Microprocessors

Dr Binu P Chacko
Associate Professor
Department of Computer Science
Prajyoti Niketan College, Pudukad

Architecture

- Internal logic design of the microprocessor
- µp manipulates data, controls timing of various operations, and communicates with memory and I/O
- It is a programmable logic device, designed with registers, flip flops, and timing elements

Functions of µp

- Microprocessor initiated operations
- Internal data operations
- Peripheral initiated operations

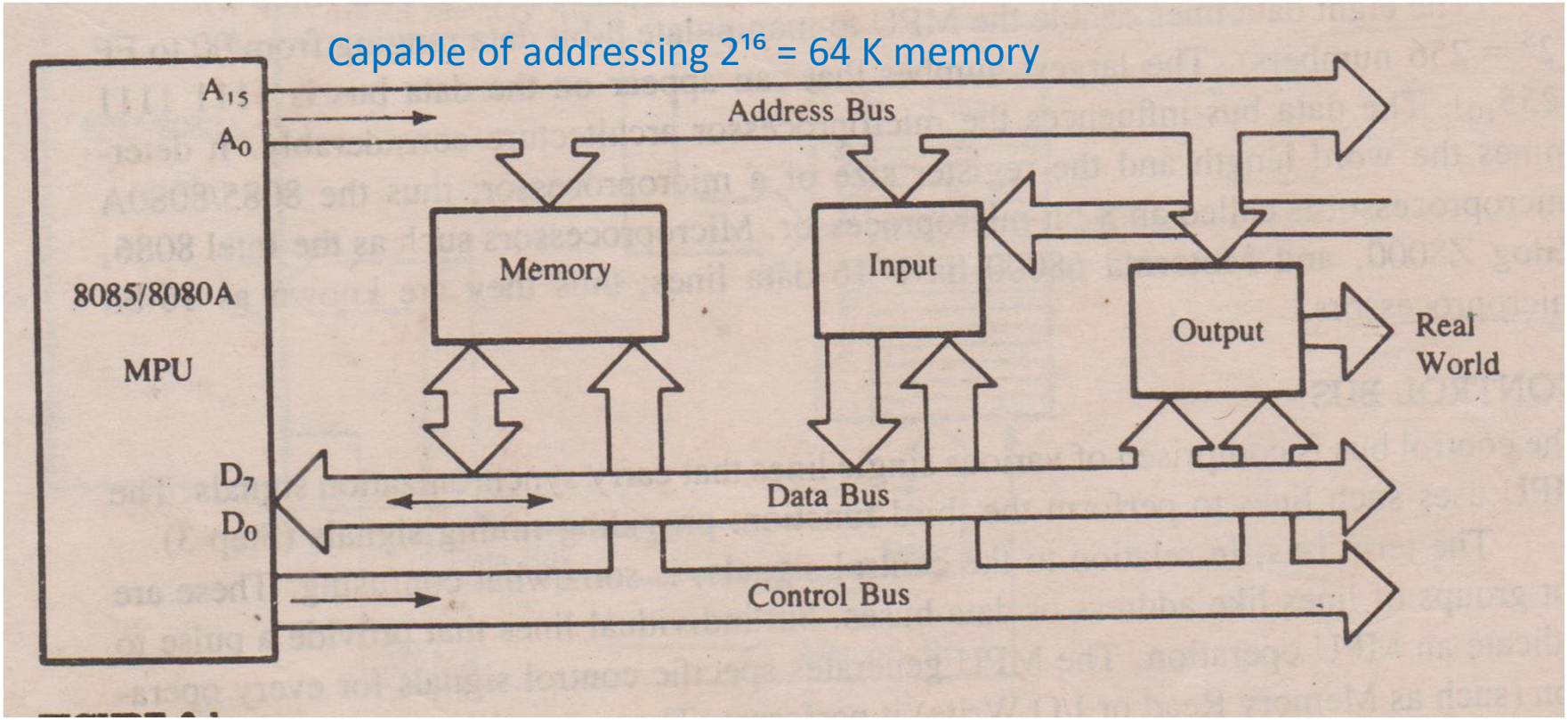
Microprocessor initiated operations

- Memory read, memory write, I/O read, I/O write

To do the above operations MPU …

- Identify the peripheral or memory location – address bus
- Transfer the data – data bus
- Provide timing or synchronization signals – control bus

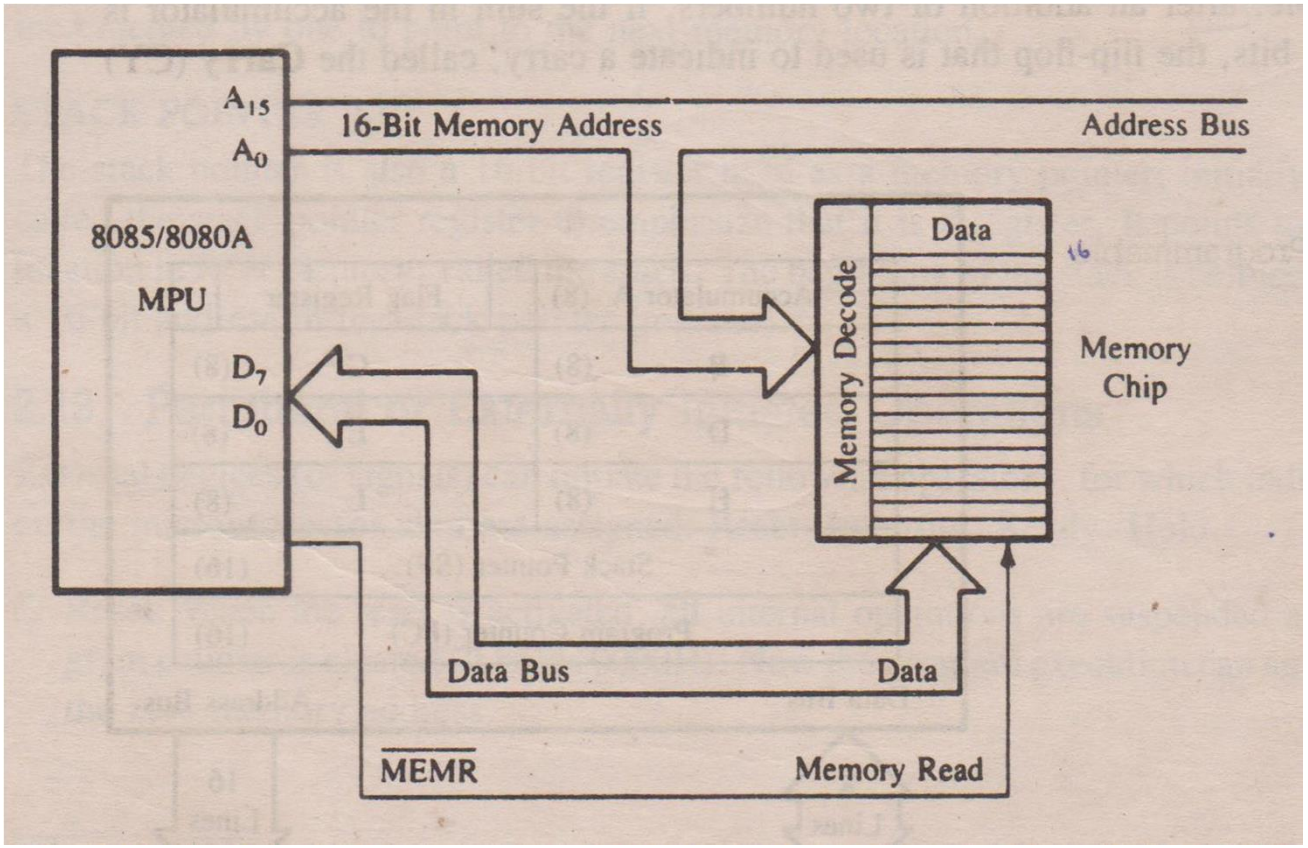
Cont…



Data bus handles 8-bit (word length) data – 00 to FF

Control bus consists of various single lines that carry synchronization signals

Internal Data Operation



Store 8-bit data, Perform arithmetic and logic operations, Test for conditions,
Store data temporarily during execution in the R/W memory locations called stack,
Sequence the execution of instructions

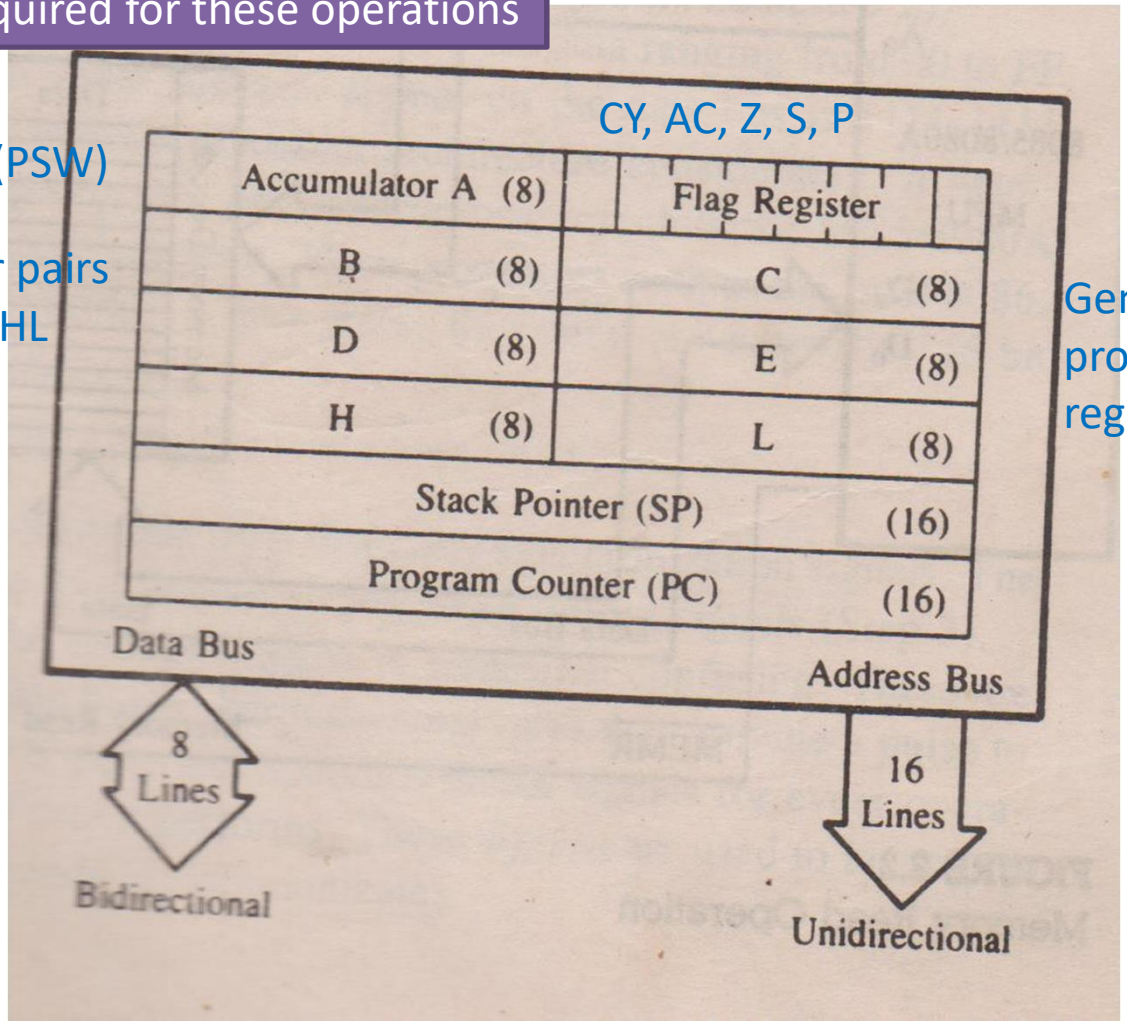
Cont…

Registers required for these operations

Part of ALU (PSW)

CY, AC, Z, S, P

Register pairs
BC, DE, HL

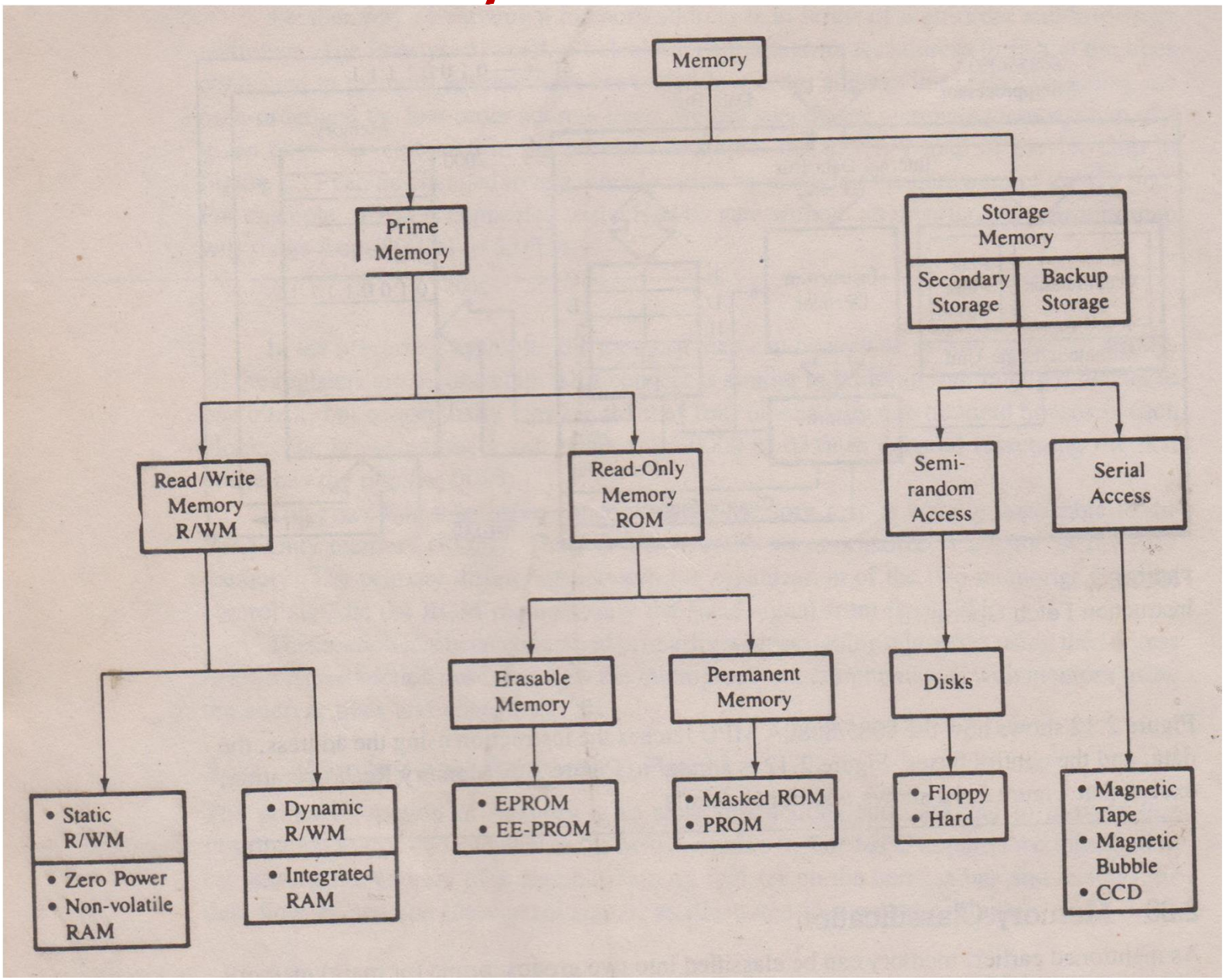


General-purpose programmable registers

Peripheral Initiated Operations

- Reset: all internal operations are suspended and PC is cleared (0000H)
- Interrupt: µp can be interrupted from normal execution of instructions, and asked to execute ISR. It resumes the previous operation after completing this
- Ready: when the signal at READY pin is low, µp enters into a Wait state – to synchronize slower peripherals with µp
- Hold: when HOLD pin is activated by an external signal, µp relinquishes the control of buses and allows external peripheral to use them. E.g., DMA data transfer

Memory Classification



Cont…

Storage memory: Magnetic disk (semi-random access) and tape (serial access)

- Nonvolatile, unlimited size, slow access, low cost
- Secondary storage, backup storage

Main memory: Used for executing and storing the program

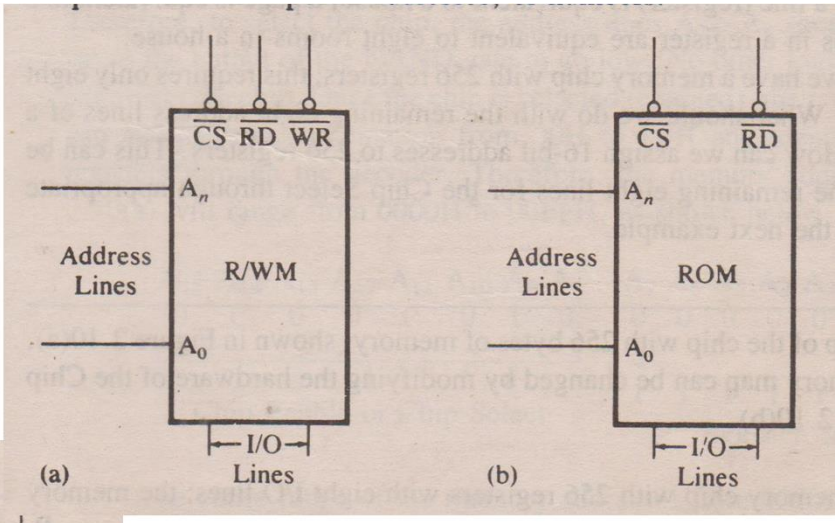
- Fast, random access memory
- R/W memory: volatile
- Static memory: made up of flip flop, store the bit as a voltage, appropriate for small systems
- Dynamic memory: made up of MOS transistor gates, store the bit as a charge, high density, low power consumption, cheap
- Charge leaks; so memory refreshing required with extra circuitry

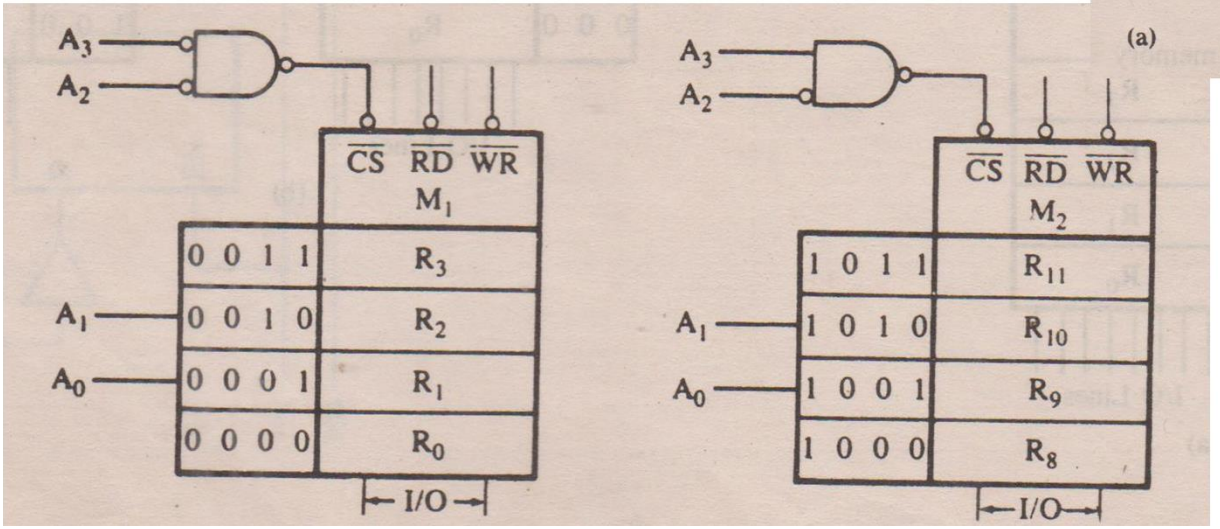
ROM

- Nonvolatile, read only
- Masked ROM: manufacturer permanently records the bit pattern by masking and metalization process
- PROM is programmed by the user with a special PROM programmer that selectively burns the fuses according to the bit pattern to be stored permanently
- EPROM: storing the information using EPROM programmer, which applies high voltage to charge the gate
- Information can be erased (by ultraviolet light) and reprogrammed
- EEPROM: information can be altered at register level using electrical signals. Chip erase mode with less time is also available

Cont…

- To communicate with the memory, µp selects the chip, identify the register, and read from or write into the register





