing. Table E-1 lists the execution times for all CPU-executed instructions, with the instructions listed by functional group. Table E-2 lists the execution times for the extended- instructions. Table E-3 shows execution times for interrupt and trap events. These tables assume that the instruction has been fetched, decoded, and is ready for execution, and that the bus is idle when the execution unit makes a reguest for a transaction. Thus, the execution times shown in these tables represent the maximum execution rate of the machine. The actual execution rate will be somewhat lower than this maximum for two reasons: . (1) the execution unit must compete with the prefetch unit for use of the external bus, and (2) some instructions may take longer to prefetch and decode than the previous instruction will take to execute.

Furthermore, the activity of the execution unit can affect the prefetch unit wnen certain instructions are executed. In Tables E-1 and E-2, an ”F” on the right-hand side of the table indicates that the pipeline is flushed when that instruction is executed; the pipeline is also flushed during all interrupt and trap processing.

In these cases, the next instruction must be completely fetched and decoded before the execution unit can proceed. The execution times in these tables do not include the time necessary to fetch and decode the next instruction if the pipeline is flushed.

In Tables E-1 through E-3, execution times are given as the number of absolute CPU clock cycles plus the number and type of bus transactions. Bus transaction timings are shown separately in Tables E-5 through E-10.

Table E-4 contains the instruction fetch and decode timing, and Tables E-5 through E-10 show bus transaction timings. The CPU clock is divided by a factor of 1, 2, or 4 to form the bus clock; thus, bus transaction timing depends on the relationship between the CPU clock and bus clock. All three types of bus timing are shown in the tables. Furthermore, because of the different phase relationships between the request for a transaction and the bus clock, a variable number of cycles is included in parentheses in Tables E-4 through E-10; the average would be half of the sum of the minimum and maximum numbers listed in the parentheses. The notation V in these tables refers to the number of wait states added to the transaction (either by asserting the WAIT input or by programming the appropriate CPU control registers) in addition to any automatically inserted wait states. Again, the numbers in these tables assume that the bus is idle when the transaction request is made.

**8-BIT LOAD GROUP**

EX A,src

src = R,RX,IR.DA.X.SX

RA.SR.BX

EX KI­

LD dst,src

R,RX: 4

IR,DA,X.SX,RA.SR.BX: 5 4- rd(src) - wr(src)

**4**

dst = A •

src = R,RX,IM,IR,DA,X

SX,RA.SR.BX (BC).(DE)

R,RX: 2

IR.DA.X.SX,RA.SR.BX: 3 4- rd(src)

(BC),(DE): 3 4- rd(IR)

or

dst = R.RX.IR.DA.X

SX,RA.SR.BX (BC).(DE) src = A - :

R,RX,IM:2

IR.DA.X.SX,RA.SR.BX: 3 + wr(dst)

(BC),(DE): 3 + wr(IR)

LD dst,src

dst = R

src = R,RX,IM,IR.SX

R,RX,IM:2

IR.SX: 3 + rd(src)

LD dst.n

LDUD dst.src

or

dst = R.RX.IR.SX src = R

dst = R.RX.IR.DA.X, SX,RA.SR.BX

dst = A

src = IR,SX in user

R,RX: 2

IR.SX: 3 + wr(dst)

R,RX: 2

IR.DA.X.SX,RA.SR.BX: 3 4- wr(dst)

3 4- rd(src)

or

dst = IR.SX in user 3 4- wr(dst)

src = *A*

LDUP dst,src dst = A 3 4-rd(src)

. > v -’ *.л. .• \**t. 1 . src — IR,SX in user - \* ■ ■ ••••-» -■» - • \* -■■ •»' ■ ■»•4 ' -

or

dst = IR.SX in user src = A

' 3 4- wr(dst)

See Table E-1 Note on page E-10.

**16-BLT LOAD GROUP**

EX src,HL

src = DE,IX,IY

EX (SP),dst

dst = HL,IX,IY

EXAF.AF'

EXX

LD[W] dst.src

dst=HL,IX,IY

src = IM,DA,X,RA,SR,BX

IM: 2

DA,X,RA,SR,BX: 3 + rd(src)

or

dst = DA,X,RA,SR,BX src = HL,IX,IY

•I ■

3 + wr(dst)

LD[W] dst.src

dst = BC,DE,HL.SP src = IM.IR.DA.SX

IM: 2

IR.DA.SX: 3 + rd(src)

LD[W] dst.nn

LD[W] dst.nn

LD[W] dst.src

or

dst = IR.DA.SX

src = BC, DE, HL.SP

3 + wr(dst)

LDA dst.src

POP dst

PUSH src

dst = RR, IR, DA, RA

RR: 2

IR,DA,RA: 3 + wr(dst)

dst = RR

dst = SP

src = HL,IX,IY,IM,IR,DA,SX

HL,IX,IY,IM: 2

IR.DA.SX: 3 + rd(src)

or

dst = IR.DA.SX src = SP

3 + wr(dst)

dst = HL,IX,IY

src = DA,X,RA,SR,BX

DA: 2

X.RA.SR: 5 BX:6

dst = RR,IR,DA,RA

IR.DA.RA: 9 + rd(IR) + wr(dst)

src = RR, IM, IR, DA, RA

RR,IM: 8 + wr(IR)

IR.DA.RA: 9 + rd(src) + wr(IR)

See Table E-1 Note on page E-10.

**BLOCK TRANSFER AND SEARCH GROUP**

CPD

CPDR

8 + rd(IR), each iteration

CPI

CPIR

8 + rd(IR), each iteration

LDD

LDDR

9 + rd(IR) + wr(IR), each iteration

LDI

LDIR

9 + rd(IR) + wr(IR), each iteration

**8-BIT ARITHMETIC AND LOGIC GROUP**

ADC [A,]src

src = R,RX,IM,IR,DA, X,SX,RA,SR,BX

R,RX,IM:2

IR,DA,X,SX,RA,SR,BX: 3 + rd(src)

ADD [A,]src

src = R,RX,IM,IR,DA, X,SX,RA,SR,BX

R,RX,IM:2

IR,DA,X,SX,RA,SR,BX: 3 + rd(src)

AND[A,]src

src = R,RX,IM,IR,DA, X,SX,RA,SR,BX

R,RX,IM:2

IR,DA,X,SX,RA,SR,BX: 3 + rd(src)

CP [A,]src

src = R,RX,IM,IR,DA, X,SX,RA,SR,BX

R,RX,IM:2

IR,DA,X,SX,RA,SR,BX: 3 + rd(src)

CPL [A]

DAA [A]

DEC dst

dst = R,RX,IR,DA,X,

sx,ra,sr,bx

R,RX: 3

IR,DA,X,SX,RA,SR,BX: 4 + rd(dst) + wr(dst)

DIV[HL,]src

src = R.RX.IM, DA, X, SX,RA,SR,BX

R,RX,IM:46

1. if divide by zero

20 if overflow

DA,X,SX,RA,SR,BX: 47 + rd(src)

1. + rd(src) if divide by zero

21 + rd(src) if overflow

DIVU [HL,]src

src = R,RX,IM,DA,X, SX,RA,SR,BX

R,RX,IM: 34 .

4 if divide by zero . ■

13 if overflow

DA,X,SX,RA,SR,BX: 35 + rd(src)

5 + rd(src) if divide by zero

14 + rd(src) if overflow

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EXTS [A] | 4 | | | |
| INC dst  « | dst= R,RX,IR,DA,X, SX,RA,SR,BX | R,RX: 3  IR,DA,X,SX,RA,SR,BX: 4 ч | F rd(dst) ч | F wr(dst) |

MULT [A,]src

src = R,RX,IM,IR,DA, X,SX,RA,SR,BX

R.RX.IM: 17

IR,DA,X,SX,RA,SR,BX: 18 + rd(src)

add 1 if src < 0

See Table E-1 Note on page E-10.

