5th Semester ETC ENGG. (COURSE CODE- BEC 1502)

Microprocessors (3-1-0)

Module-1 (8 Hours)

Introduction to Microprocessor: Intel 8085 Microprocessor: Architecture, pins & signals, Register organization, Timing &control unit, Instruction Timing & Execution, Instruction set of 8085, Memory & I/O Addressing, Assembly language programming using 8085 instructions set.

Module-2 (10 Hours)

Memory Interfacing: Interfacing EPROM &RAM Memories: 2764 and 6264. Stack &Subroutines: Stack, Subroutines, Restart, Conditional Call and Return Instructions, Advanced Subroutine Concepts. 8085 Interrupts: 8085 Interrupts, Vectored Interrupts, Restart as Software Instructions.

Module-3 (10 Hours)

Microprocessor based system Developments Aids: Programmable peripheral Interface: 8255, Programmable DMA Controller: 8257, Programmable Interrupt Controller: 8259, Programmable Interval Timer: 8253.

Module-4 (12 Hours)

Intel 8086 (16 bit): Introduction, pins & signal description, Architecture, Bus timing, minimum mode 8086, and maximum mode 8086, Multiprocessor: parallel processing, Instruction sets of 8086: Instruction formats, Addressing modes, Instruction set: data transfer instruction,arithmetic and logic instruction, program control instructions, Assembly language programming with 8086, iterative procedure, recursive procedure, parameter passing.

Intel 80386 and 80486: Architecture, Register Organization, Protected mode, Paging, Virtual mode.

Salient features of Pentium Processor.

Text books:

- Microprocessor Architecture, programming and applications with the 8085 by R.S. Gaonkar, Penram International, India.
- Microprocessors and Microcontrollers by N.Senthil Kumar, M.Saravanan, and S.Jeevananthan, Oxford University Press.

MODULE-1

Introduction to Microprocessor: Microprocessor is an electronic chip that functions as the central processing unit (CPU) of a computer. The microprocessor based systems with limited resources are called as microcomputers. Now-a-days microprocessors are found in almost all electronic machines and appliances in its different form. Some common devices using microprocessors are computer, printers, automobiles, washing machines, microwave ovens, mobile phones, fax machines, Xerox machines and advanced instruments like radar, satellites, flights etc.

Almost all microprocessors use the basic concept of “stored program execution”. By this concept, programs are stored sequentially in memory locations. The microprocessor will fetch the instructions one after the other and execute them in its arithmetic and logic unit. So it is necessary for the user to know about the internal resources and features of the microprocessor. The programmers must also understand the instructions that a microprocessor can support. Every microprocessor will have its own associated set of instructions and this list is given by all the microprocessor manufacturers. Programs are written using mnemonics called the assembly level language and then they are converted into binary machine level language. This conversion can be done manually or using an application called assembler.

In general, the programs are written by the user for a microprocessor to work with real world data. These data are available in many forms and are from many sources. A microprocessor based system need a set of memory units, set of interfacing circuits for inputs and a set of interfacing circuits for outputs. All circuits put together along with microprocessor are called as microcomputer system. The physical components of the microcomputer system are in general called as hardware. The program which makes this hardware useful is called as software.

Origin of Microprocessor

The breakthrough in transistor technology led to the introduction of minicomputers of the 1960s and the personal computer revolution of the 1970s.Microprocessors evolution is categorized into five generations i.e.first, second, third, fourth, and fifth generations.

First Generation (1971-73)

The microprocessors that were introduced in 1971 to 1972 were referred to as the first generation systems. Intel Corporation introduced 4-bit 4004 at 108 kHz, the first microprocessor in 1971, co-developed by Busicom, a Japanese manufacturer of calculators. In 1972, Intel made the 8-bit 8008 and 8080 microprocessors.

Second Generation (1974-78)

The second generation marked the beginning of very efficient 8-bit microprocessors. Some of the popular processors are Motorola’s 6800 and 6809 and Intel’s 8085, Zilog’s Z80. The distinction between the first and second generation devices is primarily the use of newer semiconductor technology to fabricate the chips. They were manufactured using NMOS technology.

Third Generation (1979-80)

Introduced in 1978, dominated by Intel's 8086 and the Zilog Z8000, which were 16-bit processors, have 16-bit arithmetic and pipelined instruction processing.