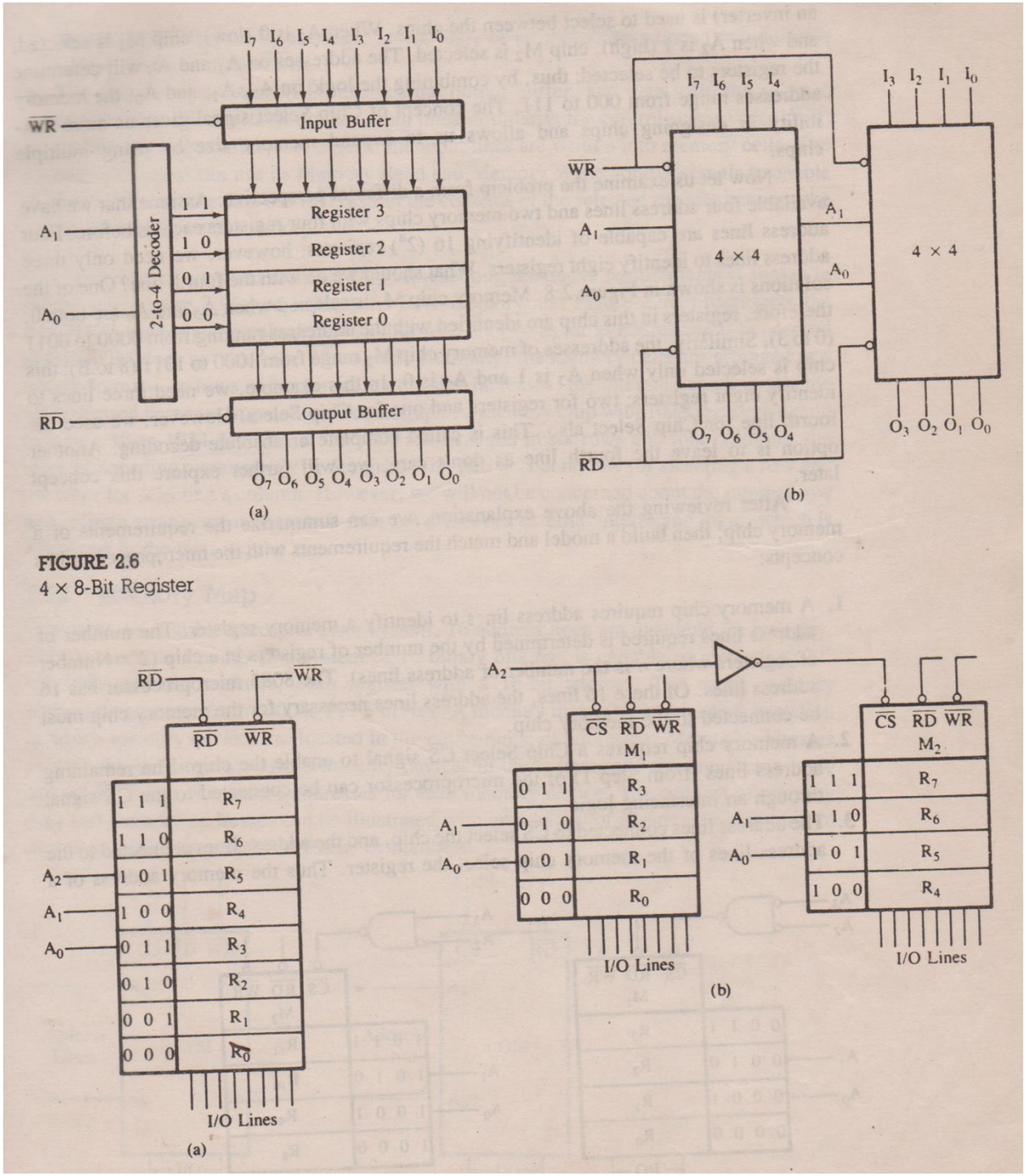
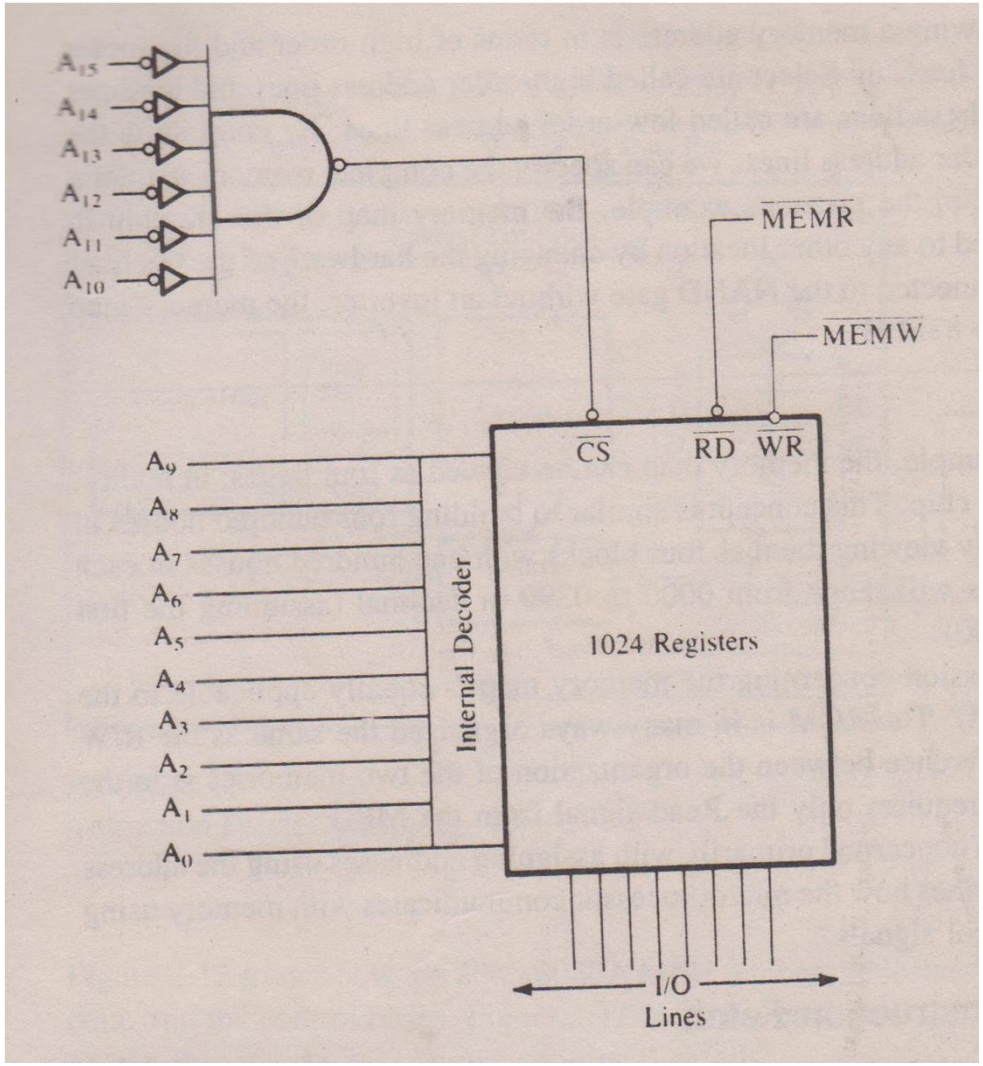


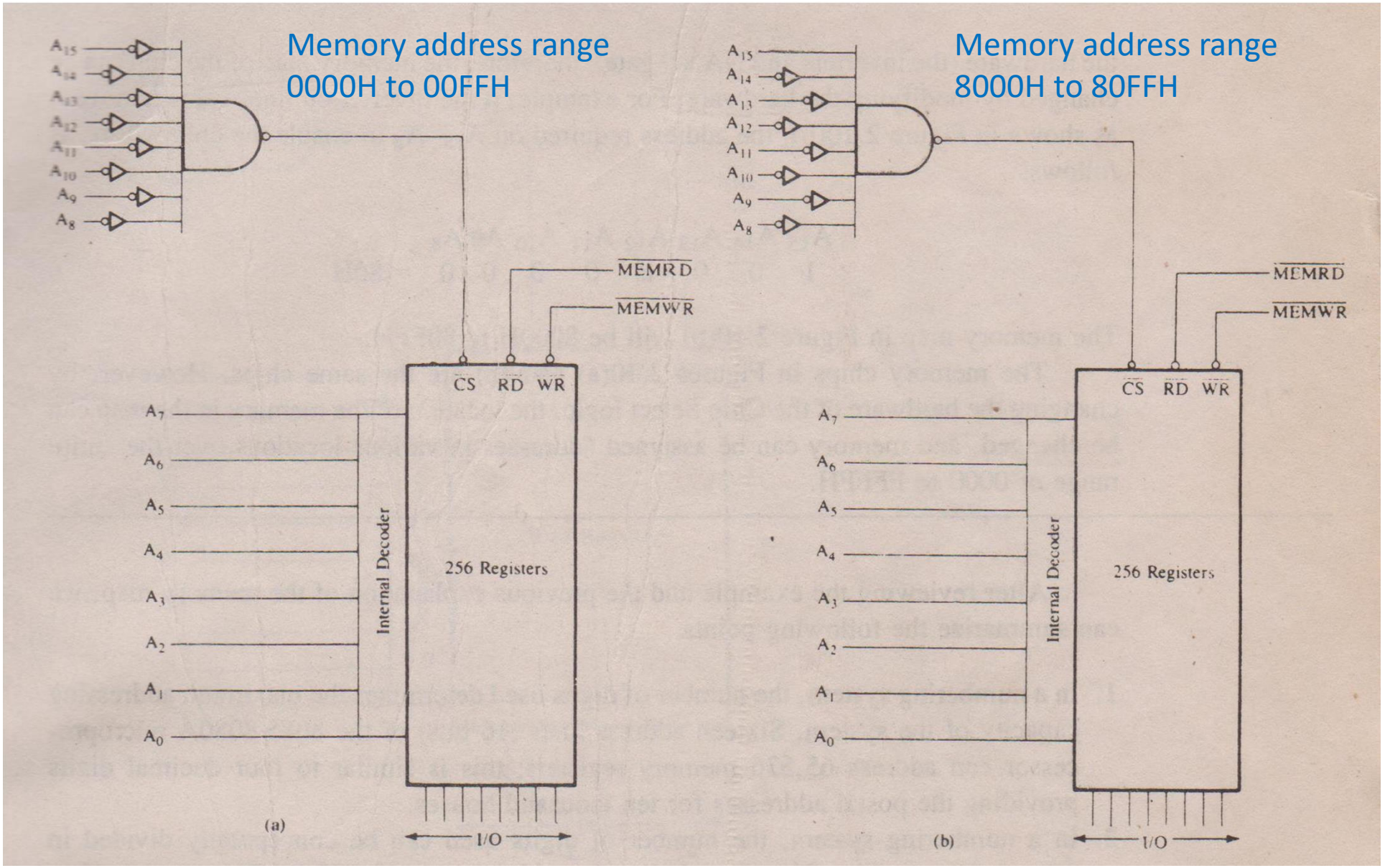
FIGURE 2.6
4 × 8-Bit Register

Memory Map

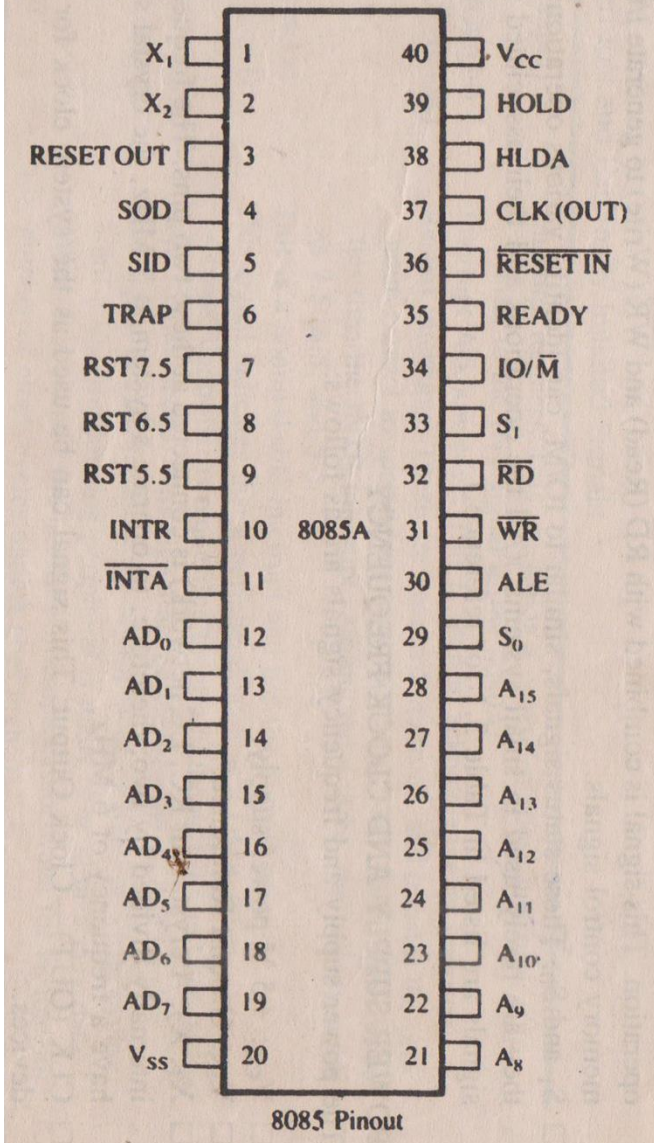
- A pictorial representation in which memory devices are located in the entire range of addresses 0000H to FFFFH
- Low-order byte and high-order byte, 16 address lines
- Memory map of 1K memory; memory address range: 0000H to 03FFH



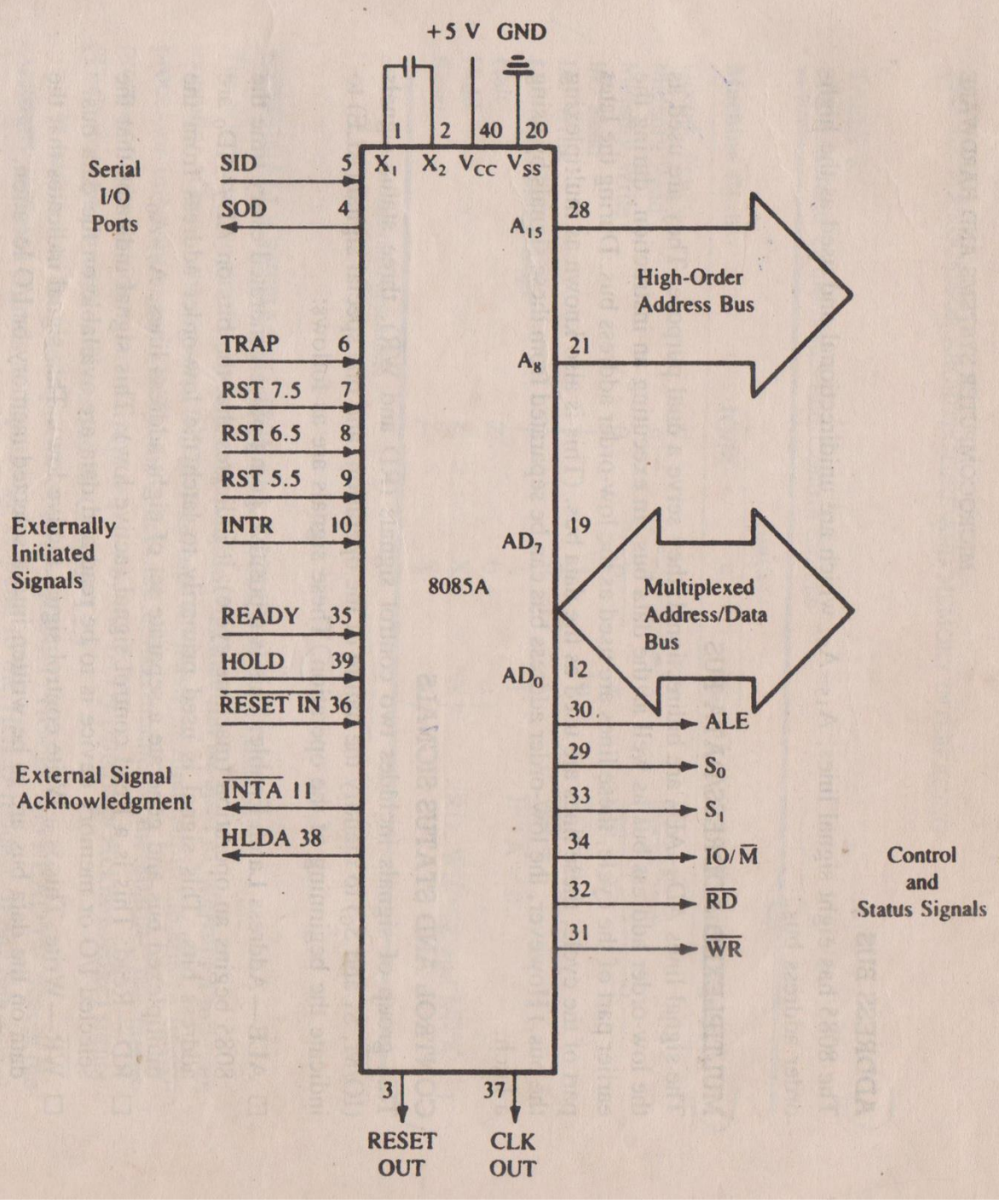
Memory map of the chip with 256 bytes of memory



8085 pins & signals



8085 Pinout



8085 MPU

- 8085A is an 8-bit general purpose µp, 40 pins, 64K addressing capability, 3 MHz speed, requires +5V single power supply
- 8080A → 8085A → 8085A-2 (5 MHZ)
- Higher-order unidirectional Address bus $A_{15} – A_8$
- Multiplexed address/data bus $AD_7 – AD_0$
- Control and status signals: ALE – positive going pulse indicates that the bits on $AD_7 – AD_0$ are address bits
- Control signals $\overline{RD}$, $\overline{WR}$
- Status signals S_0, S_1, IO/$\overline{M}$
- Power supply and clock frequency: Vcc (+5V), Vss (GND), CLK (OUT) - system clock for other devices, X_0, X_1 (a crystal is connected here, which requires 6 MHz frequency and it is internally divided by two)

Cont…

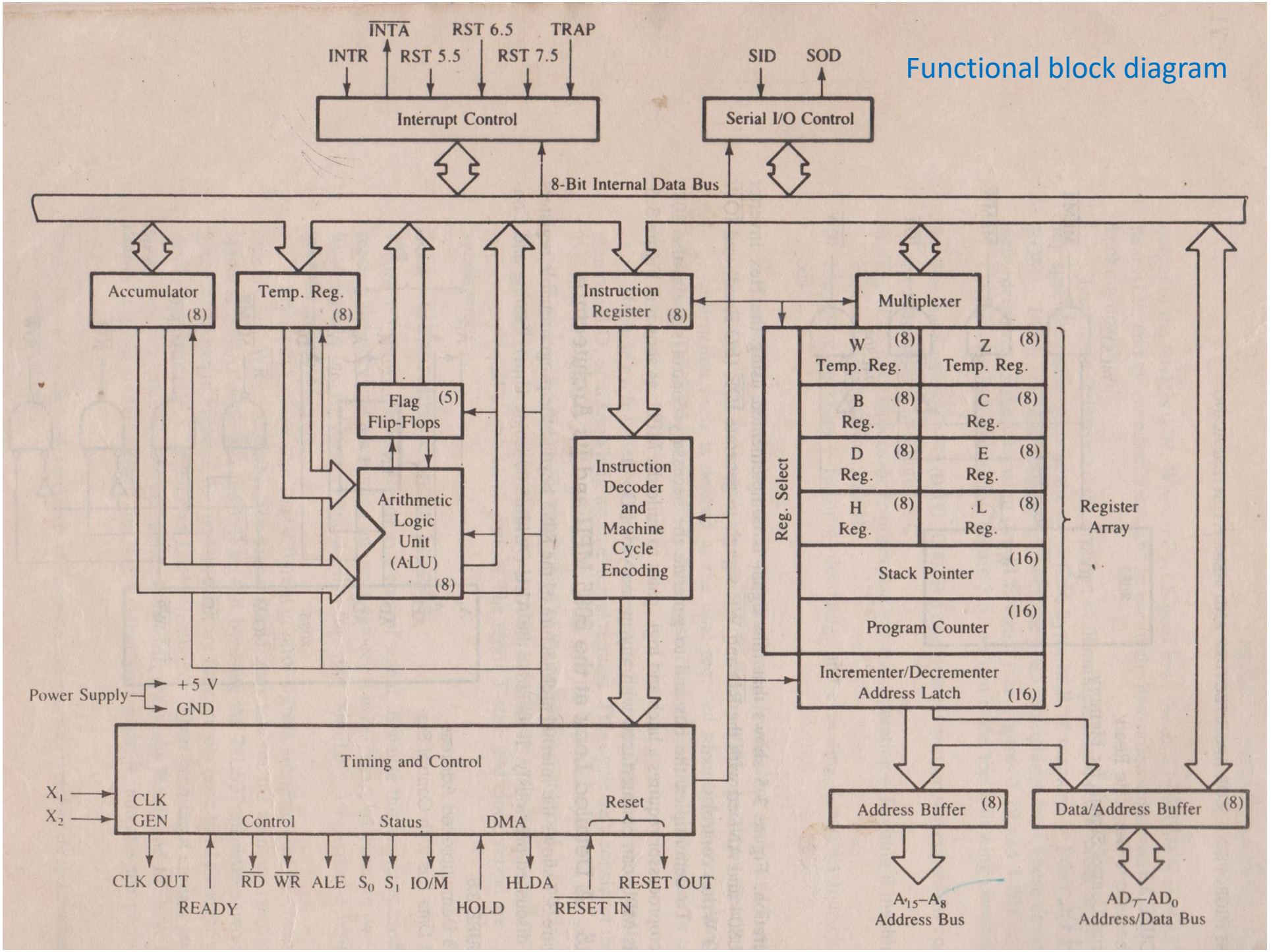
- Interrupts and externally initiated signals: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP, $\overline{\text{INTA}}$
- HOLD (input) – DMA controller is requesting the use of address and data bus. HLDA (output) acknowledges HOLD request
- READY (input) – to delay Read or Write cycles until a slow-responding peripheral is ready to send or accept data
- RESET IN – PC is set to zero, buses are tri-stated, and MPU is reset
- RESET OUT – to reset other devices
- Serial I/O ports SID, SOD