**Instruction Addressing Modes Execution Time**

**8-BIT ARITHMETIC AND LOGIC GROUP (Continued)**

MULTU [A.Jsrc src = R,RX,IM,IR,DA. R.RX.IM: 17

X.SX.RA.SR.BX IR.DA.X.SX.RA.SR.BX: 18 + rd(src)

NEG [A] **3**

OR [A.Jsrc src= R.RX.IM.IR.DA, , R,RX,IM: 2

X,SX,RA,SR,BX IR.DA.X.SX.RA.SR.BX: 3 + rd(src)

SBC [A.Jsrc src = R,RX,IM,IR,DA, R.RX.IM: 2

X,SX,RA,SR,BX IR.DA.X.SX.RA.SR.BX: 3 + rd(src)

SUB [A,]src ? ■ src = R,RX,IM,IR,DA, R.RX.IM: 2

. X,SX,RA,SR,BX IR.DA.X.SX.RA.SR.BX: 3 + rd(src)

XOR [A,]src src = R,RX.IM.IR,DA. R.RX.IM: 2

X.SX.RA.SR.BX IR.DA.X.SX.RA.SR.BX: 3 + rd(src)

**16-BIT ARITHMETIC AND LOGIC GROUP**

ADC dst,src dst = HL 3

src = BC.DE.HL.SP or

dst = IX 3

src = BC.DE.IX.SP or

dst = IY 3

src = BC,DE,IY,SP

ADD dst,src dst - HL 3

src = BC.DE.HL.SP or

dst = IX 3

src 4-BC.DE.IX.SP or

••••••• dst = IY 3 '

src = BC,DE,IY,SP

ADD dst,A dst = HL,IX,IY 3

ADDW[HL,]src src = RR,IM,DA,X,RA RR,IM: 3

DA.X.RA: 3 + rd(src)

CPW[HL,]src src = RR, IM, DA, X, RA RR,IM:3

DA.X.RA: 3 4- rd(src)

DECW dst dst = RR,IR,DA,X,RA RR: 3

IR,DA,X,RA: 4 + rd(dst) 4- wr(dst)

DEC[W] dst dst - RR **3**

DIVUW[DEHL,]src src = RR,IM,DA.X.RA RR,IM: 51

*4* if divide by zero

13 if overflow

’ DA.X.RA: 52 4- rd(src)

5 4- rd(src) if divide by zero

See fable E-1 Note on page E-10. 4-rd(src) if overflow

**Instruction Addressing Modes Execution Time**

**16-BIT ARITHMETIC AND LOGIC GROUP (Continued)**

DIVW [DEHLJsrc src = RR,IM,DA,X,RA RR,IM: 63

4 if divide by zero

20 if overflow

DA.X.RA: 64 + rd(src)

5 + rd(src) if divide by zero

21 + rd(src) if overflow

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EXTS HL | | 4 • | | |
| INCW dst | dst = RR, IR, DA.X.RA | RR: 3  IR,DA.X.RA: 4 н | f rd(dst) н | F wr(dst) |
| INC[W] dst | dst = RR | 3 |  |  |

MULTUW [HL,]src src = RR,IM,DA,X,RA RR,IM: 24\*

DA.X.RA: 25 + rd(src)\*

\*add 1 if src < 0

MULTW [HL,]src src = RR,IM,DA,X,RA RR,IM: 24

DA.X.RA: 25 + rd(src)

NEG HL 3

SBC dst.src dst = HL 3

. src = BC,DE,HL,SP or dst=IX . 3

src = BC.DE.IX.SP or dst = IY **3**

src = BC.DE.IYSP SUBW [HL,]src src = RR,IM,DA.X.RA RR,IM: 3

DA.X.RA: 3 + rd(src) See Table E-1 Note on page E-10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table Е\*1. Instruction Execution Times (Continued)** | | | | |
| **Instruction** | **Addressing Modes** | **Execution Time** | | • |
| **BIT MANIPULATION, ROTATE AND SHIFT GROUP** | | | | |
| BIT b.dst | dst = R.IR.SX | R: 2  IR.SX: 3 + | rd(dst) |  |
| RES b.dst | dst = R,IR.SX | R: 2  IR.SX: 4 + | rd(dst) + | wr(dst) |
| RLdst | dst = R, IR.SX | R:2  IR.SX: 4 + | rd(dst) + | wr(dst) |
| RLA |  | 2 |  |  |
| RLC dst | . dst = R,IR.SX | R: 2  IR.SX: 4 + | rd(dst) + | wr(dst) |
| RLCA |  | 2 |  |  |
| RLD |  | 5 + rd(IR) | 4- wr(IR) |  |
| RR dst | dst = R,IR.SX | R: 2  - IR.SX: 4 + | rd(dst) + | wr(dst) |
| RRA |  | 2 |  |  |
| RRC dst | dst = R,IR.SX | R: 2  IR.SX: 4 + | rd(dst) + | wr(dst) |
| RRCA |  | 2 |  |  |
| RRD |  | 5 + rd(IR) | + wr(IR) |  |
| SET b.dst | dst = R,IR.SX | R: 2  IR.SX: 4 + | rd(dst) 4 | ■ wr(dst) |
| SLA dst | dst = R,IR.SX | R: 2  IR.SX: 4 + | rd(dst) 4 | • wr(dst) |
| SRA dst | dst = R, IR.SX | R: 2  IR.SX: 4 + | rd(dst) 4 | ■ wr(dst) |
| SRL dst | dst = R,IR.SX | R: 2  IR.SX: 4 + | rd(dst) 4 | - wr(dst) |
| TSET dst | dst = R,IR.SX | . R: 3  IR.SX: 1 + | rd(dst) + | wr(dst) |
| See Table E-1 | Note on page E-10. | • |  | 1 |

**PROGRAM CONTROL GROUP**

CALL cc.dst

*1*

dst = IR.DA.RA

CALL dst

dst = IR.DA.RA

\*

cc not true: 3 IR.DA: 11 + wr(IR) RA: 12 + wr(IR)

IR.DA: 11 4- wr(IR)

RA: 12 4- wr(IR)

CCF

DJNZ dst

dst = RA

В is zero: **6**

В is non-zero: 7

JAF dst

dst = RA

AF'not in use: 3

AF'in use: **4**

JAR dst

dst = RA Alternate file not in use: 3 .

|  |  |  |  |
| --- | --- | --- | --- |
| JP cc.dst | dst = IR.DA.RA | cc not true: 3  cc true: 4 | **F** |
| JPdst | dst = IR.DA.RA | **4** | **F** |
| JR cc.dst | dst = RA | cc not true: 3  cc true: 4 | **F** |
| JR dst | dst = RA | **4** | **F** |
| RET |  | 9 + rd(IR) | **F** |
| RETcc | \* | cc not true: 3  cc true: 9 + rd(IR) | **F** |
| RST dst | dst DA | 9 4- wr(IR) | **F** |
| SC nn |  | 1 4- System Call Trap |  |

Alternate file in use: 4

F

SCF 2

\*“F” indicates that the pipeline is flushed when that instruction is executed.

See Table E-1 Note on page E-10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table E-1. Instruction Execution Times (Continued)** | | | | |
| **Instruction** | **Addressing Modes** | **Execution Time** | | |
| **INPUT/OUTPUT INSTRUCTION GROUP** | | | | |
| IN dst,(C) | dst = R,RX,DA,X,RA.SR.BX | R,RX: 3 + in()  DA,X,RA.SR.BX: 4 4- in() 4-wr(dst) | | |
| IN A,(n) |  | 5 4 | F in() |  |
| IN[W] HL,(C) |  | 3 4 | F in() | \* |
| IND |  | 8 4 | H in() + | wr(IR) |
| INDW |  | 8 4 | F in() 4- | wr(IR) |
| INDR |  | 8 4 | F in() + | wr(IR), each iteration |
| INDRW | \* \* <  ***’ t*** | 8 4 | F in() + | wr(IR), each iteration |
| INI |  | 8 4 | F in() + | wr(IR) |
| INIW |  | 8 4 | F in() + | wr(IR) |
| INIR |  | 8 4 | F in() + | wr(IR), each iteration |
| INIRW | •«» \* \* ' | 8 J | F in() + | wr(IR), each iteration |
| OUT (Q.src | src = R,RX,DA,X,RA.SR.BX | R,RX: 3 4- out()  DA.X,RA.SR.BX: 3 + rd(src) 4- out() | | |
| OUT (n),A |  | 5 - | F out() |  |
| OUT[W](C),HL |  | 3 - | F out() |  |
| OUTD |  | 8 - | F rd(IR) | -F out() |
| OUTDW |  | 8 - | F rd(IR) | + out() |
| OTDR |  | 8 - | F rd(IR) | + out(), each iteration |
| OTDRW |  | 8 - | F rd(IR) | 4- out(), each iteration |
| OUTI |  | 8 - | F rd(IR) | 4- out() |
| OUTIW |  | 8 + rd(IR) | | 4- out() |
| OTIR |  | 8 - | F rd(IR) | 4- out(). each iteration |
| OTIRW |  | .... 8 - | F rd(IR) | 4- out(). each iteration |
| TSTI (C) |  | 3 - | F in() |  |
| See Table E-1  « | Note on page E-10. |  | • | • |

**CPU CONTROL GROUP**

DI mask

mask = Hex value

El mask

mask = Hex value

3 + out(l)

HALT

11 + rd(halt) minimum

IMp

LD dst,src

dst = A

src = l,R

LD dst,src

dst = l,R src = A

LDCTL dst,src

dst = (C),USP

src = HL,IX,IY

(C): 4 + out(l)

USP:2

■> -I».

***I.***

or

dst = HL,IX,IY src = (C),USP

USP:2

NOP

PCACHE

RETI

Z-BUS: 8 + rd(IR)

Z80:8 + rd(reti) + rd(IR)

RETIL

RETN

F” indicates that the pipeline is flushed when that instruction is executed.

**NOTES:**

1. This table assumes that the instruction has been fetched, decoded, and is ready for execution. The execution time for instructions that cause the pipeline to be flushed do not include the time necessary to fetch and decode the following instruction.
2. This table assumes that the PAUSE input is inactive. If PAUSE is active, the execution unit will wait before beginning the next instruction.
3. The bus is assumed to be idle when the execution unit makes a request for a transaction.
4. This table assumes that no exceptions occur during instruction execution except where indicated.

Instruction Addressing Modes Execution Time

EXTENDED INSTRUCTION GROUP TEMPLATE FETCH (EPU ENABLE BIT SET TO 1)

Aligned template . 7 + epu(ifl) + epu(ifn) + out(l)

Unaligned template • 7 + epu(ifl) + 2\*epu(ifn) + out(l)

EXTENDED INSTRUCTION GROUP

EPUI (Internal Operation) *4* + p F\*

EPUF (CPU-EPU) 6 + p + epu(cpu) F

MEPU dst (Memory—EPU) dst = IR,DA,X,RA,SR,BX 5 + p + k‘[3 + epu(wr)] F

EPUM src (EPU —Memory) src = IR,DA,X,RA,SR,BX 5 + p + k\*[3 + epu(rd)] F

\*“F” indicates that the pipeline is flushed when that instruction is executed.

NOTES:

1. Additional cycles are necessary for address computation in the case of EPU-to-Memory and Memory-to-EPU instructions, as shown below.

" IR,DA ’ “ no additional cycles

X,SX,RA,SR 1 additional cycle

BX 2 additional cycles

1. The notation "p” in the table is the number of pause cycles added to the bus cycle.
2. The notation “k” in the table is a function of n, the number of bytes to be transferred that is specified in the template, and the address of the source or destination as shown below.

n is odd

к = (n+1)/2

к = n/2

к = (n = 2)/2

n is even and aligned

n is even and unaligned

1. See “Notes” from Table E-1,

Г».

**Table E'3. Interrupt, Trap, and Special Condition Execution Times**

**Type Execution Time**

**INTERRUPTS**

NMI in Modes 0,1,2 13 + iack(nmi012) + in(l) + out(l) + wr(IR)

Mode 0 9 + out(l) + [iack(mO) for each byte of opcode]

Mode 1 13 4- iack(m1) + in(l) + wr(IR) + out(l)

Mode 2 16 + iack(m2) + in(l) 4- wr(IR) + rd(IR) 4- out(l)

Mode 3 Nonvectored 28 + iack(m3) + in(l) + 3\*wr(IR) +2\*rd(IR) + out(l)

Mode 3 Vectored 31 + iack(m3) + in(l) + 3\*wr(IR) +2\*rd(IR) + out(l)

On-Chip (Mode 3) 28 + iack(m3) + in(l) + 3\*wr(IR) +2\*rd(IR) 4- out(l)

**TRAPS**

Single-Step 26 + in(l) +2\*wr(IR) + 2\*rd(IR) + out(l)

Breakpoint-on-Halt 26 4- in(l) +2\*wr(IR) 4- 2\*rd(IR) 4- out(l)

Division Exception 25 4- in(l) 4-2\*wr(IR) 4- 2\*rd(IR) 4- out(l)

Stack Overflow Warning 26 4- in(l) 4-2\*wr(IR) 4- 2\*rd(IR) 4- out(l)

Access Violation 25 4- in(l) 4-2\*wr(IR) 4- 2\*rd(IR) 4- out(l)

System Call 30 4- in(l) 4-3\*wr(IR) 4- 2\*rd(IR) 4- out(l)

Privileged Instruction 26 4- in(l) 4-2\*wr(IR) 4- 2\*rd(IR) 4- out(l)

EPU Memory 38 4- in(l) -i-4\*wr(IR) 4- 2\*rd(IR) 4- out(l)

Memory ♦- EPU 38 4- in(l) 4-4\*wr(IR) 4- 2\*rd(IR) 4- out(l)

A - EPU 31 4- in(l) 4-3\*wr(IR) 4- 2\*rd(IR) 4- out(l)

EPU Internal Operation 31 4- in(l) 4-3\*wr(IR) 4- 2\*rd(IR) 4- out(l)

**MISCELLANEOUS**

FATAL 15 4- out(l) 4- rd(halt) minimum x

RESET 3 4- rd(reset) 4- out(l) minimum

EPU Data Page Fault 1 4- epu(ifl) and then Access Violation trap

**NOTES:**

1. Additional cycles are necessary for address computation in the case of EPU-to-Memory and Memory-to-EPU traps, as shown below. . •

IR.DA no additional cycles

X.SX.RA.SR 1 additional cycle

BX 2 additional cycles

1. The pipeline is flushed at the end of any interrupt or trap sequence.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Condition | 1 X | Bus Timing | 2x | Bus Timing | 4 x | Bus Timing |
| First byte, cache | 4 |  | 4 |  | 4 | • |
| First byte, external | .9 + | w | 12 | + 2w + (0-1) | 17 | + 4w + (0-3) |
| First byte, burst | 12 | + w | 18 | + 2w + (0-1) | 29 | + 4w + (0-3) |
| Subsequent byte, cache | 1 |  | **1** |  | 1 |  |
| Subsequent byte, external | 5 + | w | 8 + | 2w + (0-1) | 13 | + 4w + (0-3) |
| Subsequent byte, burst | 8 + | w | 14 | + 2w + (0-1) | 25 | + 4w + (0-3) |

NOTES:

1. The term “first” means the first byte fetched following a flushed pipeline. All other bytes are “subsequent”. With a full pipeline, only the execution times are necessary.
2. With a 16-bit external bus, the prefetch unit tries to fetch words from external memory though bytes are transferred to the pipeline. Bytes other than the one requested are placed in cache. '
3. A burst transfer transfers a four-word block starting with the word with the three least significant bits being zero. The appropriate byte is transferred to the decoder as it is written to the cache. The execution unit of the pipeline can begin execution prior to the burst transaction completion if the necessary bytes are fetched during the early part of the burst transaction.
4. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will
5. be half of the sum of the minimum and maximum numbers in parentheses.
6. The notation "w” in the transaction tables is the number of WAIT states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.

Examples of instruction fetch/decode time (assuming flushed pipeline and 1 x bus timing):

6.