8085 Machine Cycle Status and Control Signals

Machine Cycle	Status			Control Signals
	IO/$\overline{\text{M}}$	S_1	S_0	
Opcode Fetch	0	1	1	$\overline{\text{RD}} = 0$
Memory Read	0	1	0	$\overline{\text{RD}} = 0$
Memory Write	0	0	1	$\overline{\text{WR}} = 0$
I/O Read	1	1	0	$\overline{\text{RD}} = 0$
I/O Write	1	0	1	$\overline{\text{WR}} = 0$
Interrupt Acknowledge	1	1	1	$\overline{\text{INTA}} = 0$
Halt	Z	0	0	$\overline{\text{RD}}$, $\overline{\text{WR}}$ = Z and $\overline{\text{INTA}} = 1$
Hold	Z	X	X	
Reset	Z	X	X	

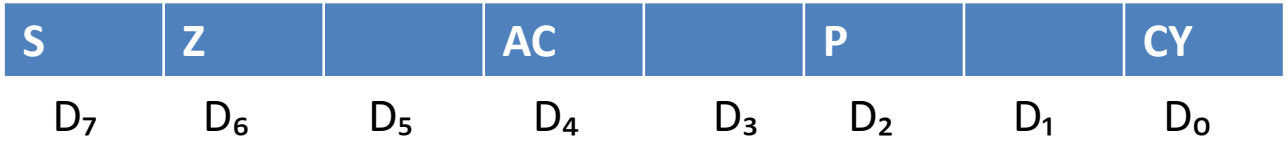
NOTE: Z = Tri-state (high impedance)
X = Unspecified

Functional block diagram



Cont…

- ALU includes accumulator, temporary register (to hold data during arithmetic/logic operation), five flags, arithmetic and logic circuits
- Sign flag is set if bit D_7 is 1 (negative number)
- Zero flag is set if ALU operation results in 0
- Auxiliary Cary flag set when carry is generated by digit D_3 (for BCD operation)
- Parity flag is set if the result has even number of 1s
- CarrY flag is set if an arithmetic operation results in a carry. It also serves as borrow flag for subtraction



Cont…

- Timing and control unit synchronizes all the µp operations with the clock and generates the control signals necessary for communication between µp and peripherals
- When an instruction is fetched from memory, it is loaded into the Instruction Register. The Decoder decodes the instruction and establishes the sequence of events to follow
- Register array

8085/8080A Assembly Language Programming

8085 (74 instructions) = 8080A+2 instruction

- Instruction set: The entire group of instructions in 8085. It determines the functions performed by the µp
- Types (functional) of instruction: Data transfer operations, Arithmetic operations, Logical operations, Branching operations, Machine-control operations
- Data transfer (copy) operations: Copies data from a location called source to another location called destination, while retaining the contents in the source
- Types of data transfer: Between registers, Specific data byte to a register or a memory location, Between a memory location and a register, Between an I/O device and the accumulator, Load 16-bit number in a register pair

Cont…

- Arithmetic operations: Addition – Any 8-bit number, or the contents of a register, or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator
- Subtraction – Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the result is stored in accumulator. Subtraction is performed using 2's complement method. If the result is negative, it will be expressed in 2's complement form
- Increment/Decrement: Contents of a register/register pair or a memory location can be incremented or decremented by 1
- Branch operations alter the sequence of program execution either conditionally or unconditionally
- Jump instructions test for a condition and alter the program sequence when the condition is met. Unconditional jump instruction is also available
- Call, Return, and Restart instructions change the sequence of a program either by calling a subroutine or returning from a subroutine. Conditional Call and Return instructions based on condition flags are also available

Cont…

- Logical operations: The following instructions perform operations with the contents of accumulator. Flags are affected due to arithmetic and logical operations
- Complement – Contents of the accumulator are complemented
- Exclusive-OR, AND, OR – Any 8-bit number, or the contents of a register, or of a memory location can be logically Exclusive-ORed, ANDed, or ORed with the contents of the accumulator. The results are stored in the accumulator
- Rotate – Each bit in the accumulator can be shifted either left or right to next position
- Compare – Any 8-bit number, or the contents of a register, or a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator
- Machine control operations: Halt, Interrupt, do nothing

Data Transfer Operations

- Copy data from register, memory or IO to another register, memory or IO
- MOV Rd, Rs — Register addressing
- MVI R, 8-bit — Immediate addressing
- IN 8-bit port address — Direct addressing
- OUT 8-bit port address
- MVI A, 82H
- MOV B, A
- IN 00H
- OUT 01H
- HLT
- NOP

Cont…

```
4100  06  MVI B, 37H          ;Load register B with 37H
4101  37
4102  78  MOV A, B            ;Copy (B) into (A)
4103  D3  OUT PORT1           ;Display accumulator
4104  PORT1                   ;contents (37H) at Port1
4105  76  HLT                 ;End of program
```

- Mnemonic = opcode + operand

```
4100  3E  MVI A, 70H
4101  70
4102  D3  OUT 01H
4103  01
4104  76  HLT
```

?

- Specify the register contents and flag status

```
MVI A, 00H
MVI B, F8H
MOV C, A
MOV D, B
HLT
```

- Write the output

```
MVI B, 82H
MOV A, B
MOV C, A
MVI D, 37H
OUT PORT1
HLT
```

Arithmetic Operations

- Addition, subtraction, increment, decrement. flags are affected
- ADD R
- ADI 8-bit
- SUB R
- SUI 8-bit
- INR R
- DCR R
- ADD C
- ADI 35H
- INR A
- ADI 01H

1 byte

2 byte

one of the operand is in A for addition and subtraction. Result is in A

1 byte

increment/decrement the contents of any register by 1. CY flag is not affected

CY sets if sum exceeds FFH

Existing value A=FFH

A = 00H, S = 0, Z = 1, CY = X

A = 00H, S = 0, Z = 1, CY = 1

Cont…

- MVI D, 8BH — D = 8BH
- MVI C, 6FH — C = 6FH
- INR C — C = 70H, S = 0, Z = 0, CY = X
- MOV A, C — A = 70H, C = 70H
- ADD D — A = FBH, D = 8BH, S = 1, Z = 0, CY = 0
- OUT PORT1 — FBH
- HLT

Instruction	Result
MVI A, 97H	A = 97H
MVI B, 65H	B = 65H
SUB B	A = 32H, B = 65H
	S = 0, Z = 0, CY = 0
HLT	

- Subtraction using 2's complement method
- MVI B, 30H — B = 30H
- MVI C, 39H — C = 39H
- MOV A, B — A = 30H, B = 30H
- SUB C — A = F7H, C = 39H, S = 1, Z = 0, CY = 1
- OUT PORT1 — F7H
- HLT
- If CY (borrow) is set, then answer is in 2's complement form

- Specify the register contents and flag status

?

```
MVI A, F2H
MVI B, 7AH
ADD B
OUT PORT0
HLT
```

- Specify the register contents and flag status

```
MVI A, F8H
SUI 69H
```

- ADD A? SUB A?
- Specify the register contents and flag status

```
MVI A, 5EH
ADI A2H
MOV C, A
HLT
```

- 3AH + 48H using ADI
- Load 00H in A, decrement it and write the output
- Write a program to clear A, add 47H, subtract 92H, add 64H and display the result

Logic Operations

- AND, OR, Exclusive-OR, Complement
- CY is reset, other flags are affected
- ANA R — 1 byte
- ANI 8-bit — 2 byte
- ORA R
- ORI 8-bit
- XRA R
- XRI 8-bit
- CMA — No flags are affected
- MVI A, 9FH
- ANI 80H — Masking except D_7
- OUT 01H
- HLT

All logic operations are performed with the contents of A

1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	9FH
1	0	0	0	0	0	0	0	80H
1	0	0	0	0	0	0	0	80H

S = 1, Z = 0, CY = 0

Cont…

- Keep radio on (D_4) without affecting other appliances

IN 00H

ORI 10H

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	1	0	0	0	0
D_7	D_6	D_5	1	D_3	D_2	D_1	D_0

- Turn off AC (D_7) without affecting other appliances

IN 00H

ANI 7FH

- MVI B, 91H
- MVI C, A8H
- MOV A, B
- ORA C
- OUT PORT1
- HLT

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	1	1	1	1	1	1	1	
0	D_6	D_5	1	D_3	D_2	D_1	D_0	
1	**0**	**0**	**1**	**0**	**0**	**0**	**1**	91H
1	0	1	0	1	0	0	0	A8H
1	0	1	1	1	0	0	1	B9H

S = 1, Z = 0, CY = 0

?

- XRA A? Z? CY?
- Specify the register contents and flag status

```
MVI  A, A9H
MVI  B, 57H
ADD B
ORA A
```

- Specify the register contents and flag status

```
XRA  A
MVI B, 4AH
ANA B
HLT
```

- Load A8H in register C, mask high-order 4 bits, and display low-order 4 bits at an output port
- Load 8EH and F7H is registers D and E respectively. Mask high-order 4 bits in both, Ex-OR low-order 4 bits and display the answer

Branch Operations

- Jump, Call and return, and Restart instructions

JMP 16-bit memory address 3 byte unconditional jump

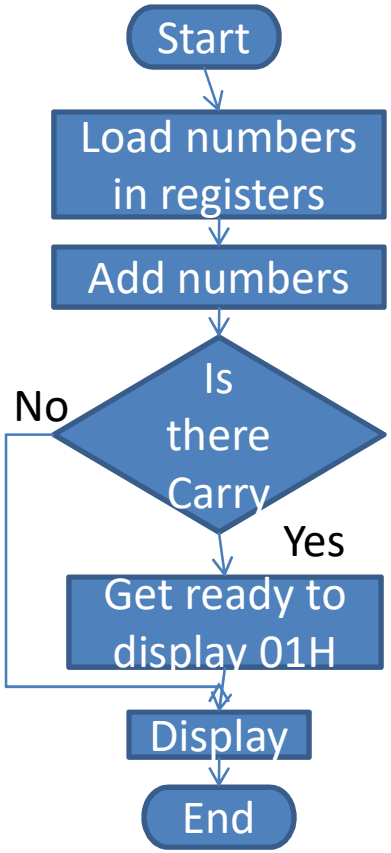
- To change the program sequence conditionally or unconditionally
- Read switch positions continuously and turn on appliances accordingly

```
2000  DB  START:  IN    00H
2001  00
2002  D3            OUT 01H
2003  01
2004  C3            JMP START
2005  00
2006  20
```

Conditional Jumps

- Check the flag conditions and make decisions to change the program sequence
- JC 16-bit, JNC 16-bit, JZ 16-bit, JNZ 16-bit, JP 16-bit, JM 16-bit, JPE 16-bit, JPO 16-bit 3 byte

```
    MVI   D, 9BH
    MVI   E, A7H
    MOV A, D
    ADD  E
    JNC   lbl
    MVI   A, 01H
lbl: OUT  PORT0
    HLT
```



?

- Write the output

```
                MVI A, 8FH
                ADI  72H
                JC     DISPLAY
                OUT PORT1
                HLT
DISPLAY:        XRA  A
                OUT PORT1
                HLT
```

- Specify the output if ADI 72H is replaced by SUI 67H
- Write instructions to clear CY, load FFH in register B and increment it. If CY is set display 01 at output port, otherwise display the contents of register B

Memory Transfer/16-bit data

- LXI Rp, 16-bit — Register pairs: BC, DE, HL, SP
- MOV R, M — Memory location in HL
- LDAX B/D — Load accumulator indirectly from memory. Memory location in BC or DE
- LDA 16-bit — Copy 8-bit data from specified memory location to accumulator
- MOV M, R — Indirect addressing
- STAX B/D — Store from accumulator indirect
- STA 16-bit
- MVI M, 8-bit
- INX Rp
- DCX Rp

Cont…

- LXI H, 2050H
- MVI H, 20H
- MVI L, 50H
- LXI H, 2040H
- MOV A, M
- LXI B, 2040H
- LDAX B
- LDA 2040H

Cont…

- LXI H, 4100H
- MOV M, B
- LXI D, 4100H
- MOV A, B
- STAX D
- STA 4100H
- LXI H, 4100H
- MVI M, F2H
- LXI B, 4100H
- INX B

Transfer a Block of Data

```
        LXI   H, 2050H
        LXI   D, 2070H
        MVI   B, 10H
NEXT:   MOV A, M
        STAX D
        INX   H
        INX   D
        DCR   B
        JNZ   NEXT
        HLT
```

?

- Specify memory location and its contents

```
MVI   B, F7H
MOV A, B
STA   2075H
HLT
```

- Show the register contents

```
MVI   C, FFH
LXI    H, 2070H
LXI    D, 2070H
MOV M, C
LDAX D
HLT
```

- Write a program to transfer a block of data in 4155H - 415AH to 4180H – 4185H in reverse order
- Write a program to transfer low-order bytes in the following data to 2050H onwards

  0581, 0595, 0578, 057A, 0598

?

- Identify the cleared memory locations

```
MVI   B, 00H
LXI   H, 2075H
MOV M, B
INX   H
MOV M, B
HLT
```

- Specify the contents of the register

```
        LXI   H, 2090H
        SUB   A
        MVI   D, 0FH
LOOP:   MOV   M, A
        INX   H
        DCR   D
        JNZ   LOOP
        HLT
```

Memory Related Arithmetic Operations

- ADD M
- SUB M
- INR M
- DCR M
- LXI H, 2040H
- ADD M
- INX H
- SUB M
- LXI H, 2040H
- MVI M, 59H
- INR M
- INX H
- MVI M, 90H
- DCR M

Array Addition

```
        XRA A
        MOV B, A
        MVI C, 06H
        LXI H, 4150H
NXTBYT: ADD M
        JNC NXTMEM
        INR B
NXTMEM: INX H
        DCR C
        JNZ NXTBYT
        STA 4170H
        MOV A, B
        STA 4171H
        HLT
```

- Write a program to add an array of numbers and store 01H at 2050H if carry is generated, otherwise store the result at that location

Rotate

- RLC, RRC
- RAL, RAR (through carry)
- Applications: multiply, divide
- Add positive numbers

```
        MVI   B, 00H
        MVI   C, 0AH
        LXI   H, 2060H
NEXT:   MOV A, M
        RAL
        JC    REJECT
        RAR
        ADD B
        JC    OVRLOD
```

```
        MOV B, A
REJECT: INX   H
        DCR   C
        JNZ   NEXT
        MOV A, B
        STA   2070H
        HLT
OVRLOD: MVI   A, FFH
        STA   2070H
        HLT
```

?

- Specify the contents of A and CY

MVI A, B7H

ORA A

RLC

```
MVI A, B7H
ORA A
RAL
```

```
MVI A, 80H
ORA A
RAR
```

```
MVI A, C5H
ORA A
RAL
RRC
```

```
MVI A, A7H
ORA A
RAR
RAL
```

- Eight data bytes are stored at 2050H onwards. If D_0 or D_7 of each data is 1, reject it; otherwise copy it to 2060H onwards.

Compare

- CMP R/M
- CPI 8-bit
- Subtract data byte from (A) and modify the flags. Contents are not affected
- If (A) > (R/M/8-bit data), then CY = 0, Z = 0
- If (A) = (R/M/8-bit data), then CY = 0, Z = 1
- If (A) < (R/M/8-bit data), then CY = 1, Z = 0

```
LXI   H, 2050H
MVI   A, 64H
CMP M
JZ    OUT1
```

Check/Add Data Byte

```
        LXI   H, 2050H
        MVI   C, 00H
        MOV B, C
NXTBYT: MOV A, M
        CPI   00H
        JZ    DSPLAY
        ADD  C
        JNC   SAVE
        INR   B
SAVE:   MOV C, A
        INX   H
        JMP   NXTBYT
DSPLAY: MOV A, C
        STA   2060H
        MOV A, B
        STA   2061H
        HLT
```

?