1. Two-byte instruction in cache
2. Two-byte instruction both bytes not in cache
3. Two-byte instruction, first byte in cache, second not in cache
4. Four-byte instruction in cache
5. Four-byte instruction not in cache, no burst, not cacheable
6. Four-byte instruction not in cache, burst, cacheable
7. Six-byte instruction, burst, first two bytes in cache

[4 + 1 ] processor cycles

[(9 + w) + (5 + w)]

[4 + (5 + w)]

[4 + 1 +1 + 1 ] processor cycles

[9 + w + 3 \* (5 + w)] processor cycles

[12 + w + 1 + 1 + 1]

[4 + 1 + (8 + w) + 1 +1 + 1]

Table E-5. Data Read Timing — rd(src), rd(dst), and rd(IR)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Condition | 1 X | Bus Timing | 2 x | Bus Timing | 4 x Bus Timing |
| Byte Hit | 5 |  | 5 |  | 5 |
| Byte Miss | 8 4 | ■ w | 11 | 4- 2w 4- (0-1) | 16 4- 4w 4- (0-3) |
| Aligned Word Hit | 5 |  | 5 |  | 5 |
| Aligned Word Miss | 8 4 | - w | 11 | 4 2w 4 (0-1) | 16 4- 4w 4~ (0—3) |
| Unaligned Word Hit Hit | 9 |  | 9 | • \* • | 9 |
| Unaligned Word Miss Hit | 12 | 4- w | 15 | 4- 2w 4- (0-1) | 20 4- 4w 4- (0-3) |
| Unaligned Word Hit Miss | 12 | 4- w | 15 | 4 2w 4 (0-1) | 20 4~ 4w 4" (0—3) |
| Unaligned Word Miss Miss | 15 | 4- W | 21 | 4- 2w 4- (0-2) | 31 4- 4w 4- (0-6) |
| TSET (cache) | 8 4 | - w | 11 | 4" 2w 4" (0—1) | 16 4- 4w 4- (0-3) |
| TSET (fixed memory) | 6 |  | **6** |  | 6 |
| Page Fault | 4 4 | - Access Violation trap | 4 ■ | F Access Violation trap | 4 4- Access Violation trap |

NOTES:

1. Additional cycles are necessary for address computation, as shown below.

IR,DA no additional cycles

X,SX,RA,SR 1 additional cycle

BX 2 additional cycles

1. A word is aligned if the address is even and the transfer is over a 16-bit bus. It is otherwise unaligned.
2. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
3. The notation “w” in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.

Table E-6. Data Write Timing — wr(src), wr(dst), and wr(IR)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Condition | 1 X | Bus Timing | 2x | Bus Timing | 4 x | Bus Timing |
| Byte | 5 |  | 5 |  | 5 |  |
| Aligned Word | 5 |  | 5 | • | 5 |  |
| Unaligned Word | 9 4 | - w | 12 - | t- 2w 4- (0-1) | 17 - | f- 4w 4- (0-3) |
| Page Fault | 4 4 | - Access Violation trap | 4 4- | Access Violation trap | 4 4- | Access Violation trap |

NOTES:

1. Additional cycles are necessary for address computation, as shown below.

no additional cycles

IR.DA

X,SX,RA,SR BX

1 additional cycle

2 additional cycles

1. A word is aligned if the address is even and the transfer is over a 16-bit bus. It is otherwise unaligned.
2. The pipeline is flushed whenever a byte being written is valid in the cache.
3. In the unaligned word case where the first byte is valid in cache, the execution time is 10 cycles with zero or one wait states and

9 4- w cycles for two or more wait states.

1. The number in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses. "
2. The notation “w” in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | 1 x Bus Timing | 2 x Bus Timing | 4 x Bus Timing |
| . in(l) | 5 | 5 | 5 |
| in() | 9 + w | 13 + 2w + (0-1) | 20 + 4w + (0-3) |
| wr(l) | 5 | 5 | 5 |
| wr() | 5 | 5 | 5 |

NOTES:

1. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
2. The notation “w" in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.
3. in(l) and wr(l) are performed internally within the Z280 MPU.

Table E-8. EPU Read and Write Timing

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Type | 1 X | Bus Timing | 2x | Bus Timing | | | 4 x | Bus Timing | | |
| epu(ifl) | 8 4- | w | 11 | + 2w 4 | F (0-1) |  | 16 | + 4w 4 | f (0-3) | ***” » ’'л*** |
| epu(ifn) | 8 + | w | 11 | + 2w d | f (0-1) |  | 16 | 4- 4w 4 | f (0-3) |  |
| epu(cpu) | 9 + | w | 13 | + 2w d | F (0-1) |  | 20 | 4- 4W 4 | f (0-3) |  |
| epu(wr) | 10 | 4- W | 15 | + 2w d | F (0-1) |  | 24 | 4- 4w 4 | f (0-3) |  |
| epu(rd) | 8 + | w | 11 | + 2w - | r- (0-1) | • | 16 | 4- 4W 4 | ь (0-3) |  |

NOTES:

***(***

1. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
2. The notation “w" in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.

Table E-9. Interrupt Acknowledge Timing

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Type | 1 X | Bus Timing | 2 x | Bus Timing | | 4 x | Bus Timing | |
| iack(nmi012) | 4 | " ■ | 4 |  |  | 4 |  |  |
| iack(mO) | 8 4- | w | 13 | 4- 2w 4 | f (0-1) | 22 | 4- 4w 4 | f (0-3) |
| iack(m1) | 10 | 4- W | 15 | 4- 2W 4 | F (0-1) | 24 | 4- 4W 4 | F (0-3) |
| iack(m2) | 10 | 4- W | 15 | 4- 2w 4 | h (0-1) | 24 | 4- 4w 4 | F (0-3) |
| iack(m3) | 10 | 4- W | 15 | 4- 2w - | F (0-1) | 24 | 4- 4w 4 | F (0-3) |

NOTES:

1. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
2. The notation "w” in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.
3. iack(nmi012) is for NMI in modes 0, 1, and 2.

iack(mO) is for mode 0 interrupts.

**Table E-10. Miscellaneous Transaction Timing**

**Type 1x Bus Timing 2x Bus Timing 4x Bus Timing**

HALT Transaction **655**

RESET Transaction **6 6 6**

RETI Transaction 21 + w 31 + 2w + (0-2) 49 + 4w + (0-6)

**NOTES:**

1. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will half of the sum of the minimum and maximum numbers in parentheses.
2. . The notation "w” in the transaction tables is the number of WAIT states added to the bus cycle in addition to any automatically inserted WAIT states. This includes any WAITs added under program control.

Appendix F.

Compatible Peripheral Families

The Z280 MPU supports two different types of bus interface: the Z80-Bus and the Z-8US. Families of peripheral devices are available for both types of component interconnect buses.

The Z80 Bus configurations of the Z280 MPU have two compatible peripheral families: the Z8400 and Z8000/Z8500 peripheral families (fables F-1 and F-2). The Z8400 family of devices were originally designed to support the Z80-Bus. The Z8000 series of peripherals are designed for systems employing multiplexed address/data buses, and are also easily interfaced to Z80-Bus Z8000 MPU systems. The Z8500 peripheral family is identical to the

Z8000 family, except the devices are configured to interface to non-multiplexed buses: the Z8500 series devices can be used in Z280 MPU systems where the address/data bus is de-multiplexed external to the processor.

The Z-BUS versions of the Z280 MPU are supported by the Z8000/Z8500 peripheral family (Table F-2). These devices interface directly to the Z-BUS.

Refer to the Zilog Components Data Book for further information regarding these peripheral families.

**Table F-1. Z8400 Peripheral Family**

**Part Number Description**

Z8410 DMA Direct Memory Access Controller

Z8420 PIO Parallel Input/Output Controller

Z8430 CTC Counter/Timer Circuit

Z8440/1/2 SIO Serial Input/Output Controller

Z8470 DART Dual Asynchronous

Receiver/Transmitter

**Table F-2. Z8000/Z8500 Peripheral Family**

**Part Number**

**Description**

Z8016/Z8516 DTC Direct Memory Access Transfer Controller

Z8030/Z8530 SCC Serial Communications Controller

CIO Counter/Timer and Parallel I/O Unit Z-FIO FIFO Input/Output Interface Unit Z-FIFO Buffer Unit and Z-FIO Expander BEP Burst Error Processor Z-DCP Data Ciphering Processor UPC Universal Peripheral Controller (ROM-based)

Z8036/Z8536

Z8038

Z8060

Z8065

Z8068

Z8090/Z8590

Z8094/Z8594

UPC Universal Peripheral Controller (RAM-based) '

**access protection:** A function of memory manage­ment that controls read, write, and execute access to memory locations, protecting proprietary or operating system memory areas from tampering by unauthorized users.

**access protection violation:** An incorrect or for­bidden attempt to access a memory location; for example, an attempt to write to a read-only page. An access violation causes the CPU to abort the current instruction and generate an Access Violation trap.

**accumulator: A** register within a central processing unit (CPU) that can hold the result of an arithmetic or logical operation.

**address space:** A set of addresses that are accessed in a similar manner. The Z280 MPU contains four types of address spaces: CPU register, CPU control register, memory, and 1/0. The memory space can be divided into four separate memory address spaces: system-mode program, system-mode data, user-mode program, and user-mode data.

**addressing mode:** The way in which the location of an operand is specified. There are nine addressing modes in the Z280 MPU: Register, Immediate, Register Indirect, Direct Address, Indexed, Short Index, Base Index, Relative Address, and Stack Pointer Relative. \_

**address tag:** The portion of certain associative memories that is compared against a referenced address to determine whether the matching value is found. The address tag for a cache block is the physical memory address.

**address translation:** The process of mapping log­ical addresses into physical addresses.

**aligned address:** An address that is a multiple of an operand’s size in bytes. Aligned word addresses are a multiple of two.

**associative теяогу:** A memory in which data is accessed by specifying a value rather than a location. The cache is an associative memory.

Glossary

**autodecreeent:** The operation of decrementing an address in a register by the operand’s size in bytes. The decrement amount is one for byte operands, two for word operands.

**autoincrement:** The operation of incrementing an address in a register by the operand’s size in bytes. The increment amount is one for byte operands, two for word operands.

**base address:** The address used, along with an index and/or displacement value, to calculate the effective address of an operand. The base address is located in a register, the Program Counter, or the instruction.

**Base Index (BX) addressing mode:** In this mode, the contents of the base register and index regis­ter are added to obtain the effective address.

**burst transaction:** The transfer of several con­secutive items of data in one memory transaction.

**bus master:** The device in control of the bus.

**bus reguest: A** request for control of the bus initiated by a device other than the bus master.

**byte:** A data item containing eight contiguous bits. A byte is the basic data unit for addressing memory and peripherals.

, . ■ . ' '7 . • -J ***, . 1 • •' .*** " ' » » - >\* » • . \*•« » • • \*•» - 7» •

**cache:** An on-chip buffer that automatically stores copies of recently used memory locations (both instructions and data), allowing fast access for memory fetches.

**Central Processing Unit (CPU):** The primary functioning unit of a computer, consisting of an ALU, control logic for decoding and executing instructions and controlling program flow, and registers.

**coprocessor:** A processor that works synchronously with the CPU to execute a single instruction stream using the Extended Processing Architecture (EPA).

**destination:** The register, memory location, or device to which data are to be transferred.

**Direct Address (DA) addressing node:** In this mode, the effective address is contained in the instruction.

**displacement:** A constant value located in the instruction that is used for calculating the effective address of an operand.

**effective address:** The logical memory address of an operand, calculated by adding the base address, an optional index value, and an optional displacement•

**EPU internal operation:** An EPU-handled operation that controls EPU operations but does not transfer data.

**exception:** A condition or event that alters the usual flow of instruction processing. The Z280 MPU supports three types of exception: reset, interrupts, and traps.

**Extended Processing Architecture (EPA):** A CPU facility that allows the operations defined in the architecture to be extended by hardware or software. If enabled, the CPU transfers EPA instructions to an Extended Processing Unit (EPU) for execution; if disabled, the CPU traps EPA instructions for software emulation.

**Extended Processing Unit (EPU):** An external device, that handles Extended Processing Architecture instructions (such as floating-point arithmetic).

**flowthrough transaction:** A DMA-initiated data transfer consisting of separate read and write transactions. Data is temporarily stored in the DMA channel between the read and write transactions.

**flyby transaction:** A transaction controlled by the bus master, but in which another device transfers data to the responding device.

**frame:** A block of physical memory used by the memory management mechanism to map logical memory pages.

**global bus:** A bus shared by tightly coupled, multiple CPUs; the bus master is chosen by an external arbiter device.

**hit:** A hit occurs when a associative memory is searched for a value and a match is found.

**identifier word:** A 16-bit code saved on the

system stack during exception processing that provides information about the cause of the exception.

**Immediate (1И) addressing mode:** In this mode, the operand is contained in the instruction.

**index:** A value located in a register used for calculating the effective address of an operand. The index value usually specifies the calculated offset of an operand from the orgin of an array or other data structure.

**Indexed (X) addressing mode:** In this mode, the contents of an index register are added to a base address contained in the instruction to obtain the effective address.

**Indirect Register (IR) addressing mode:** In this mode, the effective address is contained in a register.

**interrupt:** An asynchronous exception that occurs when an NMI or INT line is activated, usually when a peripheral device needs attention.

**least recently used (LRU):** The CPU records the order of use for cache blocks. When a tag miss occurs, the CPU replaces the least recently used block.

**local bus:** The bus controlled by the CPU and shared with slave processors.

**logical address:** The address manipulated by the program. The memory management mechanism translates logical addresses to physical addresses•

loosely coupled CPUs: CPUs that execute

independent instruction streams and communicate through a multi-ported peripheral, such as a Z8038 FIO I/O interface unit.

**Master Status register:** A 16-bit CPU control register that contains status information describing the current operating states of the CPU. \* •

**memory management:** Ihe process of translating logical addresses into physical addresses, plus certain protection functions. In the Z280 MPU, memory management is integrated into the chip.

**memory-mapped I/O:** An I/O device accessed in the memory address space.

**miss:** A miss occurs when an associative memory is searched for a value and no match is found.

**nonmaskable interrupt:** The highest priority interrupt; cannot be disabled.

**page:** A logical memory unit mapped by the memory management mechanism to a physical memory frame.

**paged translation: A** method of address translation in which the logical and physical address spaces are divided into fixed, equal-sized units called pages and frames, respectively. During address translation, a logical page is mapped to an arbitrary physical frame.

**physical address:** The 24-bit address required for accessing memory and peripherals, obtained by the CPU’s address translation hardware.

**relocation:** The process of mapping a logical address to a different physical address, so that multiple processes can use the same logical address for distinct physical memory locations.

**request:** A signal or message used by a device to indicate the need for some action or resource.

**reset:** A CPU operating state or exception that results when a reset request is signaled on the RESET line.

**responder:** The device to which bus transactions

transfer data.

Register (R) addressing

In this mode, the

**pipeline:** A computer design technique in which an instruction is executed in a sequence of stages by different functional units. The functional units can be operating on several different instructions simultaneously, similiar to an automobile assembly line.

**prefetching:** Ability of the CPU to fetch an instruction or operand before the previous instructions have been completed.

**privileged instruction:** An instruction that performs I/O operations, accesses control registers, or performs some other operating system function. Privileged instructions execute in system mode only.

**Program Counter (PC):** One of the two Program Status registers; it contains the address of the current instruction.

**Program Status registers:** The two registers (Program Counter and Master Status register) that contain the Program Status. The Program Status is automatically saved during exception processing.

**protection:** See access protection.

**read access:** The type of memory access used by the CPU for fetching data operands other than those specified by Immediate addressing mode.

**refresh:** To restore information that fades away if left alone. For example, dynamic memories must be refreshed periodically in order to retain their contents.

operand is in a general-purpose register.

Relative Address (RA) addressing mode: In this mode, the displacement in the instruction is added to the contents of the Program Counter to obtain the effective address.

**self-modifying program:** A program that stores to a location from which a subsequent instruction is fetched.

**semaphore:** A storaqe location used as a Boolean variable to synchronize the use of resources among multiple programming tasks. A semaphore ensures that a shared resource is allocated to only one task at any given time.

**service routine:** Program code that is executed in response to an interrupt or trap.

**Short Index addressing mode:** In this mode, the contents of the IX or IY register are added to an 8-bit displacement contained in the instruction to obtain the effective address of the operand.

**slave processor:** A processor, such as a Direct Memory Access transfer controller, that performs dedicated functions asynchronously to the CPU.

**Source:** The register, memory location, or device from which data are being read.

**spatial locality:** The characteristic of program behavior whereby consecutive memory references often apply to closely located addresses.

**stack:** An area of memory used for temporary storage and subroutine linkages. A stack uses the first-in, last-out method for storing and retrieving data; the last data written onto the stack will be the first data read from the stack.

‘ \*

**Stack Pointer (SP):** A register indicating the top (lowest address) of the processor stack used by Call and Return instructions for linking procedures. User and system modes of operation use separate Stack Pointers, the User Stack Pointer (USP) and System Stack Pointer (SSP).

**system mode:** A operating system can execute

CPU mode of operation, used for

funct ions .  
privileged

In this mode, the CPU

(and all other)

**unaliqned address:** An address that is not a multiple of an operand’s size in bytes. Odd addresses are unaligned for words.

instructions.

**user mode:** A CPU mode of operation, generally

**System Stack Pointer (SSP):** The Stack Pointer used while the CPU is in system mode. User mode programs cannot access the SSP.

**tag hit:** On a memory reference, a tag hit occurs when the cache address tags are searched for the referenced address and a match is found.