- Identify the contents of A and flag status

MVI A, 7FH

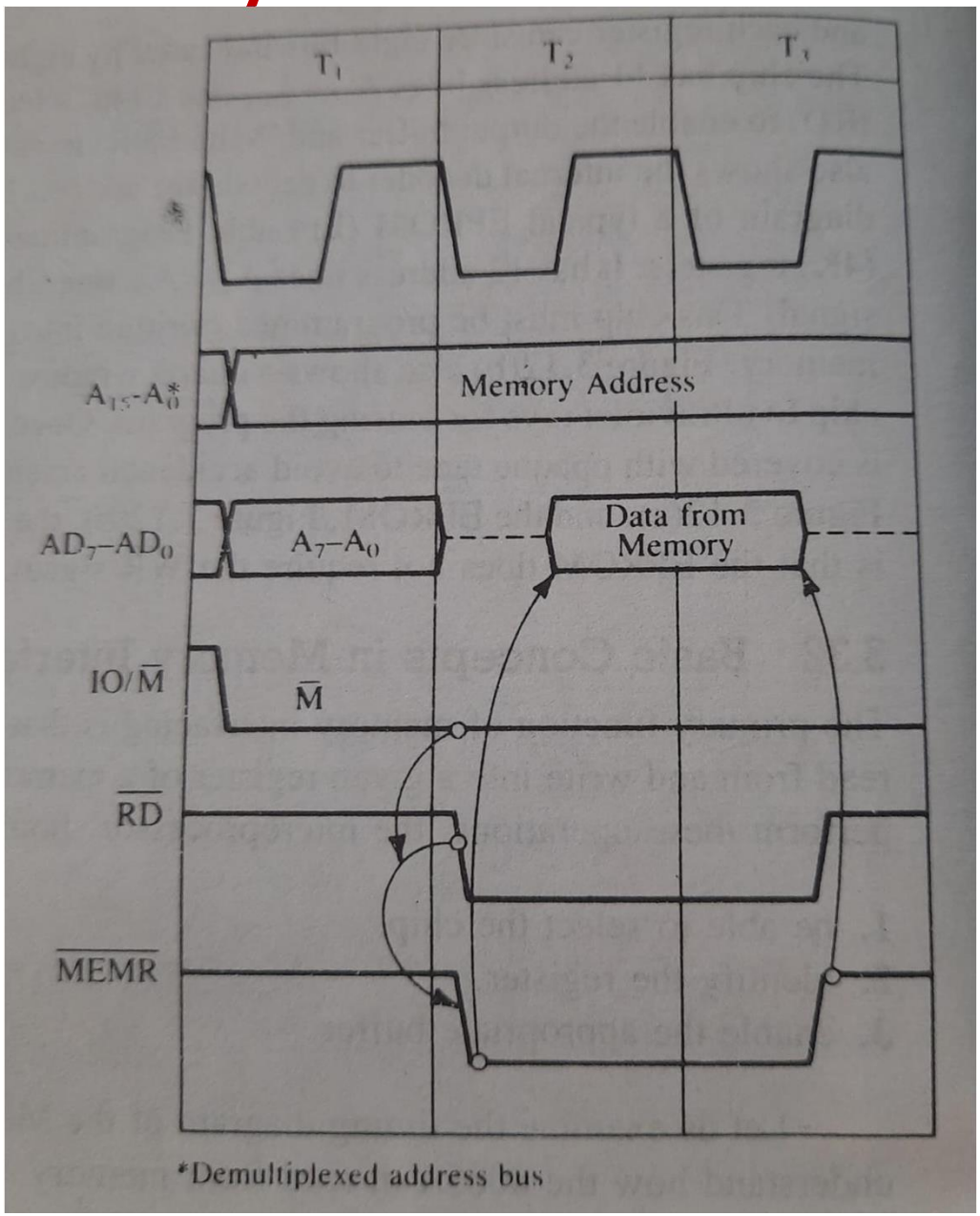
ORA A

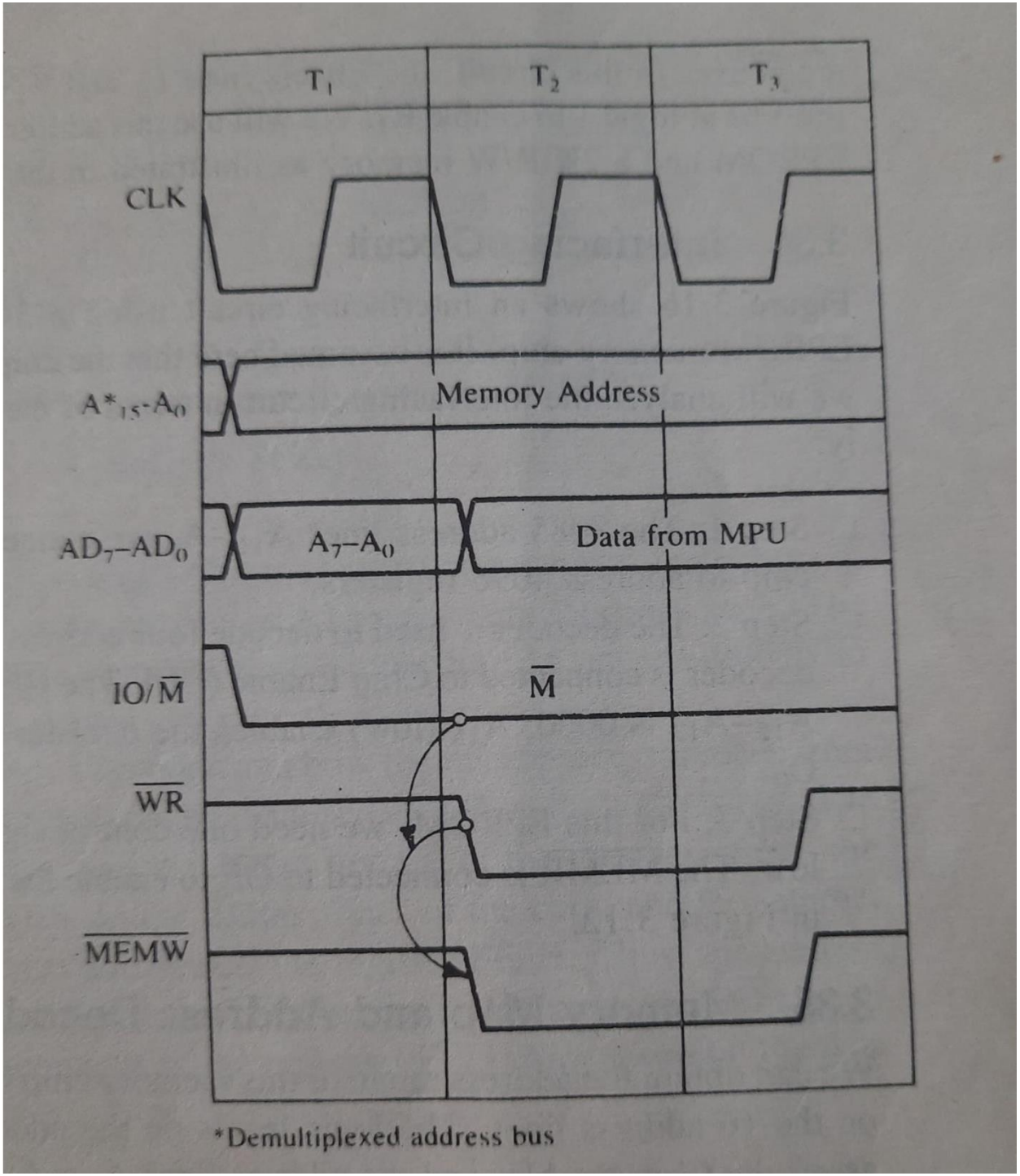
CPI A2H

```
LXI   H, 2070H
MVI M, 64H
MVI  A, 8FH
CMP M
```

- Find out the location in which data byte 40H is stored in a group of eight data bytes placed at 2050H onwards
- Find out the lowest data in a group
- Find out the data bytes between 32H and 64H in the given group

Machine Cycles and Bus Timings





*Demultiplexed address bus

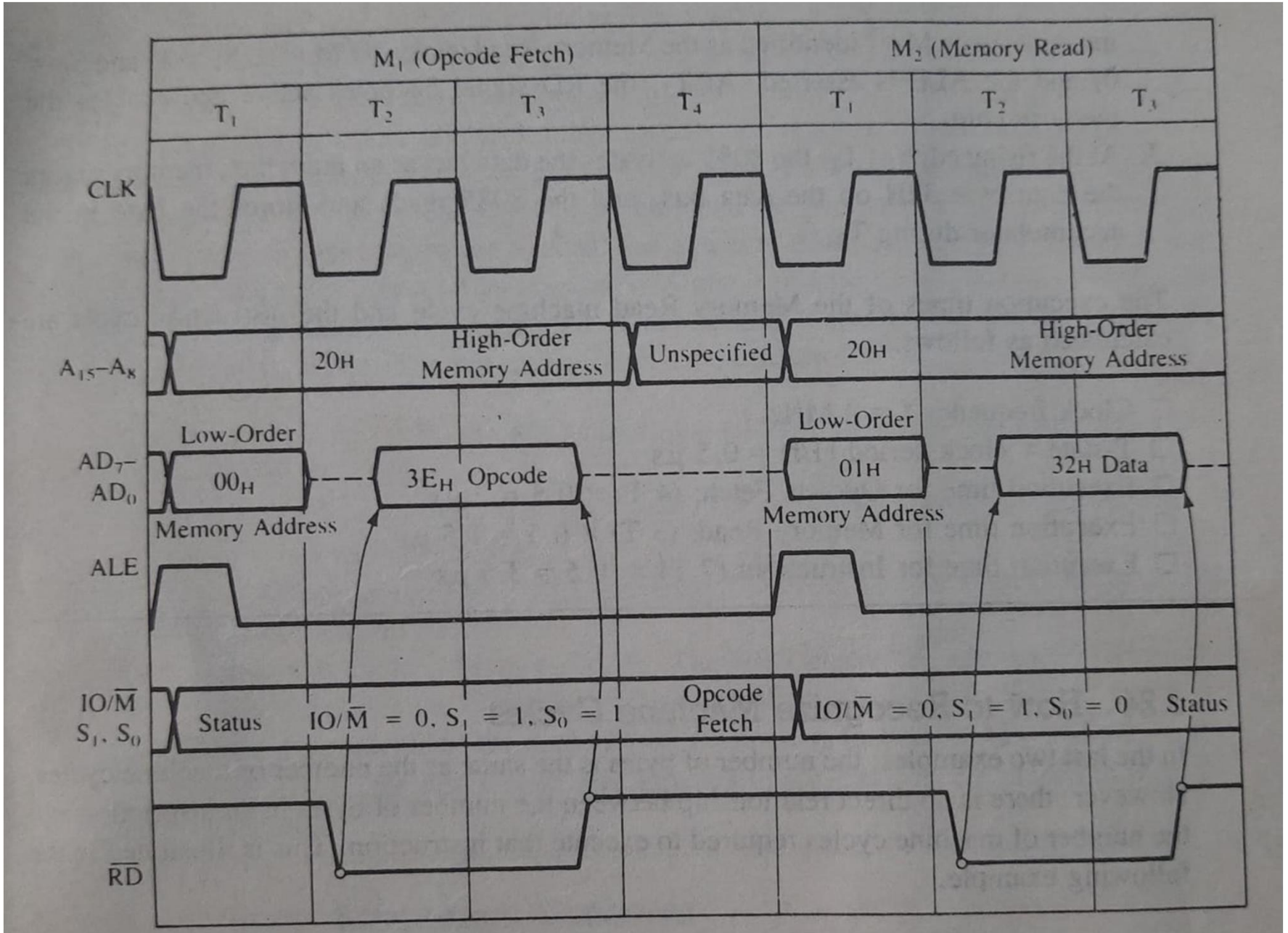


FIGURE 3.11
8085 Timing for Execution of the Instruction MVI A,32H

Cont…

Opcode	Operand	Bytes	Machine Cycles	T-states
STA	2065H	3	4	13

Memory Address	Machine code	Hex code	Remarks
2010	0011 0010	32H	Opcode
2011	0110 0101	65H	Low-order address
2012	0010 0000	20H	High-order address

- M_1: places the address 2010H on address bus and fetches the opcode 32H
- M_2: places the address 2011H on the address bus and read low-order byte 65H from memory
- M_3: read high-order byte 20H from memory location 2012H
- M_4: places the address 2065H on the address bus, contents of accumulator of data bus, and write contents of data bus into memory location 2065H

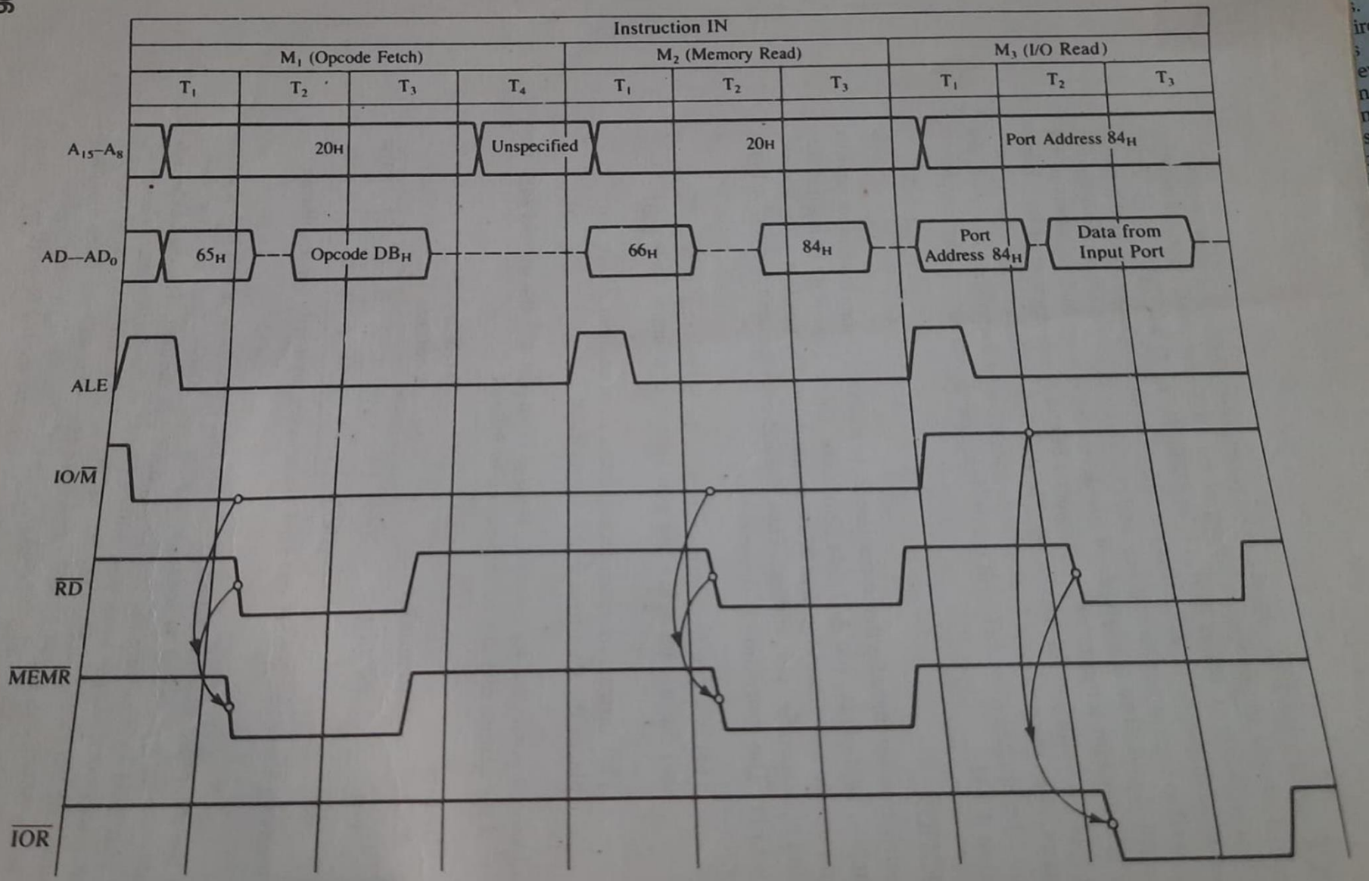


FIGURE 4.2
8085 Timing for Execution of IN Instruction

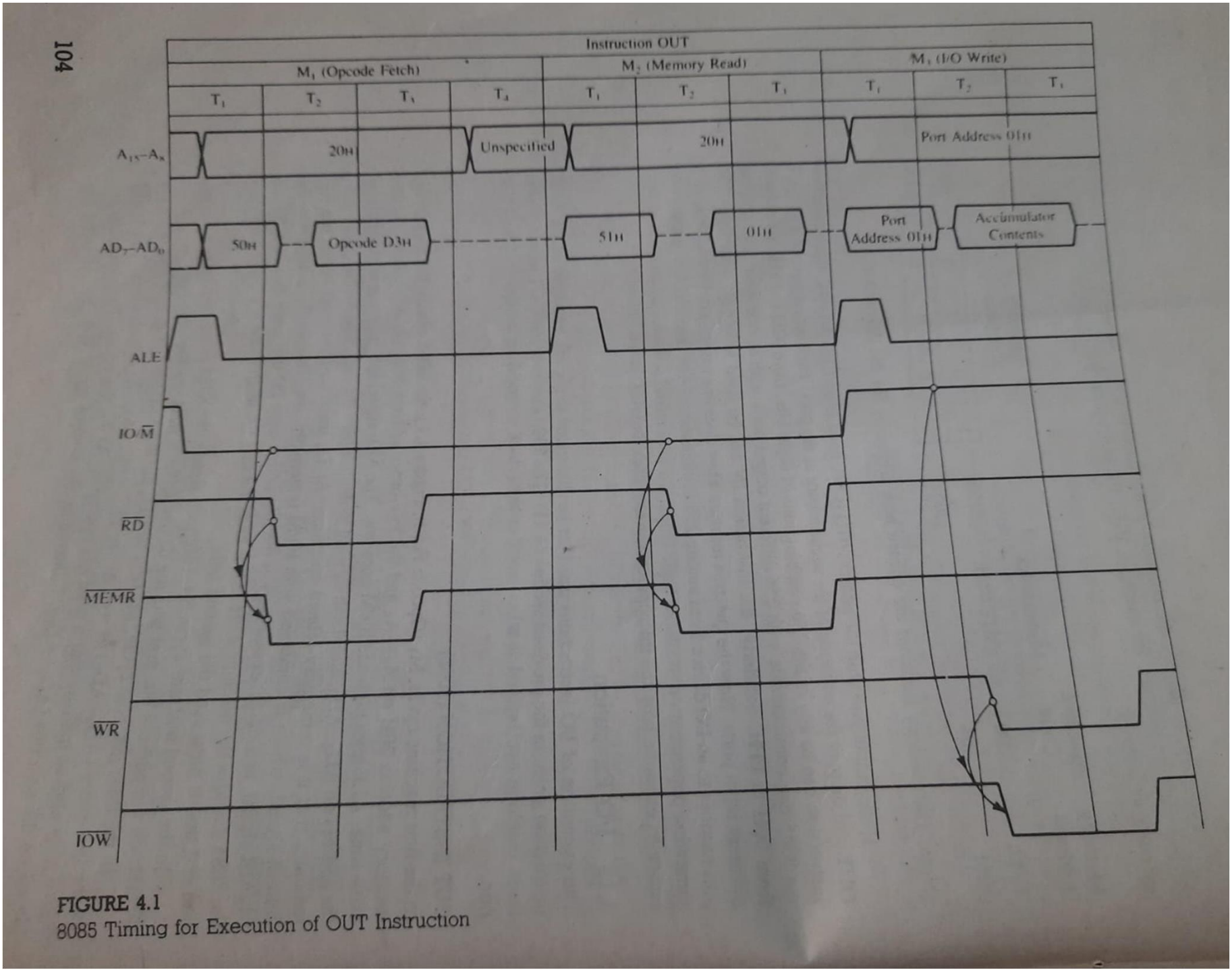


FIGURE 4.1
8085 Timing for Execution of OUT Instruction

Time Delay

```
        MVI C, FFH          7
LOOP:   DCR C               4
        JNZ  LOOP           10/7
```

T-states

- If clock frequency of the system is 1 MHz, MVI instruction will take 7 µs for its execution
- Total delay = 7 + (4+10) * 255 – 3 µs

```
        LXI    B, 1234H                    10
LOOP:   DCX  B          Doesn't set zero flag   6
        MOV A, C                           4
        ORA  B                             4
         JNZ   LOOP                        10/7
```

T-states

- Total delay = 10 + (6+4+4+10) * 4660 – 3 µs

Cont…

```
        MVI B, 12H      7T
LOOP2:  MVI C, 34H      7T
LOOP1:  DCR C           4T
        JNZ  LOOP1      10/7T
        DCR B           4T
        JNZ  LOOP2      10/7T
```

- Total delay = 7 + (21 + (14 * 52) -3) * 18 – 3 µs

Stack

- Memory locations in RAM (specified by a programmer) that is used for temporary storage of information during the execution of a program
- Stack instructions will not affect flags
- LXI SP, 16-bit address to define the stack
- PUSH Rp
- PUSH B/D/H/PSW
- SP is decremented and contents of higher-order register are copied into that location
- SP is decremented again and contents of low-order register are copied into that location
- POP Rp
- POP B/D/H/PSW
- Contents of memory location pointed by SP is copied into low-order register and then SP incremented by 1
- Contents of memory location pointed by SP is copied into higher-order register and then SP incremented by 1
- Program Status Word – contents of A and flags

Cont…

- LXI SP, 2099H
- LXI H, 42F2H
- PUSH H
- POP H

Subroutine

- A group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program
- CALL 16-bit address
- Saves contents of PC on the stack. Decrements contents of SP by two
- Transfers program sequence to the subroutine
- RET
- Inserts two bytes from TOS into PC and increments SP by two
- Unconditionally returns from a subroutine

Cont…

2000 LXI SP, 2400H

…….

2040 CALL 2070H

2043 Next instruction

…….

205F HLT

2070 Subroutine begins

…….