**tag miss:** On **a** memory reference, a tag miss occurs when the cache address tags are searched for the referenced address and no match is found.

**temporal locality:** The characteristic of program behavior whereby memory references often apply to -a location that has been referred to recently.

used for application programs. In this mode, the CPU cannot execute privileged instructions or access protected memory locations.

**User Stack Pointer (USP):** The Stack Pointer used while the CPU is in user mode. System mode programs can access the USP with the Load Control instruction.

**vectored interrupt:** A interrupt that uses the low-order byte of the identifier word as a vector to an interrupt service routine; can be disabled.

**virtual memory:** A memory management technigue in Г >

which the system’s logical memory address space is

not necessarily the same as, and can be much

**tightly coupled CPUs:** CPUs that execute independent instruction streams and communicate through shared memory on a common (global) bus.

**transaction:** A basic bus operation involving the transfer of one byte or word of data between the CPU and a memory or peripheral device.

**trap:** An exception that occurs when certain conditions, such as an access protection violation, are detected during execution of an instruction.

larger than, the available physical memory.

**wait state:** A clock period that is added to a memory or 1/0 transaction due to an active WAIT signal. Wait states are used to prolong memory and 1/0 transactions to devices with long access times.

**word:** A data item containing sixteen contiguous bits.

**write access:** The type of memory access used by

the CPU for storing data operands.

Index

Access violation, 1:4, 7:5

Access violation trap, 1:3, 5:3,4, 6:4,5, 7:1,2,7

Add/Subtract flag, 5:1

Address spaces, 1:2,6, 4:1,6 CPU control register space, 1:2, 2:1,2, 4:2,6 CPU register space, 1:2, 2:1,2, 4:6 I/O address space, 1:2, 2:1,4, 4:2,6 Memory address space, 1:2, 2:1,3, 4:1-6 Address translation, 2:3, 7:1-6 with program/data separation, 7:1,2,4 without program/data separation, 7:2,3

Addressing modes, 1:3, 4:1-6, 5:1,6,10, 7:2,5

Base Index (BX), 1:6, 4:1,5,6, 5:6,10, 10:7, B:1 ' ”’‘

Direct Address (DA), 1:3, 2:4, 4:1,2, 5:6-10, 10:7

Immediate (IM), 1:3, 4:1

Indexed (X), 1:3,6, 4:1,3,6, 5:6,10, 10:7

Indirect Register (IR), 1:3,6, 2:4, 4:1,2, 5:4,6-8,10, 10:7

Program Counter Relative (RA), 1:3, 4:1,4, 5:6,8,10, 7:2,5, 10:7 Register (R, RX), 1:3, 4:1

Short Index (SX), 1:3, 4:1,3,6, 5:4,6,7

Stack Pointer Relative (SR), 1:3,6, 4:1,5, 5:6,10, 10:7

-B-

Base Index (BX) addressing mode, 1:3,6 4:1,5,6, 5:6,10, 10:7 B:1

Bit manipulation, rotate and shift instructions, 1:3, 5:1,7 Block move port, 7:6 Block transfer and search instructions, 1:3, 4:6, 5:1-5 Bootstrap mode, 3:2, 9:20-22, 11:1 Вгeakpоint-on - Ha11 trap, 1:3, 3:4, 5:3,4, 6:4-6

Burst mode, 3:3,4, 8:2, 9:10,15-17, 10:3, 12:3, 13:1,3,9, E:13

Bus configuration and timing ,

Z-BUS, 1:1, 9:12,16, 12:1, 13:4 Z80 Bus, 1:1, 9:12,16, 12:1,2,4, A:1

Bus request, 1:4, 9:9,10, 10:1-5,8, 11:1, 12:2-3

Bus request protocols, 10:2,3

Bus Timing and Control register, 2:2, 3:1-3, 12:4,5,12,13, 13:4,5,9,13 Bus Timing and Initialization register, 2:2, 3:1, 9:1,9, 10:2, 11:1,2,



Cache Control register, 2:2, 3:1,3,4, 8:1-4, 12:4, 13:4,9

Cache, 1:4-6, 3:3,4, 6:9, 7:1,2, 8:1-4 9:1,15, 10:8, 12:4, 13:4,9,14, A:1, E:1,13-14

Fixed-Address mode, 8:4

Memory mode, 3:3,4, 8:1-3

Organization, 3:3,4, 8:1

Carry flag, 5:1-3,7,8

Clock oscillator, 1:1,2,5, 9:1,2

Condition codes, 5:1-3

Continuous mode, 9:10,15,17,21, 12:3, 15:3

Coprocessors, 10:1,6, 12:1, 13:14

and Extended Processing Architecture, 10:6

Count register, 9:9,12-16, 11:1

Count-Time register, 9:2-6, 11:1,3

Counter/Timer registers, 6:9,10 9:4,4

Counter/Timer Command/Status register, 9:2,3,5-9

Counter/Timer Configuration register, 9:2-5,7-9,19 Count-Time register, 9:2-6, 11:1,3 I/O addresses of, 9:7 Time Constant register, 9:2,4-7,9, 11:1

Count ег/T liners, 1:4,5, 9:1-9,17,19, 10:2, 11:1,3, 12:3, 13:2

Gates and triggers, 9:2-9 \*

Linking counter/timers, 9:5,7

Operating modes, 9:3-5 .

Segue nee of events, 9:7,8 '

Terminal count condition, 9:3-5,8,9,15,16

Count-Time register, 9:2-6

CPU control instructions, 5:1,9,10

CPU control register space, 1:2, 2:1,2

CPU Control registers, 3:1-6, 6:1

CPU register file, 2:1,2

Byte/Word registers, 2:1,2

Flag and accumulator registers, 2:1,2

Index registers, 2:2

Interrupt register, 2:2

Program Counter, 2:1,2

Refresh register, 2:1,2 Stack Pointers, 2:1,2 CPU register space, 1:2, 2:1,2

Daisy chain timing, 3:2,3, 8:3

Data types, 1:2,6, 2:4, 4:6

Descriptor Select port, 7:6

Destination Address register, 9:9,10,12-14,16,17, 11:1

Direct Address (DA) addressing mode, 1:3, 2:4, 4:1,2, 5:6-8,10, 10:7

Division Exception trap, 1:3, 3:4, 5:3, 6:4,5

DMA channels, 1:1,4,5, 3:2, 7:1, 8:2, 9:1,9-17,21, 10:2,4,6, 11:1,3, л', ч

12:2,3,13-15, 13:4,5,17,18,19

DMA linking, 9:9,12,13

DMA programming, DMAs linked to UART, 9:9,13,17,21,22

DMA programming, linked DMAs, 9:9,13,16

DMA registers, 9:12,13,15,16,21 •

DMAseqиence of events, 9:15,16 ? . .

DMA transfer mode, 9:10,11

End-of-process , 9:11-16,21, 13:2

Priority resolution, 9:12 .

Types of DMA operations, 9:10

DMA Flowthrough transaction, 9:9-11,15-17,21, 13:5,17

DMA Flyby transaction, 9:9-11,14,15 12:2,13, 13:2

DMA modes of operation, 9:10,11,14, 12:3, 13:3

burst mode, 9:10,15-17, 12:3, 13:3,9, E:13

continuous mode, 9:10,15,17,21, 12:3, 13:3 single transaction mode, 9:10,17, 12:3, 13:3

DMA registers, 9:12,13,15,16,21

Count register, 9:10,12,13,14,16, 11:1,3

Destination Address register, 9:9,10,12,13,15-17, 11:1,3

DMA Master Control register, 9:9-13,15,17 DMA Transaction Descriptor register, 9:9,11-17 Source Address register, 9:9,10,12-17, 11:1

-Е-

End-of-Process, 9:11-16,21, 12:5, 13:2

Exception conditions, 1:3, 5:3,4, 6:1 interrupts, 1:3,5,6, 2:2, 3:4,5, 5:3,9,10, 6:1-4,6-11, 7:1 '

resets, 1:3, 3:1-6, 6:1,3,11 traps, 1:3-5, 2:2, 3:4,5, 5:3,4,9,10, 6:1,4-11, 7:1

Extended instructions, 1:4, 3:5, 5:1,3,10, 6:4, 8:2,3, 10:6-9, 13:5,9,14,15 execution sequence, 10:7 Extended Instruction trap, 1:3, 3:5, 5:3,10, 6:4, 10:7, 13:14