207F RET

M_1 – opcode fetch
Contents of PC (2040H) are placed on the address bus, and instruction code CD (corr: to CALL) is fetched using data bus. At the same time, PC is incremented to 2041H. After the instruction is decoded and executed, SP is decremented to 23FFH

M_2 & M_3 – Memory Read
70H is fetched and placed in register Z. 20H is fetched and placed in W register. PC is upgraded to 2043H during M_3

M_4 & M_5 – higher-order byte of PC (20H) is placed on the data bus and stored in stack location 23FFH. At the same time, SP is decremented to 23FEH. Low-order byte of PC (43H) is placed on the data bus and stored in stack location 23FEH

Next instruction cycle – Program execution sequence is transferred to CALL location 2070H by placing contents of W and Z registers (2070H) on the address bus

Cont…

RET (C9)

- M_1: opcode fetch – C9
- M_2: contents of SP are placed on the address bus. Data byte 43H is fetched from TOS and stored in Z register. SP is incremented to 23FFH
- M_3: next byte 20H is copied from stack and stored in register W. Increment the SP to 2400H
- Next instruction cycle: The program sequence is transferred to location 2043H by placing contents of W/Z registers on the address bus

Nesting of Subroutines: a subroutine calls another subroutine

Multiple-ending Subroutines: contains multiple return statements

Cont…

Conditional Call	
CC	Call subroutine if Carry flag is set
CNC	Call subroutine if Carry flag is reset
CZ	Call subroutine if Zero flag is set
CNZ	Call subroutine if Zero flag is reset
CM	Call subroutine if Sign flag is set
CP	Call subroutine if Sign flag is reset
CPE	Call subroutine if Parity flag is set
CPO	Call subroutine if Parity flag is reset
Conditional Return	
RC	Return if Carry flag is set
RNC	Return if Carry flag is reset
RZ	Return if Zero flag is set
RNZ	Return if Zero flag is reset
RM	Return if Sign flag is set
RP	Return if Sign flag is reset
RPE	Return if Parity flag is set
RPO	Return if Parity flag is reset

Traffic Light Controller

Light	Data bit	Seconds
Green	D_0	15
Yellow	D_2	5
Red	D_4	20
Walk	D_6	15
Don't walk	D_7	25

seconds	Don't walk D_7	Walk D_6	D_5	Red D_4	D_3	Yellow D_2	D_1	Green D_0	Hex code
15	0	1	0	0	0	0	0	1	41H
5	1	0	0	0	0	1	0	0	84H
20	1	0	0	1	0	0	0	0	90H

Cont…

```
        LXI SP, 4200H
START:  MVI A, 41H
        OUT PORT#
        MVI B, OFH
        CALL DELAY
        MVI A, 84H
        OUT PORT#
        MVI B, 05H
        CALL DELAY
        MVI A, 90H
        OUT PORT#
        MVI B, 14H
        CALL DELAY
        JMP START
```

Multiple calling subroutine

```
DELAY:  PUSH D
        PUSH PSW
SECOND:LXI D, COUNT
LOOP:   DCX D
        MOV A, D
        ORA E
        JNZ LOOP
        DCR B
        JNZ SECOND
        POP PSW
        POP D
        RET
```

Interrupts

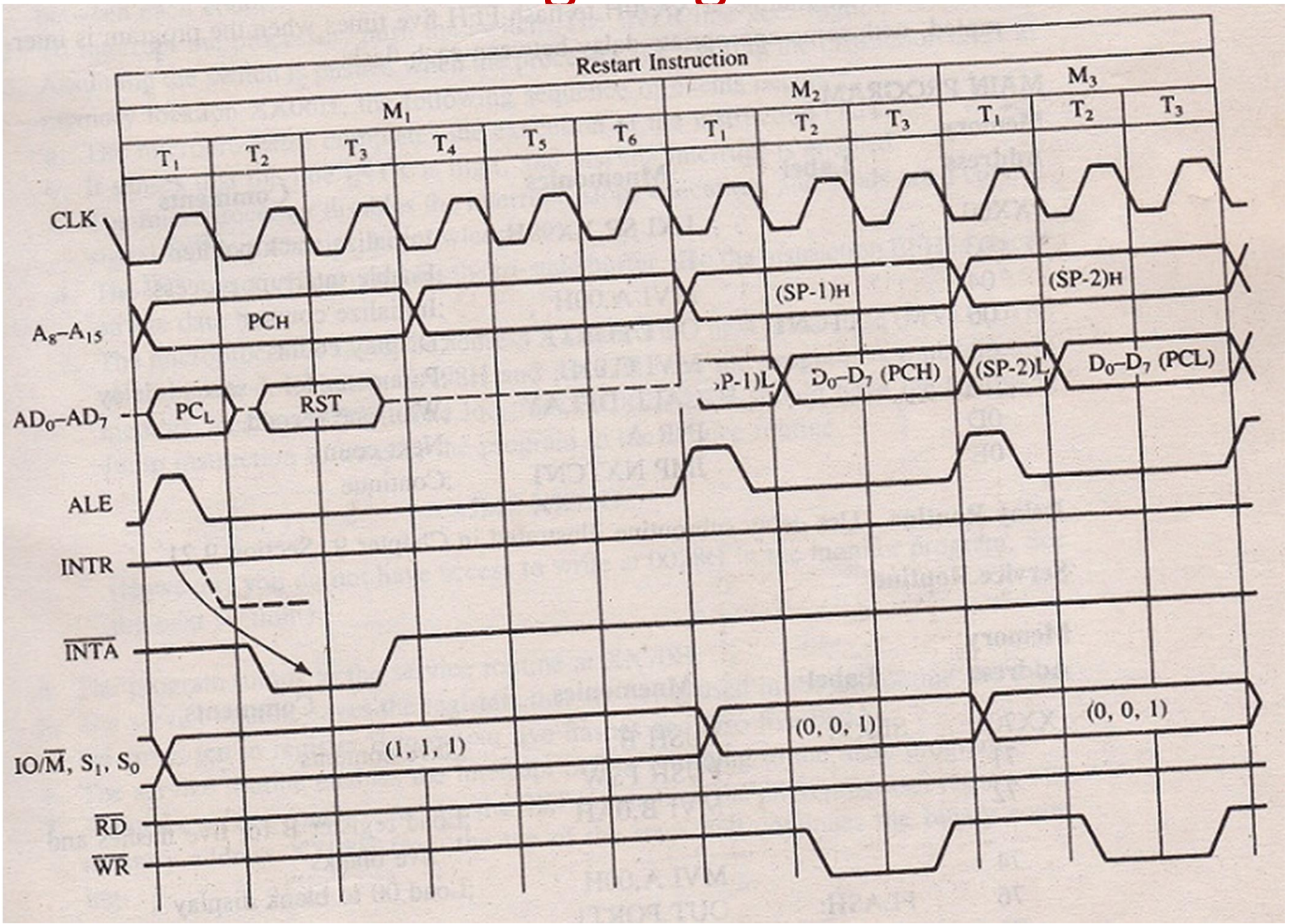
- Controlled by Interrupt Enable flip flop. If flip flop is enabled (using EI) and input to INTR goes high, µp is interrupted
- µp completes the current instruction, and disables the interrupt using DI and sends $\overline{\text{INTA}}$ signal
- Insert an RST instruction through additional hardware
- µp saves memory address of the next instruction on stack. Transfer program control to CALL location in page 00H
- The task to be performed is written as a subroutine (service routine)
- Service routine contains EI instruction to enable interrupt again
- At the end of the subroutine RET instruction transfers control back to interrupted program

Cont…

Mnemonic	hex code	Call location
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

- If the service routine requires more that 8 locations, the routine is written in memory, and Jump instruction is written at the respective location to connect with the routine

Timing Diagram



Interrupt Service Routine

Service routine to flash FFH

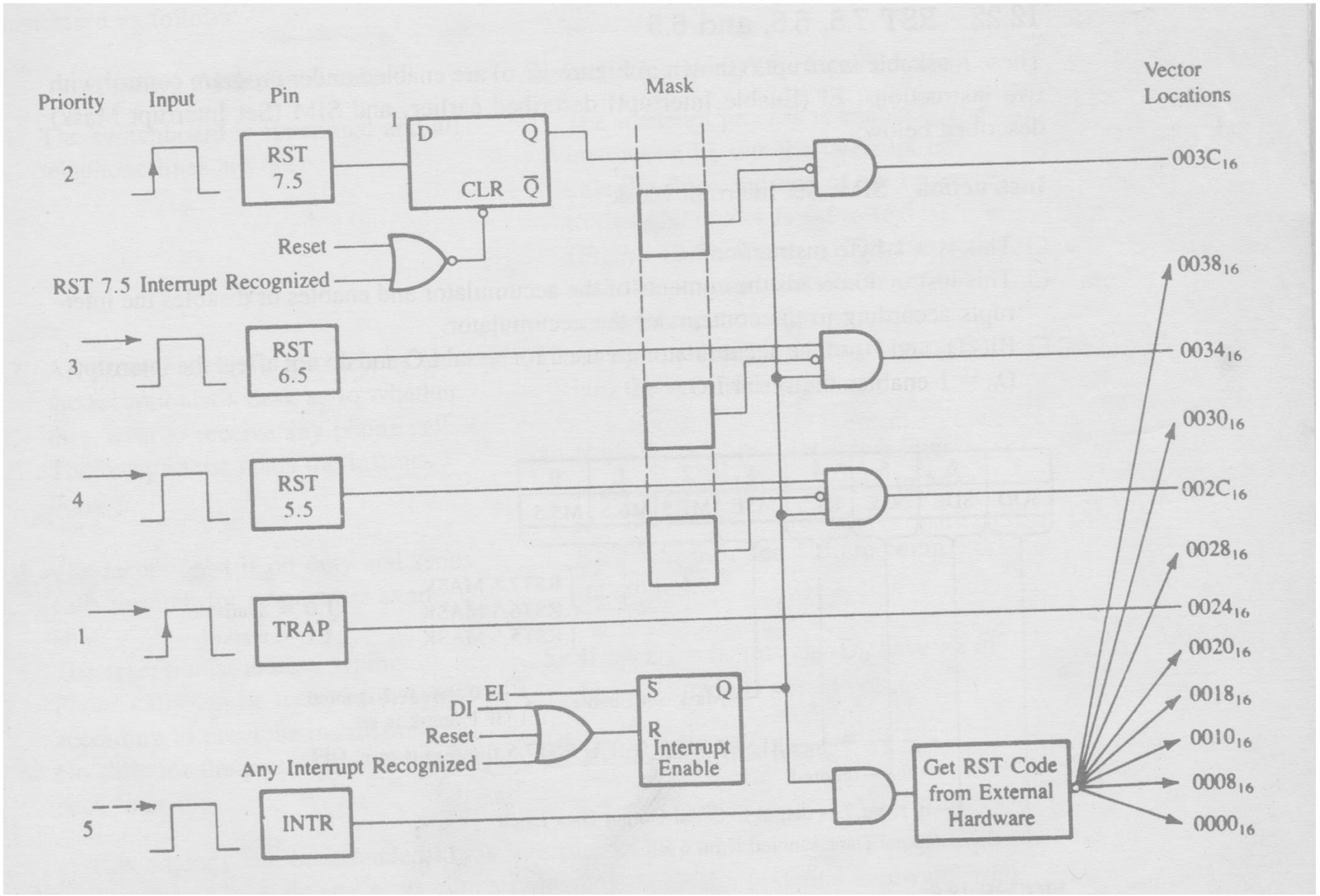
```
SERV:   PUSH B
        PUSH PSW
        MVI  B, 0AH
        MVI  A, 00H
FLASH:  OUT  PORT1
        MVI  C, 01H
        CALL DELAY
        CMA
        DCR  B
        JNZ  FLASH
        POP  PSW
        POP  B
        EI
        RET
```

Main program – count continuously

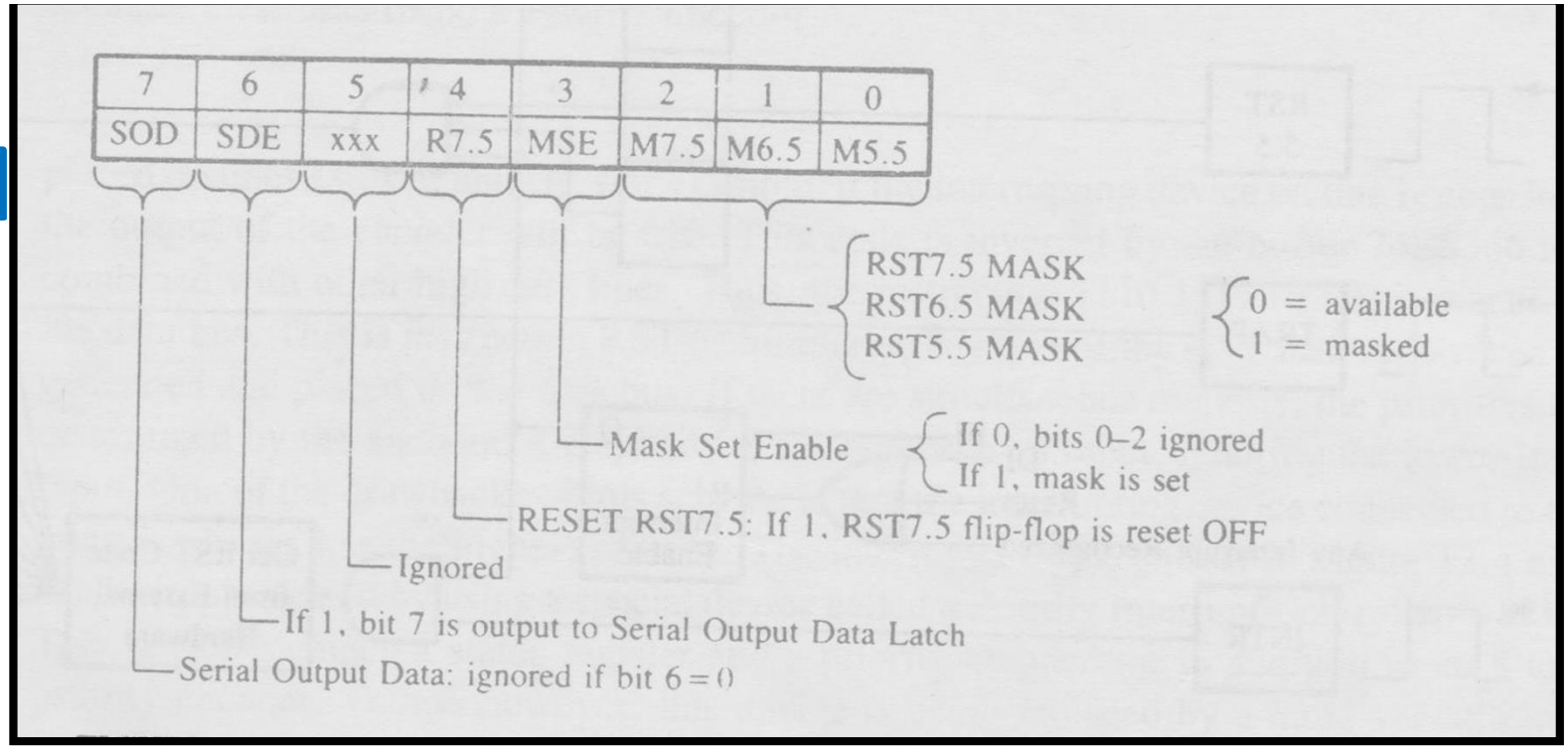
```
        LXI  SP, 4399H
        EI
        MVI  A, 00H
NXTCNT: OUT  PORT1
        MVI  C, 01H
        CALL DELAY
        INR  A
        JMP  NXTCNT
```

Interrupt Inputs

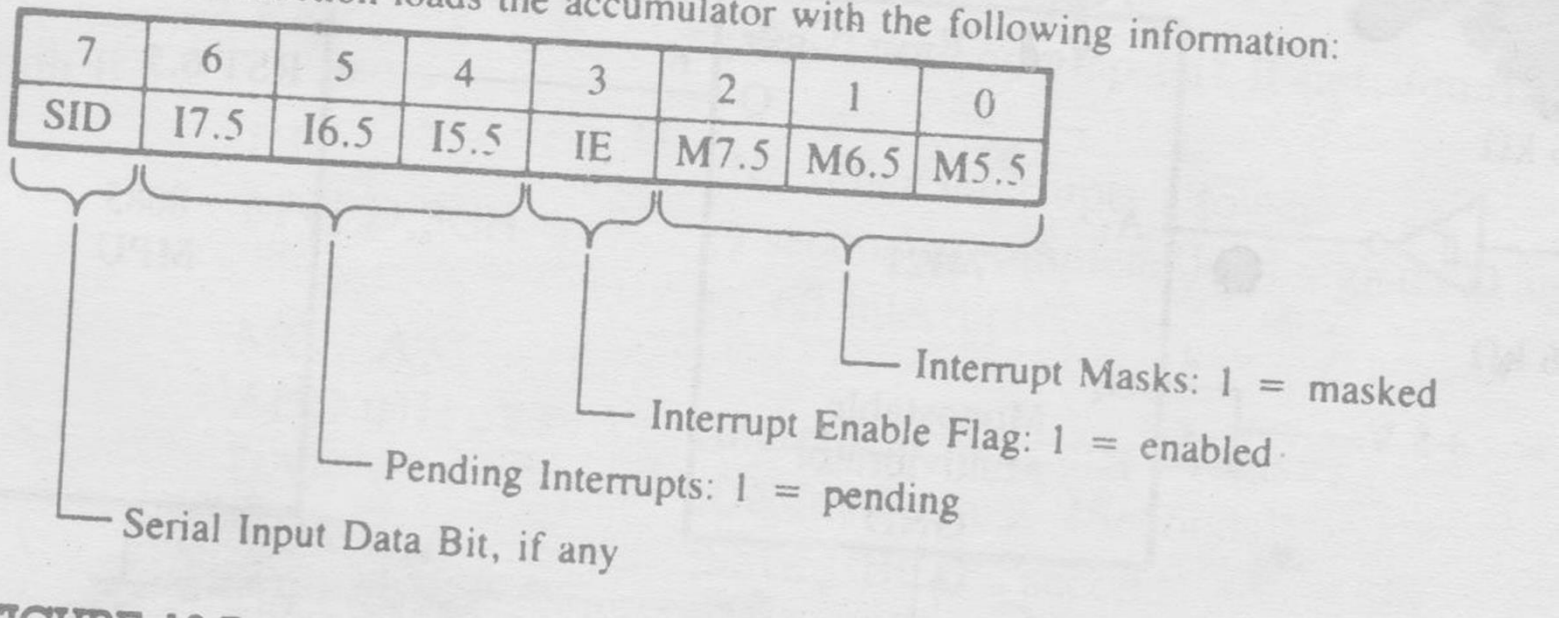
- INTR, RST 5.5, RST 6.5, RST 7.5, TRAP – in the priority order. $\overline{\text{INTA}}$ for INTR only
- TRAP: NMI, used for critical events, no enable/disable, input should go high and stay high
- RST interrupts are enabled with EI and SIM (Set Interrupt Mask)
- SIM: this one byte instruction enables or disables the interrupts by reading the contents of accumulator
- RIM (Read Interrupt Mask): this one byte instruction loads accumulator with eight bits indicating the current status of interrupt masks, interrupt enable, pending interrupts, and serial input data


Priority
Input
Pin
Mask
Vector Locations
2
RST 7.5
1
D
Q
CLR
$\overline{Q}$
Reset
RST 7.5 Interrupt Recognized
$003C_{16}$
3
RST 6.5
0034_{16}
4
RST 5.5
$002C_{16}$
1
TRAP
0024_{16}
EI
DI
Reset
Any Interrupt Recognized
S
Q
R
Interrupt Enable
5
INTR
Get RST Code from External Hardware
0038_{16}
0030_{16}
0028_{16}
0020_{16}
0018_{16}
0010_{16}
0008_{16}
0000_{16}

SIM



The RIM instruction loads the accumulator with the following information:



Cont…

```
        RIM
        MOV B, A
        ANI   20H          Check whether RST 6.5 is pending
        JNZ   NEXT
        EI
        RET                RST 6.5 isn't pending, return to main program
NEXT:   MOV  A, B          RST 6.5 is pending
        ANI   0DH          Enable RST 6.5
        ORI   08H          Enable SIM
        SIM
        JMP   SERV         Jump to service routine for RST 6.5
```

Interrupt Driven Clock

```
0034    JMP RWM        Monitor program – RST 6.5

        LXI SP, STACK
        RIM
        ORI 08H        Bit pattern to enable RST 6.5
        SIM            Enable RST 6.5
        LXI B, 0000H   Register B for minutes, register C for seconds
        MVI D, 3CH     Register D to count 60 interrupts
        EI
DISPLAY:MOV A, B
        OUT PORT1      Display minutes at PORT1
        MOV A, C
        OUT PORT2      Display seconds at PORT2
        JMP DISPLAY
RWM:    JMP TIMER      Go to TIMER routine to upgrade the clock
```