Extended Processing Units (EPUs), 1:4, 2:3, 3:5,6, 4:6, 5:3,10, 6:4, 8:3,4, 10:6-9, 13:14,15, B:1

EPU transaction, 13:2-4,14

Fixed Address mode, 9:15 <:

Flag register, 1:2, 2:1,2, 5:2 Flowthrough mode, 9:9-11,15-17, 13:5 Flyby mode, 9:9-11,14,15, 12:2,13, 13:2,17,18 Framing error, 9:18,20

-H-

Half-Carry flag, 5:2

— I —

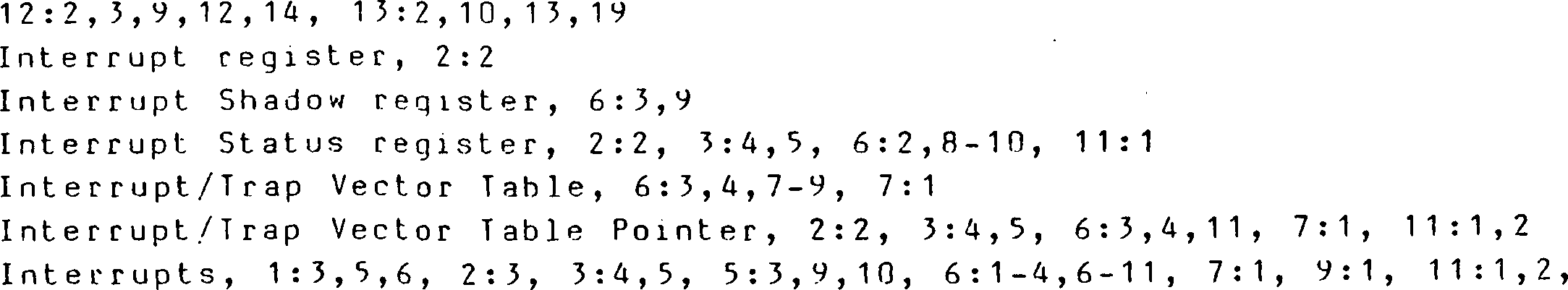
Immediate (IM) addressing mode, 1:3, 4:1, Index registers, 2:1,2 Indexed (X) addressing mode, 1:3,6, 4:1,3,6, 5:6,7,10, 10:7 Indirect Register (IR) addressing mode, 1:3,6, 2:4, 4:1,2, 5:4,6-8,10, 10:7 Input/Output instruction group, 1:3, 5:1,9 Instruction aborts, 7:7 Instruction Execution, 5:3,4 t

and exceptions, 5:3 and interrupts, 5:3,4 and traps, 5:3,4 Instruction set, 1:3,6, 5:12-172 binary encoding, 5:10,11 functional groups, 5:4 Block Transfer and Search group, 1:3, 4:6, 5:1-5 CPU Control group, 5:1,9,10 Extended Instruction group, 5:1,10, 10:6,7, 13:5,9,14,15 Input/Output group, 1:3, 5:1,9 Program Control group, 5:1,7,8 Rotate, Shift, and Bit Manipulation group, 1:3, 5:1,7 8-bit Arithmetic and Logical group, 1:3, 5:1,6 8-bit Load group, 5:1,4 16-bit Arithmetic Group, 1:3, 5:1,6,7 16-bit Load and Exchange group, 5:1,5 notation, 5:10,11

Interrupt Acknowledge, 2:2, 3:2, 6:2,3,6-8, 12:2,3,12,14, 13:2-4,13,18, A:1 Interrupt and Trap handling, 1:2,5 Interrupt Mask register, 5:10 Interrupt Modes, 3:4,5, 6:1,4,6,8,9, A:1 0: 3:5, 5:10, 6:1-3,7-9, 11:1, 12:14, 13:19, A:1 1: 5:10, 6:1-3,7-9 2: 2.2, 5:10, 6:2,3,7-9, 7.2 •

3: 3:4,5, 5:3,9,10, 6:1,3,4,7-10, 7:1, 9:1

Interrupt request, 3:4,5, 5:5, 6 : 1 - 3,6 , 7,9 8 : 3 , 9 : 1 - 5 , 7 , 1 ’ , 1 2 , 1 4 , 1 6- 18 , 20,



12:5,12,14, 15:5,5,19, E: 1,12-15 maskable, 3:4, 6:1-3,7-9, 12:3,14, 13:3,19 nonmaskable, 5.4.9, 6:2,3,7-9, 12:3,14, 13:3,19 Invalidation port, 7:6 I/O address space, 1:2, 2:1,4, 4:2,6, 9:1 Page register, 2:2,4, 3:4,5, A:1 transaction, 3:2,5, 9:1, 10:2, 12:2,4,10, 13:2,3, E:9,16 -L- Local Address register, 2:2, 3:1,3, 10:2,4, 12:10,15, 13:4,19 Loosely coupled multiple CPUs, 10:1,6 -M-

Master Status register (MSR), 2:2,3, 3:4,5, 4:5, 5:2,4, 6:1-11, 7:7, 9:4,12, 12:14, 13:19, A : 1 Memory Access Violation trap 1:3, 5:3,4, 6:4,5, 7:1,2,7 Memory Address space, 1:2, 2:1-4 System, 2:3 User, 2:3 Memory management, 1:1,3,4, 7:1 Memory transaction, 12:2,5,10,13, 13:2-11,14,17,19 MMU, 1:2,4,5, 2:3, 4:1, 5:9, 6:2,5,8,11, 7:1,2,5-7, 8:2, 9:1,14, 11:1,2, A:1 Architecture, 7:1,2 Control registers, 7:1,5,6 MMU Master Control register, 7:1,3,5,7 Page Descriptor register, 2:3, 6:5, 7:1-7, 8:2, 11:1,2 Page Descriptor Register Pointer, 7:5,6 Multiprocessor v. ...

configurations, 1:4, 3:1 mode, 1:4, 3:1,3, 10:2,4, 11:1, 12:15, 13:19 -0 -

Overrun error, 9:18,20,21

-P- < • • Page Descriptor register, 2:3, 6:5, 7:1-7, 8:2 Page Descriptor Register Pointer, 7:5,6 Page Fault trap, 3:4, 5:4 Parity error, 9:18,20,21 Par it y./Ove г Flow Flag, 5:2,5, 9:21 Peripheral Families, 1:1, F:1 Pindescriptions, Z-BUS, 15:1-5 Z80 BUS, 12:1-5 Privileged instructions, 5:4-6, 5:5,4,10, 6:4,5, A:1 Privileged Instruction trap, 1:5, 5:4,5, 5:4, 6:4,5 Processor Flags, 5:1,7,9, 6:5

Add/Subtract Flag, 5:1

Carry flag, 5:1,7,8

Half-Carry flag, 5:2

Parity-Overflow flag, 5:2

Sign flag,5:2 .

Zero flag, 5:2

Program Control instructions, 5:1,7,8

Program Counter, 2:1,2, 3:4,5, 5:7,8,10, 6:2-4,7-11, 7:7, 10:7

Program Counter Relative (RA) addressing mode, 1:3, 4:1,4, 5:6-8,10, 7:2,

, 10:7

-R-

Reason code, 6:3,8,9

Refresh, 1:2,4,5, 10:4, 12:2-4,9,10, 13:2,4,10, A:1

Refresh controller, 9:1,2

Refresh Rate register, 1:4, 9:1,2

Refresh register, 2:1,2, A:1

Register (R,4 RX) addressing mode, 1:3, 4:1, B:1

Reset, 1:3, 3:1,3-6, 5:10, 6:1,3,11, 7:5, 11:1, 12:1,3,4,9, 13:3,4,10, A:1

RETI transaction, 5:9,10, 6:3,9, 8:2-4, 12:2,9,14, E:10

Rotate, Shift, and Bit Manipulation instructions, 1:3, 5:1,7

-S-

Short Index (SX) addressing mode, 1:3, 4:1,3,6, 5:4,6,7, ; } \*

Sign flag, 5:2,3

Single-Step trap, 1:3, 3:4, 5:3,4, 6:4-6,8

Single transaction mode, 9:10,17, 12:3, 13:3

Slave processors, 10:1,2, 12:1

Source Address register, 9:9,10,12-17, 11:1

Stack Limit register, 6:5

Stack Pointer registers, 1:2, 2:1,2, 3:4, 5:3,4, 6:5, A:1

System, 2:2, A:1

User, 2:2, A:1 .

Stack Pointer Relative (SR) addressing mode, 1:3,6, 4:1,5,6, 5:6,10, 10:7

System Call trap, 1:3, 5:4, 6:4,5

System Configuration registers, 3:1

Bus Timing and Control register, 3:1,3

Bus Timing and Initialization register, 3:1

Cache Control register, 3:1,3,4

Local Address register, 3:1,3

System mode, 1:2,3,5,6, 2:2,3, 3:1,4-6 5:4,9, 6:2,3,5,7,8, 7:1,2,5, A:1

System Stack Limit register, 2:2, 3:4-6

System Stack Overflow Warning trap, 1:3, 3:6, 5:4,5, 6:4,5

System Stack Pointer (SSP), 2:1,2, 3:6, 4:5, 6:2, A:1

System Status registers, 3:1,4

Interrupt Status register, 3:4,5

Interrupt/Trap Vector Table Pointer, 3:4,5

1/0 Page register, 3:4,5 t

Master Status register (MSR), 3:4

System Stack Limit register, 2.2, 3:4-6

Trap Control register, 3:4-6 \* •

— T —

Terminal count condition, 9:3-5,8,9,15,16

Tightly coupled multiple processors, 10:1,2,4,5

Time Constant register, 9:2,4-7,9, 11:1

Trap Control register, 2:2, 3:4-6, 5:9,10, 6:4,5, 10:7, 15:14, A:1

Traps, 1:3-5, 2:2, 3:4,5, 5:1,3,4,7-10, 6:1,4-11, 7:1, 10:6, 11:1,2, 12:5 13:2,5, E : 1,12 .

Access Violation, 1:6, 5:3,4, 6:4-6, 7:1,2,7

Breakpoint-on-Halt, 1:6, 3:4, 5:3,4, 6:4-6

Division Exception, 1:6, 3:4, 5:3, 6:4,5,

Extended Instruction, 1:6, 5:5, 5:3,10, 6:4, 10:6-9, 13:14 Page Fault, 3:4, 5:3,4 \*

Privileged Instruction, 1:6, 3:4-6, 5:3,4, 6:4,5 Single-Step, 1:6, 3:4, 5:3,4, 6:4-6,8 System Call, 1:6, 5:3,4, 6:4,5 System Stack Overflow Warning, 1:6, 3:5,6, 5:3-5, 6:4,5, A:1

-U-

UART, 1:1,4,5, 3:1,2, 9:1,17-22, 11:1,3, 12:3, 13:3 bootstrapping option, 3:2, 9:20-22 operation, 9:21 registers, 9:17,18,20 I/O addresses of, 9:20 Receive Data register, 9:17,18,20,21 Receiver Control/Status register, 9:17,18,20,21 Transmit Data register, 9:17-21 Transmitter Control/Status register, 9:17-21 UART Configuration register, 9:18,19,21 receiver operation, 9:18,20, 12:3, 13:3 - .

transmitter operation, 9:17-20, 12:3, 13:3 User mode, 1:2,5,6, 2:2,3, 3:4,5, 5:4, 6:3-5, 7:1,2,5 User Stack Pointer (USP), 2:1,2, 4:5, 5:9, A:1

-Z-

Z-BUS, 1:1, 9:2,12,16, 10:6, 12:1, 13:1-19, F:1 bus configuration and timing, 9:12,16, 12:1, 13:4 bus operation, 13:2 external interface, 12:1 pin descriptions, 13:1-3 request s, 13:2,18 global, 13:18,19 . interrupt, 13:2,18,19 local, 13:18,19 transactions, 13:2-5,9-16 DMA flyby, 13:2,17 13 : 2-4, 14

Extended Processing Unit (EPU), 10:6, Halt, 13:2,4,10

‘ 1/0, 13:2,3,11  ' ■■■

Interrupt Acknowledge, 13:2-4,18 Memory, 13:2-11,14,18,19 Refresh, 13:2,4,10

Z80 Bus, 1:1, 9:2,12,16, 10:6, 12:15, 13:1 bus configuration and timing, 9:12,16,

, F:1 12:4, A:1 busoperation,12:2 .

external interface, 12:1 pin descriptions, 12:1-3 .

requests, 12:2,14 global, 12:14,15 interrupt, 12:2,14 local, 12:14,15 transactions, 12:2,4,5,9,10,12,15 DMA flyby, 12:2,13 Halt, 12:2,9,10 1/0, 12:2,10 Interrupt Acknowledge, 12:2,12,14 Memory, 12:2,5,10,13 Refresh, 12:2,9,10 RETI, 12:2,9, 14 Zeroflag,5:2,3-

**ZILOG DOMESTIC SALES OFFICES AND TECHNICAL CENTERS**

**INTERNATIONAL SALES OFFICES**

**CANADA**

**CALIFORNIA**

Agoura

Campbell

Tustin

**COLORADO**

Boulder

**FLORIDA**

Largo

**GEORGIA**

Norcross

**ILLINOIS**

Schaumburg

**NEW HAMPSHIRE**

Nashua

**MINNESOTA**

Edina

**NEW JERSEY**

Clark

818-707-2160 I oron to 416-673-0634

408-370-8120

714-838-7800 **GERMANY** Munich 49-89-672-045

303-494-2905 **JAPAN** Tokyo 81-3-587-0528

813-585-2533 **HONGKONG** Kowloon 852-3-723-8979

404-448-9370 **KOREA "** Seoul 82-2-552-5401

312-517-8080 **SINGAPORE** Singapore 65-235-7155

603-888-8590 **•«, TAIWAN .... . ... .... .**

Taipei 886-2-741-3125

612 831-7611 **UNITED KINGDOM** Maidenhead 44-628-392-00

**OHIO**

201-382-5700

Seven Hills 216-447-1480

**PENNSYLVANIA**

Ambler 215-653-0230

**TEXAS**

Dallas 214-987-9987

**WASHINGTON**

Seattle 206-523-3591

© 1989 by Zilog, Inc. All rights reserved. No part of this publica­tion may be reproduced, stored in a retrieval system, or transmit­ted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Zilog.

The information contained herein is subject to change without notice. Zilog will not be responsible for any such changes. Zilog will not be responsible for notifying any user of changes. Zilog assumes no responsibility for the use of any circuitry or other technology embodied in a Zilog product. No patent licenses, industrial property rights, or other rights are implied.

Zilog will not be responsible for any damage to the user that may result from accidents or any other reasons during operations of the products described herein.

All specifications (parameters) are subject to change without notice. Zilog will not be responsible for any such changes. Zilog will not be responsible for notifying any user of changes. The ap­plicable Zilog test documentation will specify which parameters are tested.

Zilog, Inc. 210 Hacienda Ave., Campbell, CA 95008-6609 Telephone (408) 370-8000 TWX 910-338-7621

1. **Chapter 6. Interrupts and Traps**

   6.1 Introduction

   [6.2 Interrupts](#bookmark363)

   [6.2.1 Interrupt Mode 0](#bookmark363)

   [6.2.2 Interrupt Mode 1](#bookmark363)

   [6.2.3 Interrupt Mode 2](#bookmark363)

   6.2.4 interrupt Mode 3 . . .

   [6.3 Traps](#bookmark377)

   ***t ’***

   [6.3.1 Extended Instruction Trap](#bookmark377)

   [6.3.2 Privileged Instruction Trap](#bookmark377)

   [■ 6.3.3 System Call Trap](#bookmark377)

   . [6.3.4 Access Violation Trap](#bookmark377)

   [6.3.5 System Stack Overflow Warning Trap .....](#bookmark377)

   [6.3.6 Division Exception Trap](#bookmark377)

   [6.3.7 Single-Step Trap](#bookmark377)

   6.3.8 Breakpoint-on-Halt Trap [↑](#footnote-ref-1)
2. ♦

   6.4 Interrupt and Trap Handling

   6.4.1 Interrupt Acknowledge . [↑](#footnote-ref-2)
3. [6.4.2 Status Saving](#bookmark378)

   [6.4.3 Loading New Program Status](#bookmark378)  [↑](#footnote-ref-3)
4. [6.4.4 Executing the Service Routine ..](#bookmark381) [↑](#footnote-ref-4)
5. [6.4.5 Returning from a Service Routine](#bookmark381)

   [6.5 Interrupt/Trap Vector Table .](#bookmark381)

   [6.6 The Fatal Condition](#bookmark389)  [↑](#footnote-ref-5)
6. [↑](#footnote-ref-6)