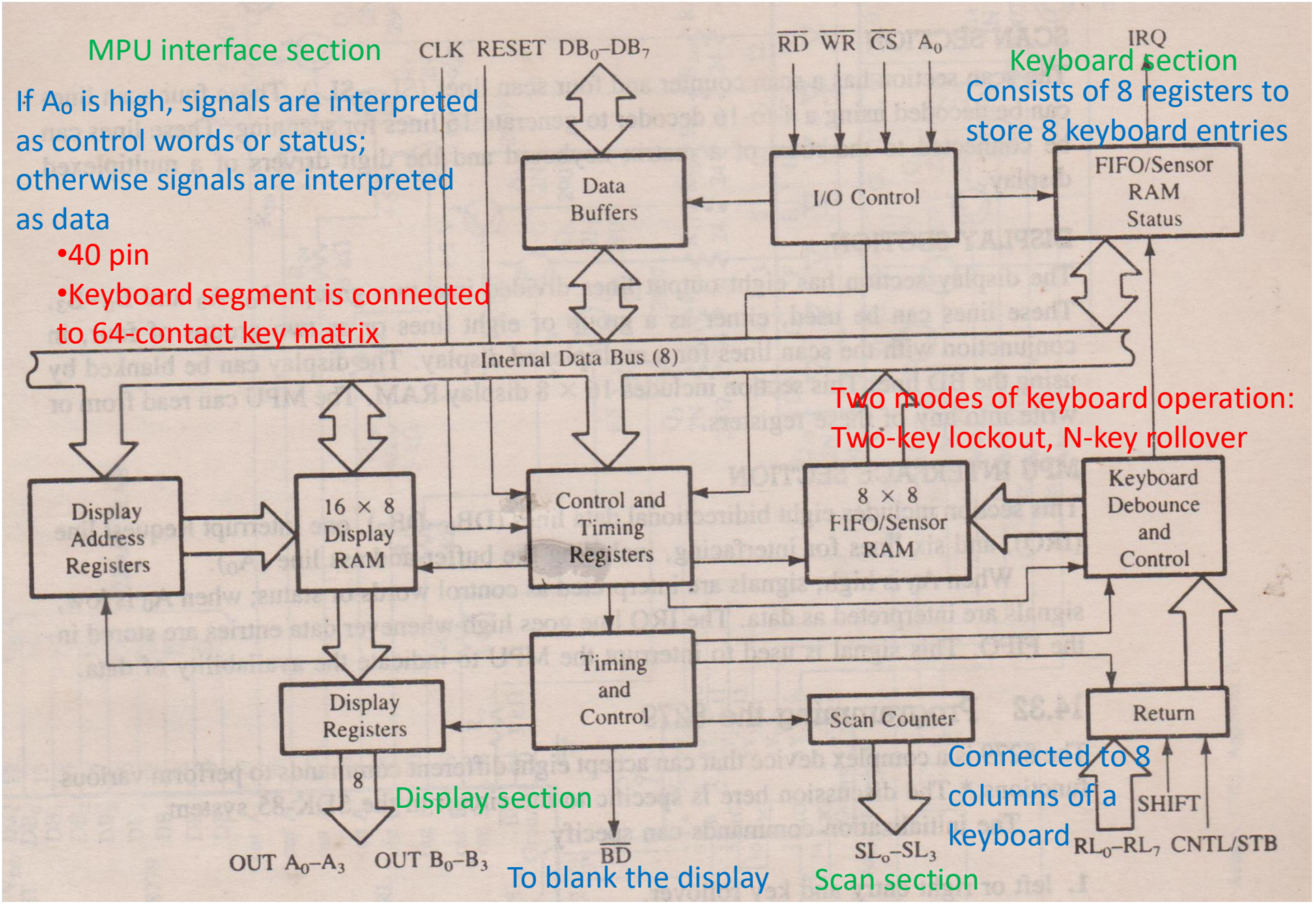
Cont…

Interrupt Service Routine

```
TIMER:  DCR D        One interrupt occurred, reduce count by 1
        EI
        RNZ          Has 1 second elapsed? If not, return
        DI
        MVI D, 3CH   1 second completed; load register D with 60
        MOV A, C
        ADD 01H      Increment "Second" register
        DAA          Decimal adjust "Seconds"
        MOV C, A     Save BCD "Seconds"
        CPI 60H
        EI
        RNZ          Is time = 60 seconds? If not, return
        DI
        MVI C, 00H   60 seconds completed, clear "Second" register
        INR B        Increment "Minutes"
        RET          1 minute elapsed
```

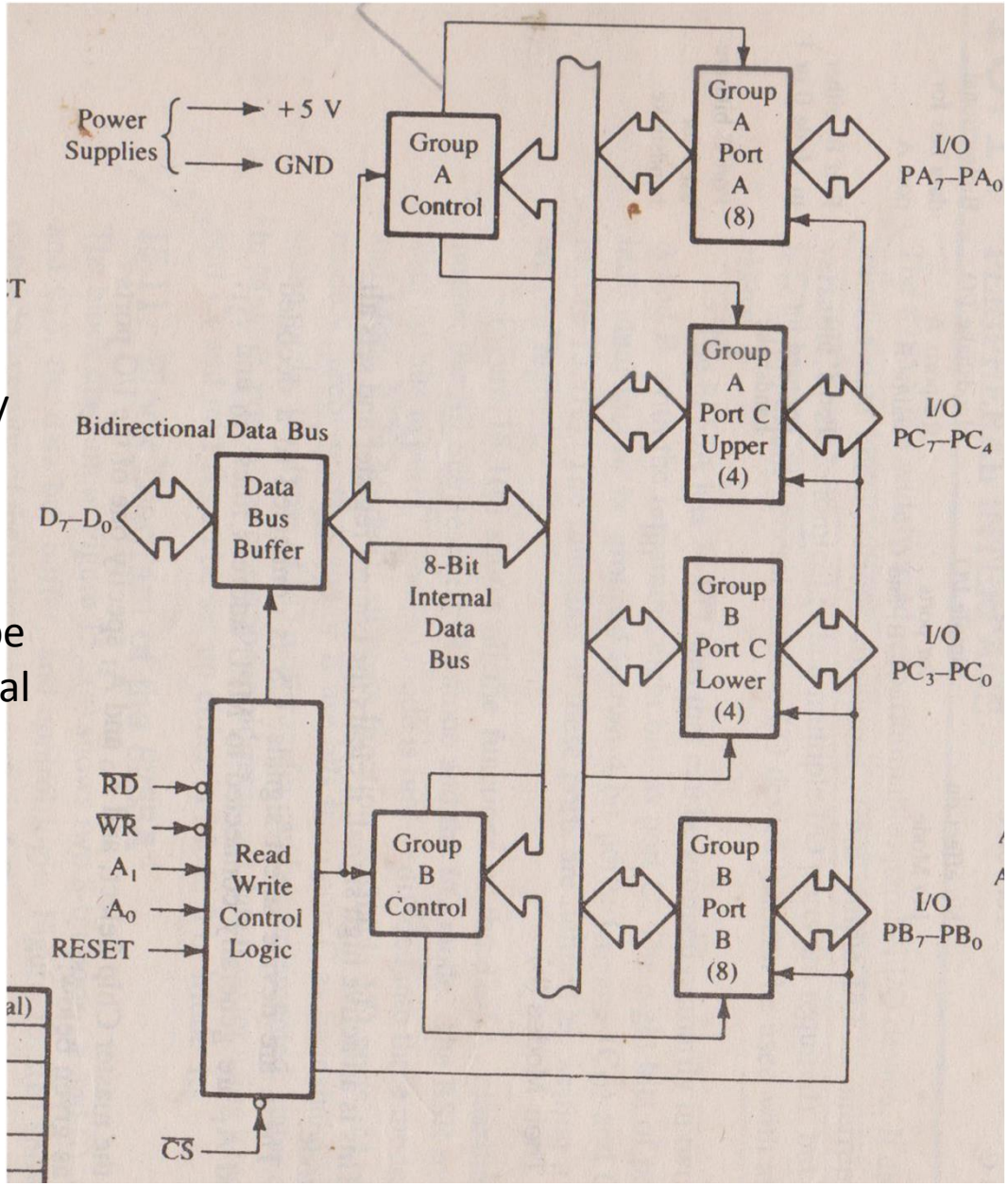
8279 Programmable Keyboard/Display Interface

IRQ goes high if RAM contains data



8255A Programmable Peripheral Interface

BSR mode: To set/ reset bits in port C
I/O mode
Mode 0: all ports functions as simple I/O ports
Mode 1: ports A and/ or B use bits from port C as handshake signals
Mode 2: port A can be set up for bidirectional data transfer



Cont…

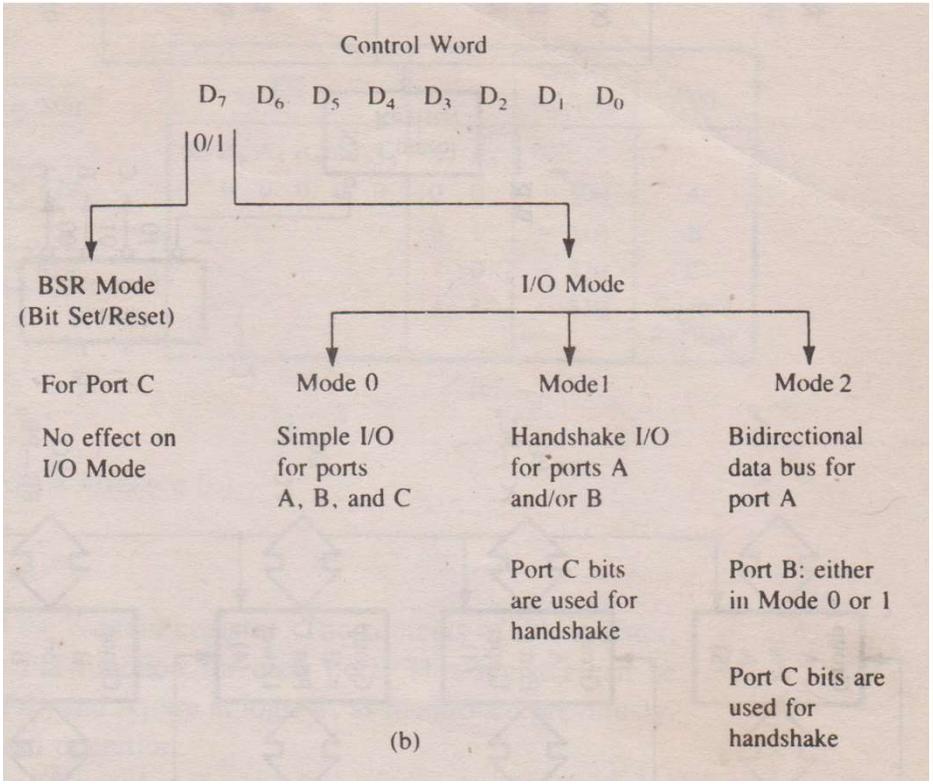
$\overline{CS}$	A_1	A_0	Selected
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control register
1	X	X	8255A isn't selected

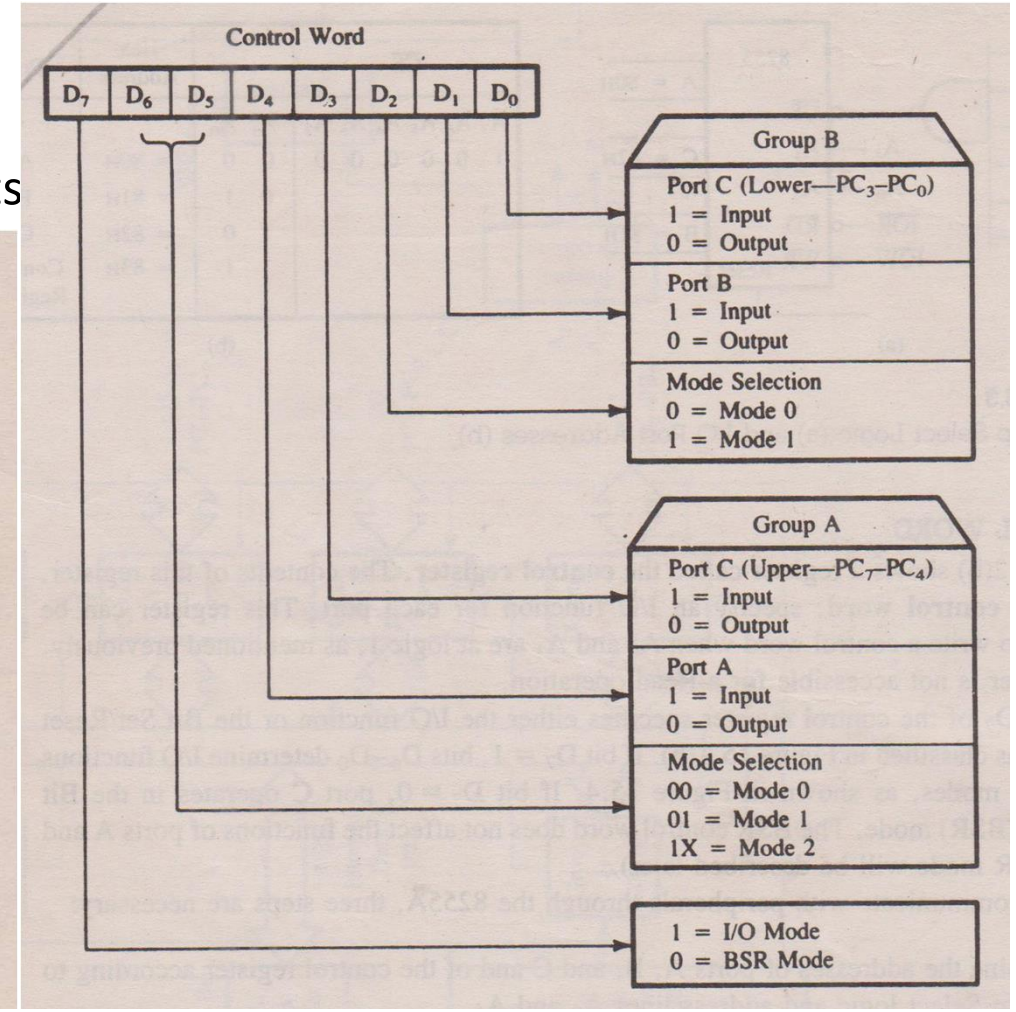
- Control logic
- $\overline{RD}$: MPU reads data from a selected I/O port of 8255A
- $\overline{WR}$: MPU writes into a port or control register
- RESET: clears the control register and sets all ports in input mode
- $\overline{CS}$, A_0, A_1: device select signals
- Control word (contents of control register) specify an I/O function for each port

Cont…

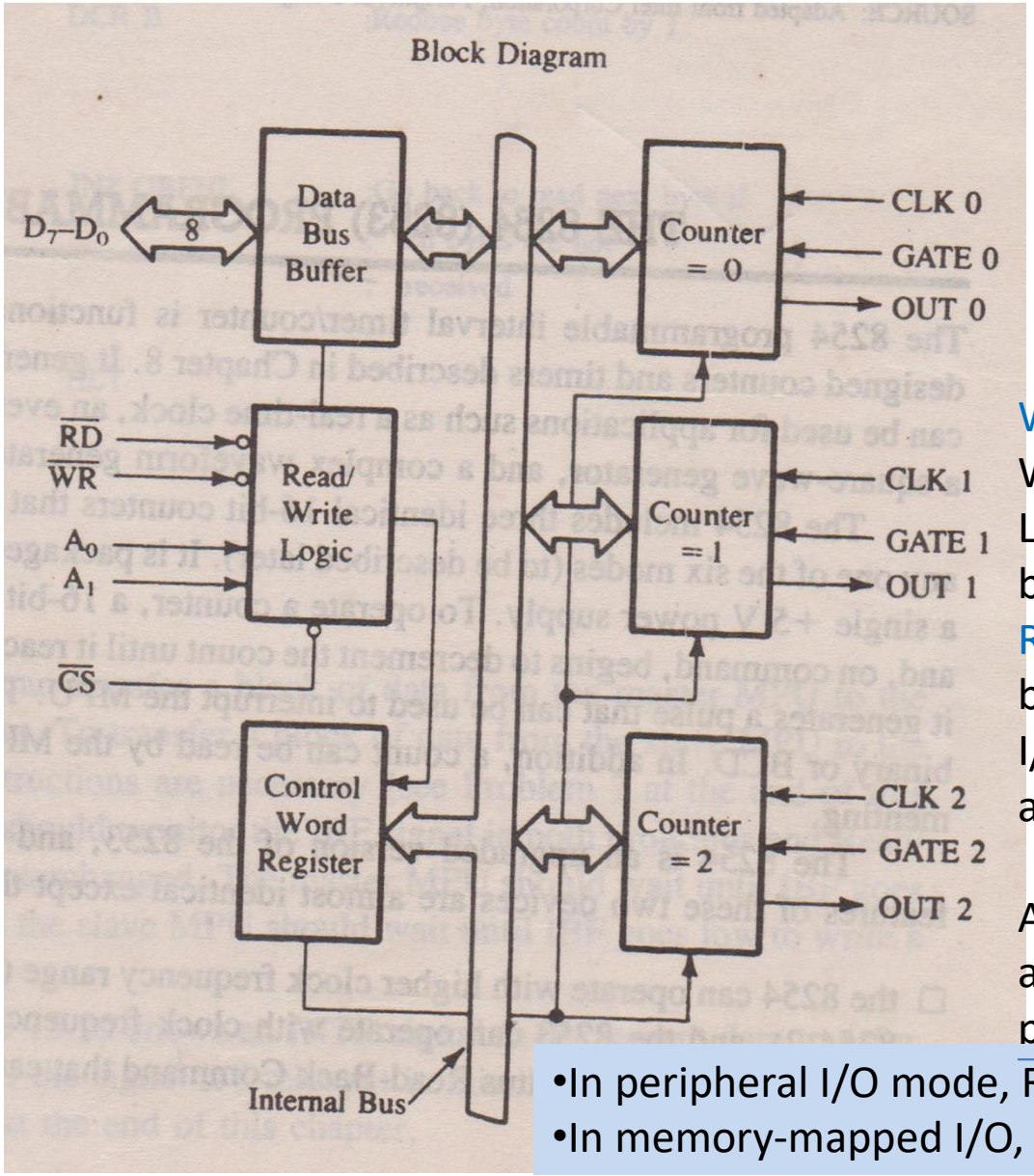
To communicate with peripherals through 8255A

- Determine address of ports and control register
- Write control word into control register
- Write I/O instructions through ports





8254 (8253) Programmable Interval Timer



- 24 pin, +5V
- Binary or BCD count

A_1	A_0	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control register

Write operation: To initialize a counter,
Write control word into the control register
Load low-order byte followed by high-order byte of a count to counter register

Read (count) operation: counting is stopped by controlling the gate/clock input, and two I/O read operations (one for low-order byte and other for high-order byte)are performed

OR

A control word is written into the CR to latch a count, and two I/O read operations are performed by MPU

- In peripheral I/O mode, $\overline{RD}$ and $\overline{WR}$ are connected to $\overline{IOR}$ and $\overline{IOW}$
- In memory-mapped I/O, these are connected to MEMR and MEMW

Cont…

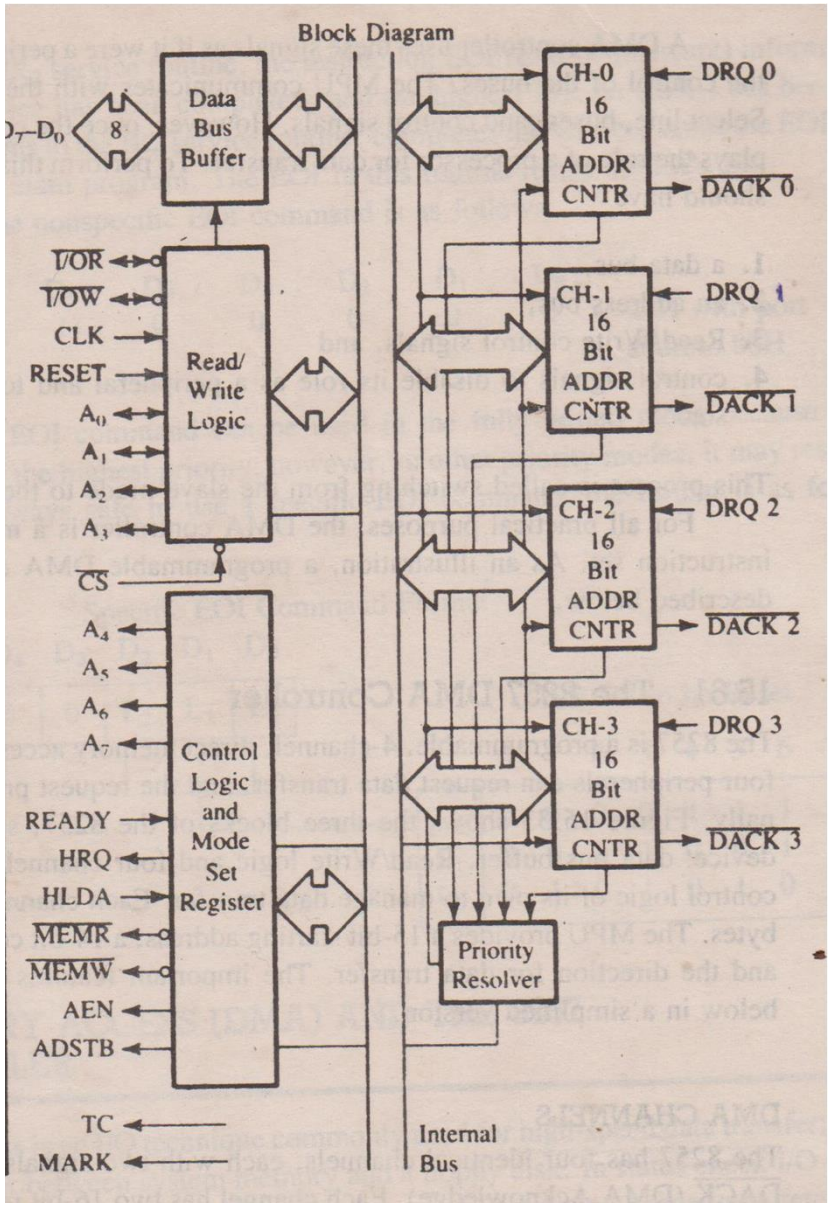
Modes

- Mode 0: interrupt on TC – initially OUT is low, count in the counter is decremented in every cycle until count = 0, and OUT goes high
- Mode 1: h/w triggered one shot – initially OUT is high. When gate is triggered, OUT goes low, and at the end of count OUT is high, thus generating a one shot
- Mode 2: rate generator – it is used to generate pulse equal to the clock period
- Mode 3: square wave generator - when count is loaded, OUT is high. Count is decremented by two at every clock cycle, and it reaches 0, OUT goes low, and the count is reloaded
- Mode 4: s/w triggered strobe – initially OUT is high; it goes low for one clock period at the end of the count
- Mode 5: h/w triggered strobe – similar to Mode 4, except that it triggered by rising pulse at the gate

Direct Memory Access

- An I/O technique, commonly used for high speed data transfer
- MPU releases the control of buses to DMA controller. The controller manages data transfer between memory and a peripheral
- HOLD from another master requesting the use of address and data buses
- HLDA: MPU relinquishes the control of buses

8257 DMA Controller



•Each channel has two 16-bit registers – one for memory address and other for count register

•ADSTB generates additional eight address lines. 8257 has only eight address lines

•AEN disables system data bus and control bus.

It is necessary to switch 8257 from slave mode (treated as peripheral) to master mode (processor)

Slave mode: MPU selects DMA controller through $\overline{CS}$

MPU writes command mode and terminal count in channel registers (A_0 – A_3, $\overline{IOR}$, $\overline{IOW}$)

Master mode: When the peripheral is ready for data transfer it sends DRQ

When DRQ received and channel enabled, send HRQ

MPU relinquishes the control of buses and sends HLDA to 8257

On receiving HLDA, send $\overline{DACK}$ to peripheral

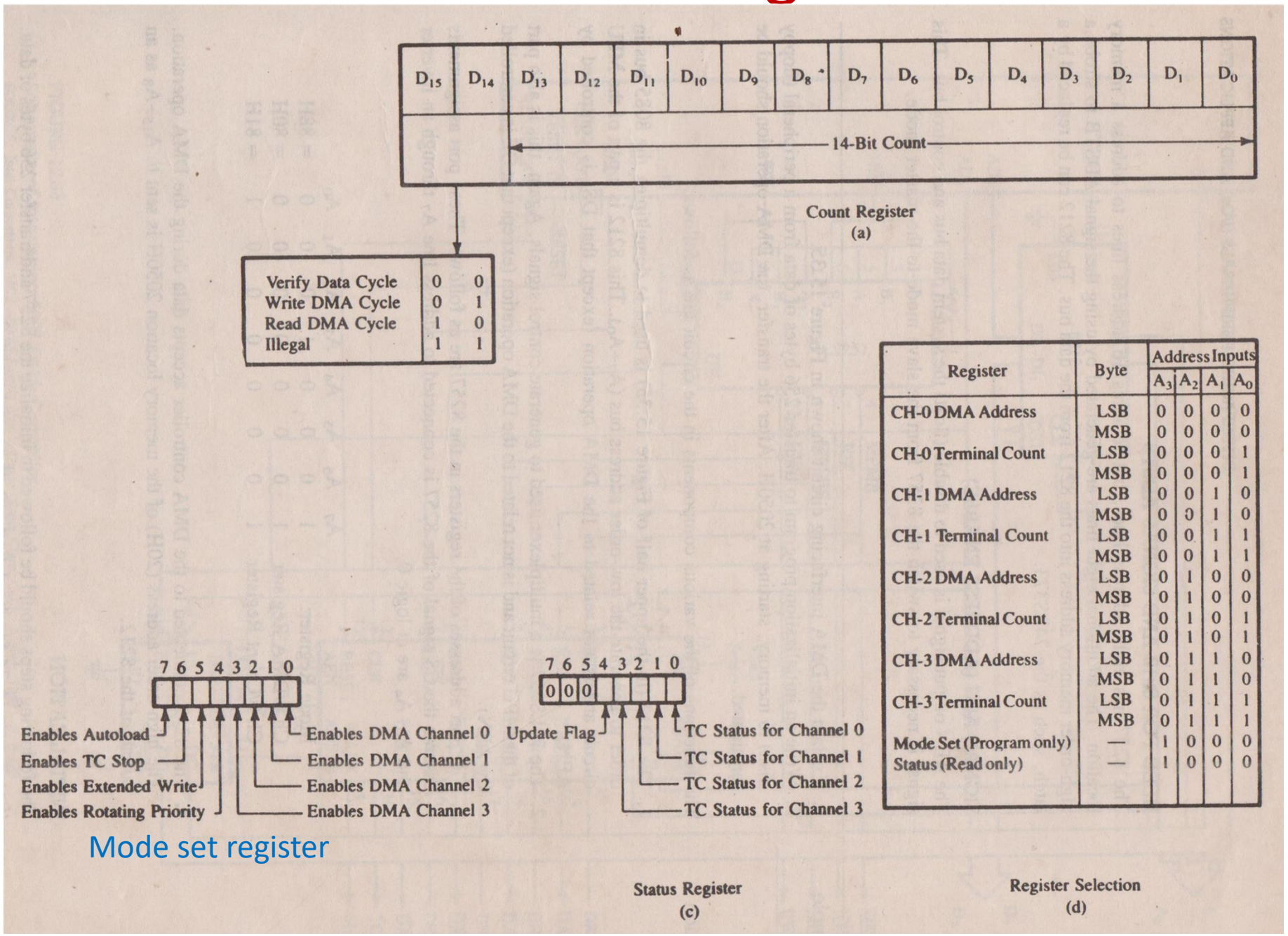
8257 enables AEN

Low-order byte of memory location is placed on A_7 – A_0 of 8257

When AEN is high, ADSTB goes high and places high-order address generated by 8212 on A_{15} – A_8.

Data transfer continues

DMA channel registers



Register	Byte	Address Inputs A_3	A_2	A_1	A_0
CH-0 DMA Address	LSB	0	0	0	0
	MSB	0	0	0	0
CH-0 Terminal Count	LSB	0	0	0	1
	MSB	0	0	0	1
CH-1 DMA Address	LSB	0	0	1	0
	MSB	0	0	1	0
CH-1 Terminal Count	LSB	0	0	1	1
	MSB	0	0	1	1
CH-2 DMA Address	LSB	0	1	0	0
	MSB	0	1	0	0
CH-2 Terminal Count	LSB	0	1	0	1
	MSB	0	1	0	1
CH-3 DMA Address	LSB	0	1	1	0
	MSB	0	1	1	0
CH-3 Terminal Count	LSB	0	1	1	1
	MSB	0	1	1	1
Mode Set (Program only)	—	1	0	0	0
Status (Read only)	—	1	0	0	0

8086 Architecture

Execution Unit (EU)

- (i) Tells to BIU from where to fetch instructions or data, (ii) decode and execute instructions
- Control circuitry directs internal operations
- Decoder translates instructions into a series of actions which EU carries out
- ALU can do add, subtract, AND, OR, XOR, increment, decrement or shift operations

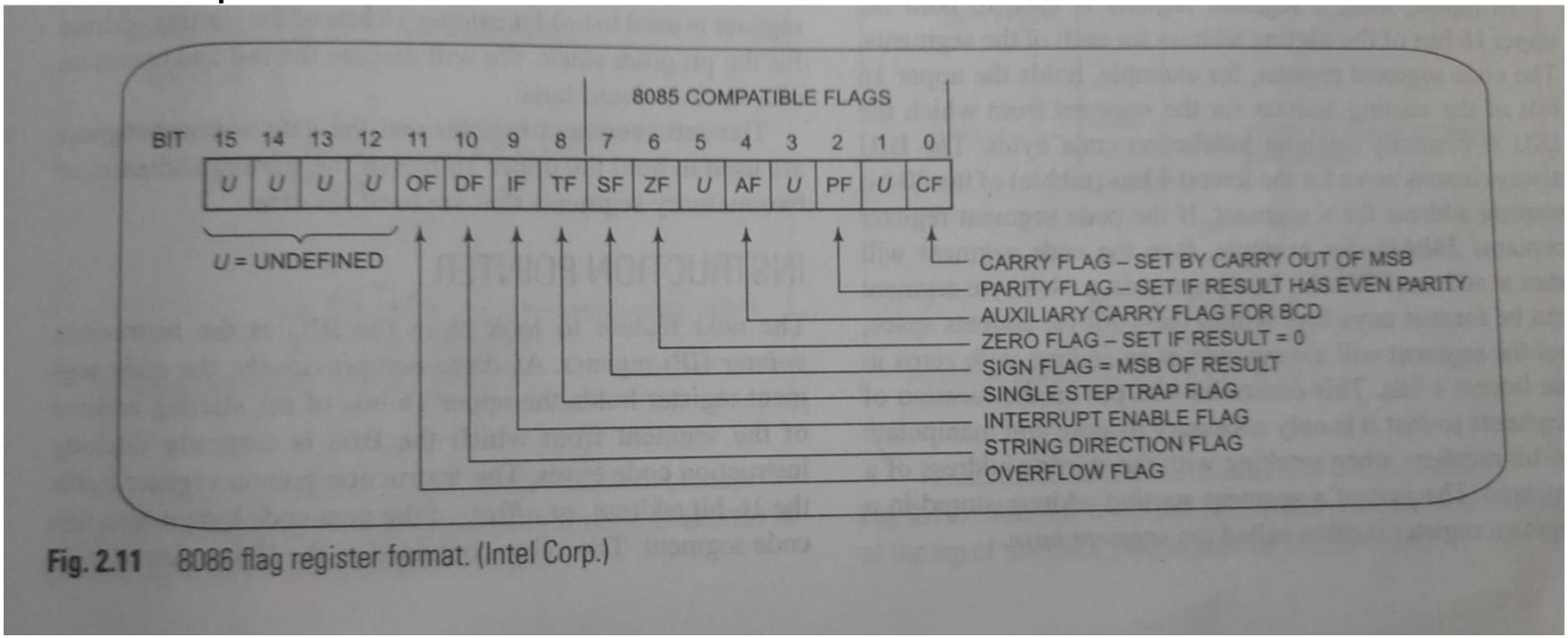


Fig. 2.11 8086 flag register format. (Intel Corp.)

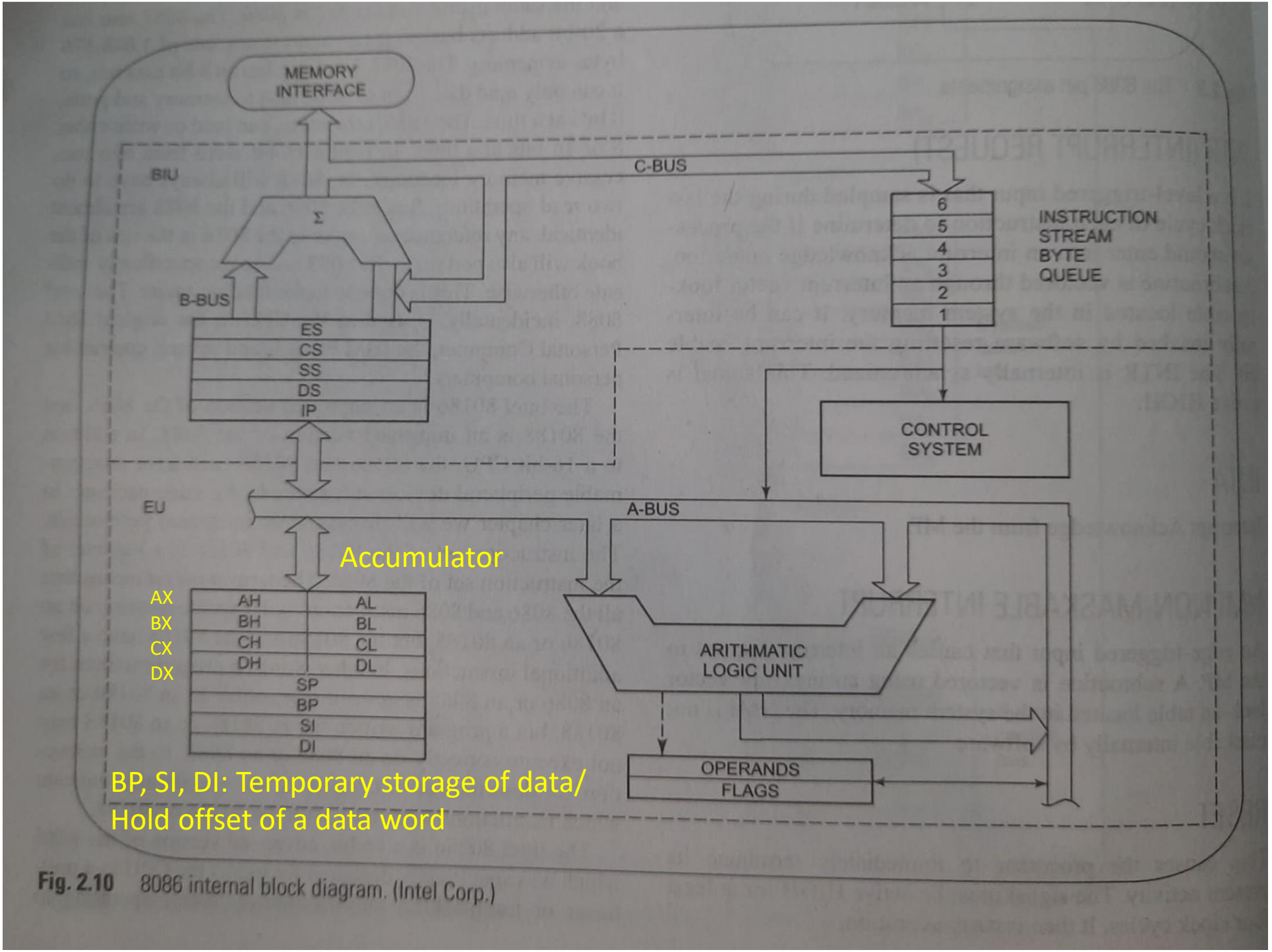


Fig. 2.10 8086 internal block diagram. (Intel Corp.)

Bus Interface Unit (BIU)

- Fetches up to six instruction bytes and stores in register Queue while EU executes an instruction - pipelining
- EU simply reads the next instruction from the queue, and not from memory

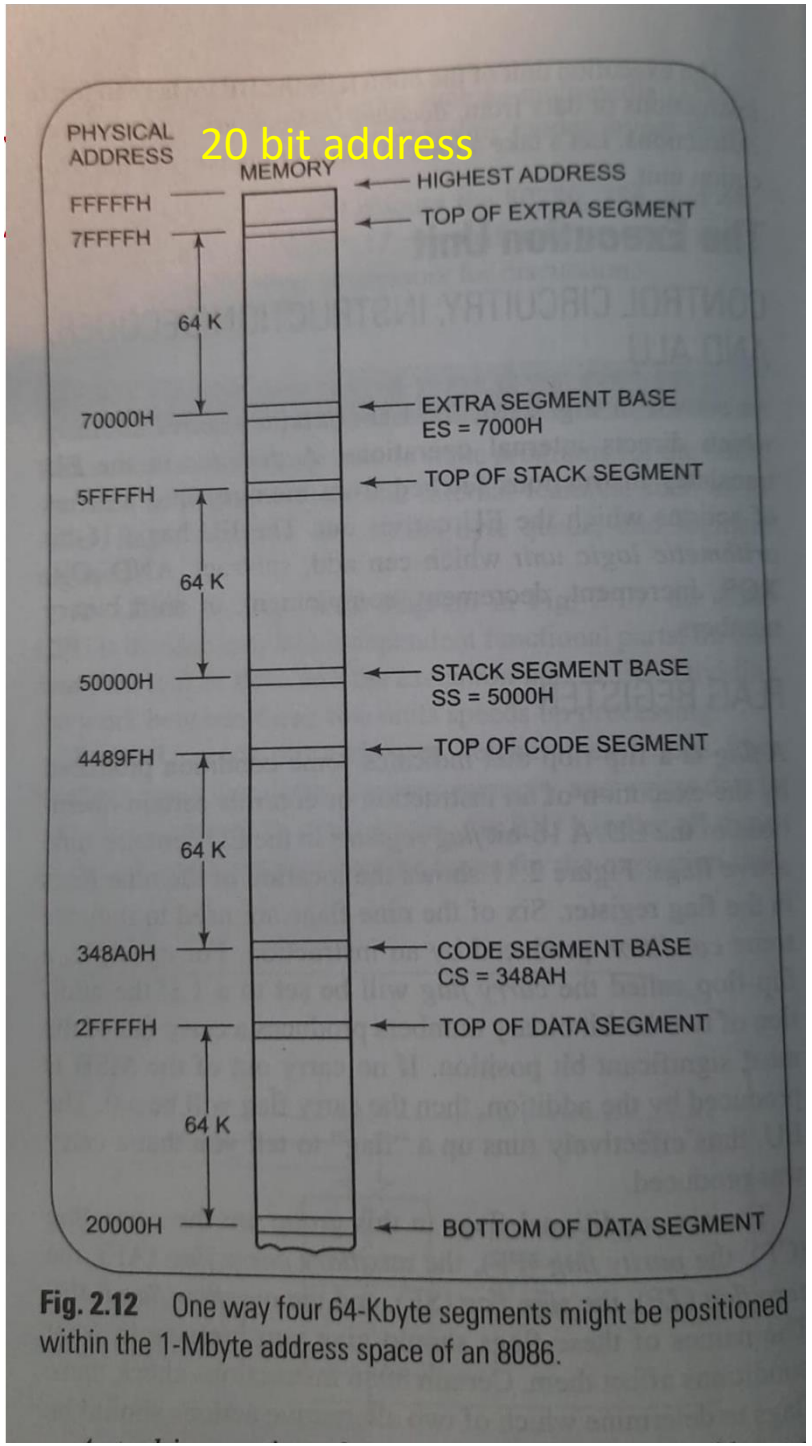


Fig. 2.12 One way four 64-Kbyte segments might be positioned within the 1-Mbyte address space of an 8086.

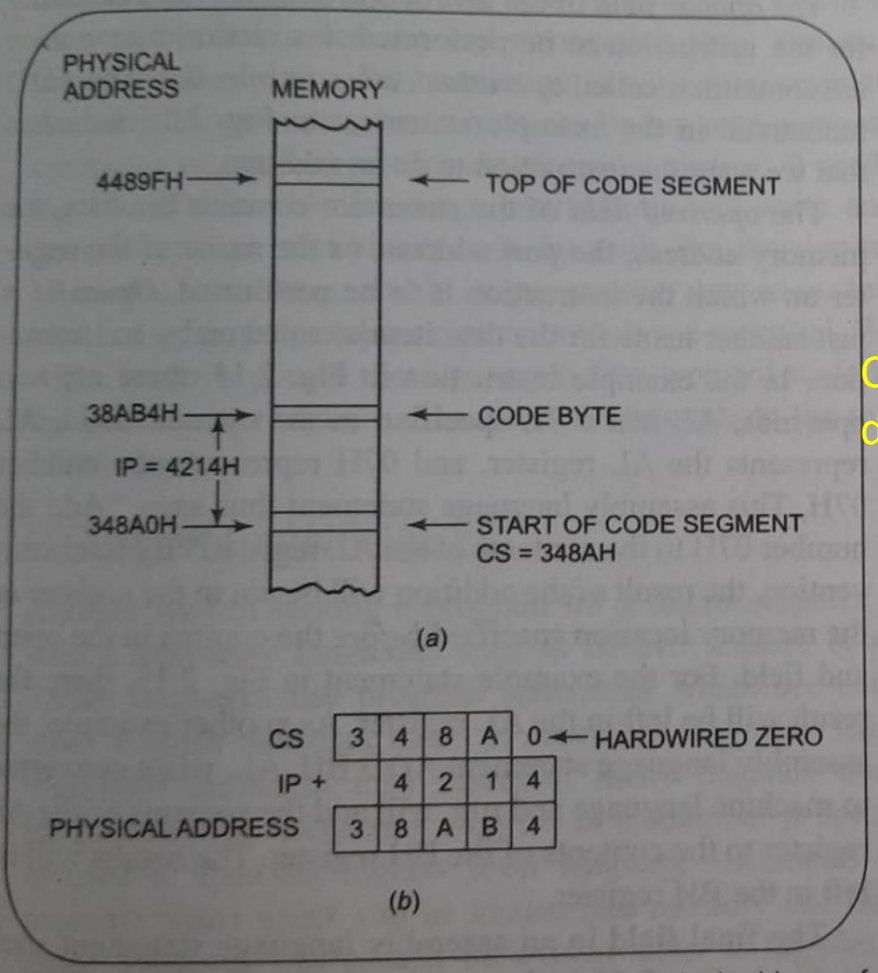
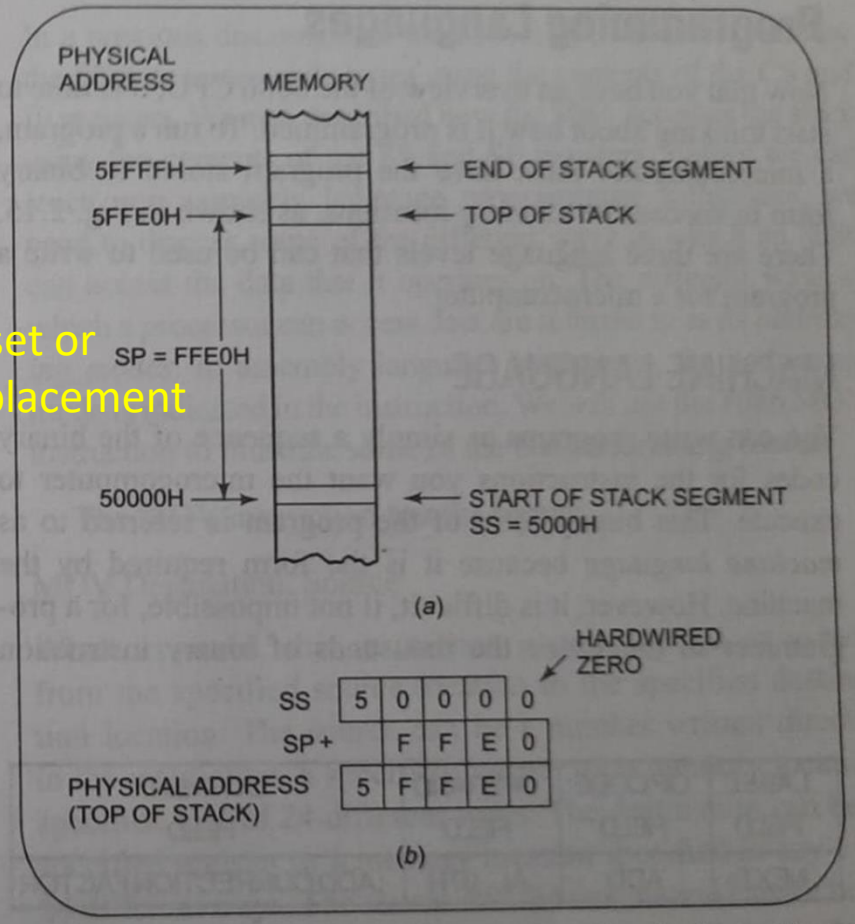


Fig. 2.13 Addition of IP to CS to produce the physical address of the code byte, (a) Diagram, (b) Computation.



Offset or displacement

Fig. 2.14 Addition of SS and SP to produce the physical address of the top of the stack. (a) Diagram, (b) Computation.

Addressing Mode

- Immediate addressing

  MOV CX, 2050H

  MOV CL, 35H

- Register addressing

  MOV CX, AX

  MOV CH, AL

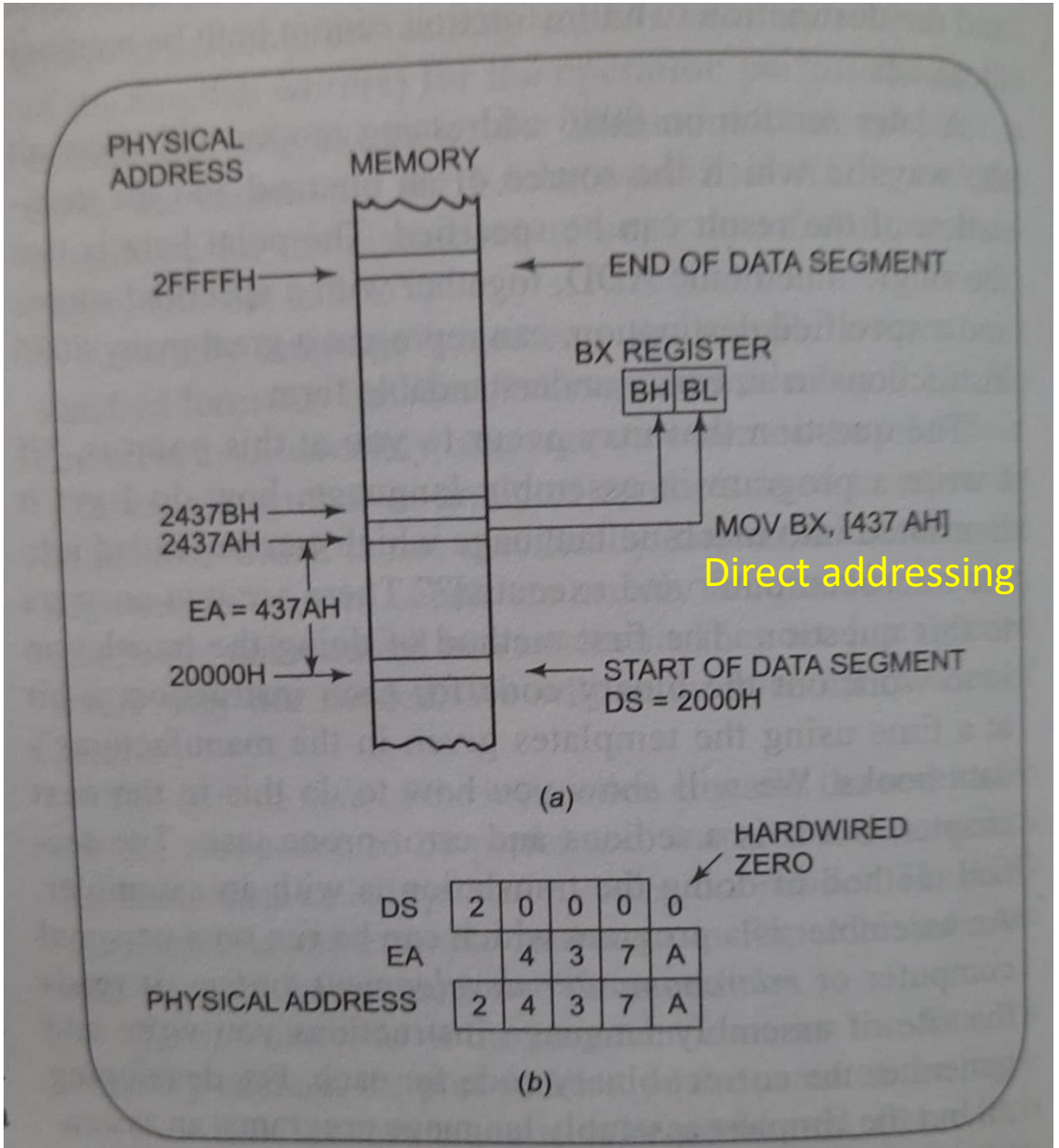
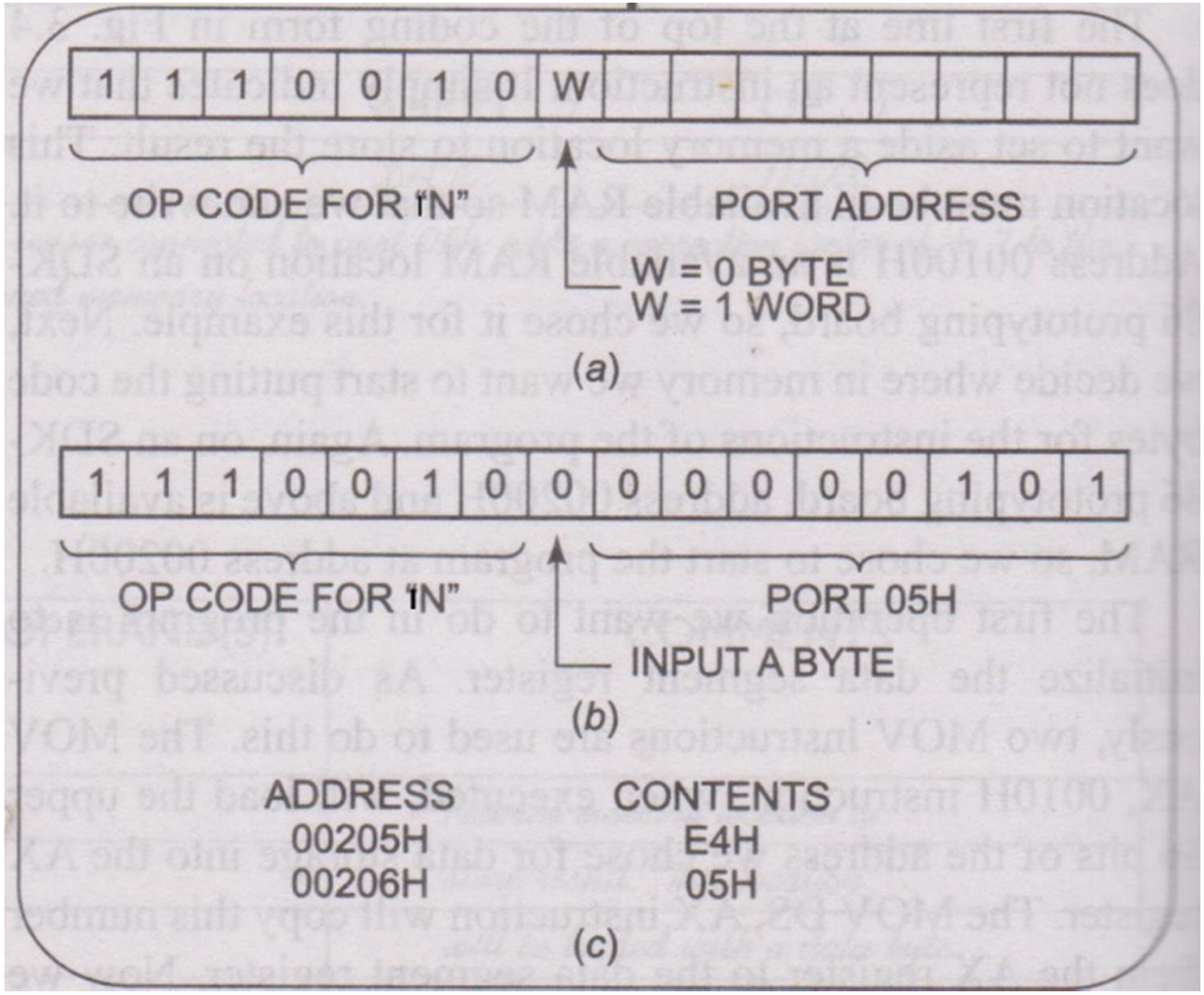
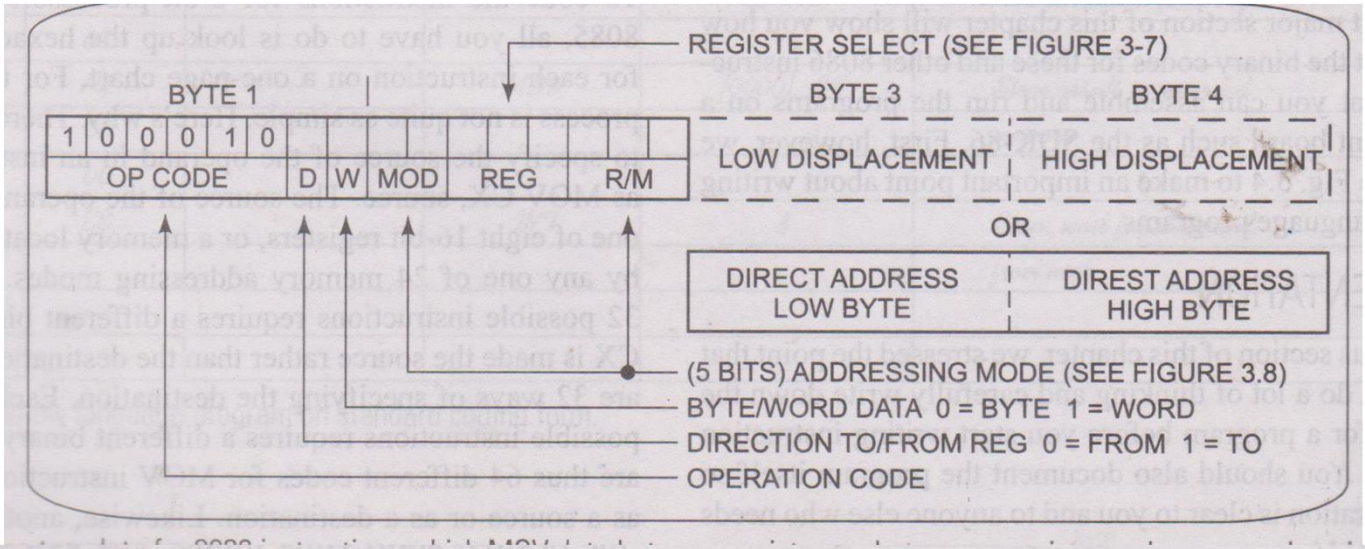


Fig. 2.16 Addition of data segment register and effective address to produce the physical address of the data byte. (a) Diagram, (b) Computation.

Instruction Format



MOV instruction



MOD / R/M	00	01	10	11	
				W = 0	W = 1
000	[BX] + [SI]	[BX] + [SI] + d8	[BX] + [SI] + d16	AL	AX
001	[BXI + [DI]	[BX] + [DI] + d8	[BX] + [DI] + d16	CL	CX
010	[BP] + [SI]	[BP] + [SI] + d8	[BP] + [SI] + d16	DL	DX
011	[BP] + [DI]	[BP] + [DI] + d8	[BP] + [DI] + d16	BL	BX
100	[SI]	[SI] + d8	[SI] +d16	AH	SP
101	[DI]	[DI] + d8	[DI] +d16	CH	BP
110	d16 (direct address)	[BP] + d8	[BP] +d16	DH	SI
111	[BX]	[BX] + d8	[BX] + d16	BH	DI
	MEMORY MODE			REGISTER MODE	

d8 = 8-bit displacement d16 = 16-bit displacement

Instruction Set

Data transfer instructions

- MOV DL, [BX] — copy a byte from memory at [BX] to DL
- PUSH DS — Decrement SP by 2, copy DS to stack
- POP DX — copy a word from ToS to DX, increment SP by 2
- PUSHA — copy all registers to stack
- POPA — copy words from stack to all registers
- XCHG BL, CH — exchange the bytes in BL and CH
- XLAT — translate a byte in AL using a table in memory
- IN AX, 34H — from port 34H to AX — IO port transfer instructions
- OUT 3BH, AL — AL -> port 3BH
- LEA CX, [BX][DI] — load CX with EA = (BX) + (DI) — Address transfer instructions
- LDS BX, [4326H] — copy the contents of memory at displacement 4326H in DS to BL, contents of 4327H to BH. Copy the contents at displacement 4328H, 4329H in DS to DS register
- LES DI, [BX] — copy the contents of memory at offset [BX], [BX+1] in DS to DI register. Copy the contents of memory at [BX+2], [BX+3] to ES register
- LAHF — load AH with low byte of flag register — Flag transfer instructions
- SAHF — copy AH register to low byte of flag register
- PUSHF — copy flag register to ToS
- POPF — copy word at ToS to flag register

Arithmetic Instructions

• ADD AL, 74H	AL <- AL + 74H	Addition instructions
• ADC CL, BL	add contents of BL, carry, and CL	
• INC CX	increment CX by 1	
• AAA	ASCII adjust after addition	
• DAA	decimal adjust after addition	
• SUB AX, 3427H	AX – 3427H	Subtraction instructions
• SBB CH, AL	CH <- CH – AL – carry flag	
• DEC BP	subtract BP by 1	
• NEG BX	replace the word in BX with its 2's complement	
• CMP BH, CL	compare the bytes in CL with BH	
• AAS	ASCII adjust after subtraction	
• DAS	decimal adjust after subtraction	
• MUL BH	AX <- AL * BH	Multiplication instructions
• IMUL BH	AX <- signed byte in AL * signed byte in BH	
• AAM	ASCII adjust after multiplication	
• DIV BL	AL (quotient) <- AX / BL; AH (remainder) <- AX / BL	
• IDIV BL	signed word in AX / signed byte in BL	Division instructions
• AAD	ASCII adjust before division	
• CBW	fill upper byte of word with copies of sign bit of lower byte	
• CWD	fill upper word of double word with sign bit of lower word	

Bit Manipulation Instructions

Logical instructions

- NOT BX — Complement the contents of BX
- AND BX, 00FFH — AND 00FFH with BX
- OR BP, SI — ORed BP with SI
- XOR BP, DI — XORed BP with DI
- TEST AL, BH — AND AL with BH, update parity, sign, and zero flags
- Shift instructions
- SAL BX, 1 — shift word in BX by 1 bit position left, 0 in LSB
- SHR BP, 1 — shift word in BP by 1 bit position right, 0 in MSB
- SAR DI, 1 — shift word in DI by 1 bit position right, new MSB=old MSB
- Rotate instructions
- ROL AX, 1 — rotate word in AX by 1 bit position left, MSB to LSB and carry flag
- ROR BL, 1 — rotate the byte in BL right by 1 position, LSB to MSB and carry flag
- RCL DX, 1 — rotate word in DX by 1 bit left, MSB to carry flag, carry flag to LSB
- RCR BX, 1 — rotate word in BX by 1 bit right, LSB to carry flag, carry flag to MSB

String Instructions

- REP repeat the following instruction until CX=0
- REPE/REPZ repeat the instruction until CX=0 or ZF=1
- REPNE/REPNZ repeat the instruction until CX=0 or ZF=0
- MOVS/MOVSB/MOVSW move byte or word from one string to another
- COMPS/COMPSB/COMPSW compare two string bytes or two string words
- INS/INSB/INSW input string byte or word from port
- OUTS/OUTSB/OUTSW output string byte or word to port
- SCAS/SCASB/SCASW scan a string. Compare a string byte with AL or a string word with AX
- LODS/LODSB/LODSW load string byte into AL or string word into AX
- STOS/STOSB/STOSW store byte from AL or word from AX into string

Execution Transfer Instructions

Conditional transfer instructions

- JA/JNBE jump if above/jump if not below or equal
- JAE/JNB jump if above or equal/jump if not below
- JB/JNAE jump if below/jump if not above or equal
- JBE/JNA jump if below or equal/jump if not above
- JC jump on carry
- JE/JZ jump if equal/jump on zero
- JG/JNLE jump if greater/jump if not less than or equal
- JGE/JNL jump if greater than or equal/jump if not less than
- JL/JNGE jump if less than/jump if not greater than or equal
- JLE/JNG jump if less than or equal/jump if not greater than
- JNC jump on no carry
- JNE/JNZ jump if not equal/jump if not zero
- JNO jump if no overflow
- JNP/JPO jump if not parity/jump if parity odd
- JNS jump if not sign
- JO jump if overflow
- JP/JPE jump if parity/jump on even parity
- JS jump if sign

Cont…

Unconditional transfer instructions

- JMP label — jump to the specified address
- CALL procedure
- RET

Iteration control instructions

- LOOP label — jump to specified label if CX≠0 after autodecrement
- LOOPE/LOOPZ — loop while CX≠0 and ZF=1
- LOOPNE/LOOPNZ — loop while CX≠0 and ZF=0
- JCXZ — jump if CX=0

Interrupt instructions

- INT 0 to 255 — interrupt program execution, and call ISR
- INTO — interrupt program execution if OF=1
- IRET — return from ISR

HLL interface instructions

- ENTER — enter procedure
- LEAVE — leave procedure
- BOUND — check if effective address within specified array bounds

Processor Control Instructions

Flag set/clear instructions

- STC set CF
- CLC clear CF
- CMC complement CF
- STD set DF (decrement string pointer)
- CLD clear direction flag (DF)
- STI set interrupt enable flag (enable INTR input)
- CLI clear interrupt enable flag (disable INTR input)

External h/w synchronization instructions

- HLT halt
- WAIT wait until signal on the TEST pin is low
- ESC escape to external coprocessor such as 8087 or 8089
- LOCK prevents another processor from taking the bus while adjacent instruction executes
- No operation instruction NOP