Fourth Generation (1981-95)

Intel introduced 32 bit processor, 80386 and Motorola 68020/68030. Fabricated using low-power version of the HMOS technology called HCMOS.

Fifth Generation (1995 till date)

Chips carry on-chip functionalities and improvements in the speed of memory and I/O devices. Design surpassed 10 million transistors per chip. Introduction of 64-bit processors.Intel leads the show with Pentium, Celeron and dual and quad core processors working with up to 3.5GHz speed.

Intel 8085 Microprocessor

Introduction

8085 is an eight bit microprocessor of Intel Corporation, usually called as a general purpose 8-bit processor. It is upward compatible with microprocessor 8080, which is the earlier product of Intel. Several faster versions of 8085 microprocessor are 8085AH, 8085AH-1, and 8085AH-2.

A microprocessor system consists of three functional blocks: central processing unit (CPU), input and output units, memory units as shown in figure1.1. The central processing unit contains several registers, arithmetic logic unit (ALU) and control unit.

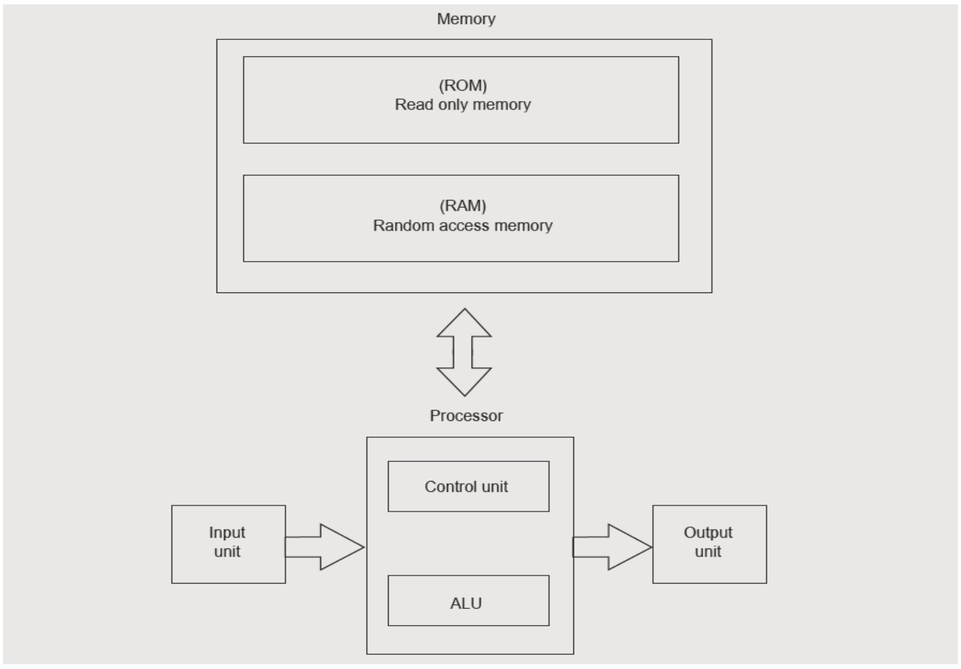


Figure1.1: Microprocessor System

Microprocessor is an integrated chip that functions as the central processing unit of a computer. The microprocessor basically performs

- Memory Read: Accept data (instruction) from memory.
- Memory Write: Send data to memory.
- I/O Read: Accept data from input device.
- I/O Write: Send data to output device.
- Controls timing of instruction flow.

Memory includes ROM (read only memory) and RAM (random access memory or read write memory).The memory

- Stores instructions and data.
- Provides the instructions and data to processor.
- Stores results.

The input devices enter instructions and data to processor. The input device includes Key-Board (Hexadecimal / ASCII), Switches, and Analog-to-Digital (A/D) convertor.

The output devices accept data from processor. This includes LED (Light Emitting Diode) display, LCD (Liquid Crystal Diode) display, CRT (Cathode Ray Tube) Screen.

Architecture

The internal block diagram of 8085 is shown in figure 1.2. It is a 40 pin IC package and uses +5V for power. It can run at a maximum frequency of 3MHz. It is a 8-bit processor which has a data bus of 8-bits wide. It has addressing capability of 16-bit.That is it can address 2^{16} = 64K Bytes of memory (1Kbyte =1024 byte).

The architecture of 8085 processor consists of five functional units: Arithmetic and logic Unit, General purpose registers, Special Purpose Registers, Instruction register/decoder and Timing and control unit.

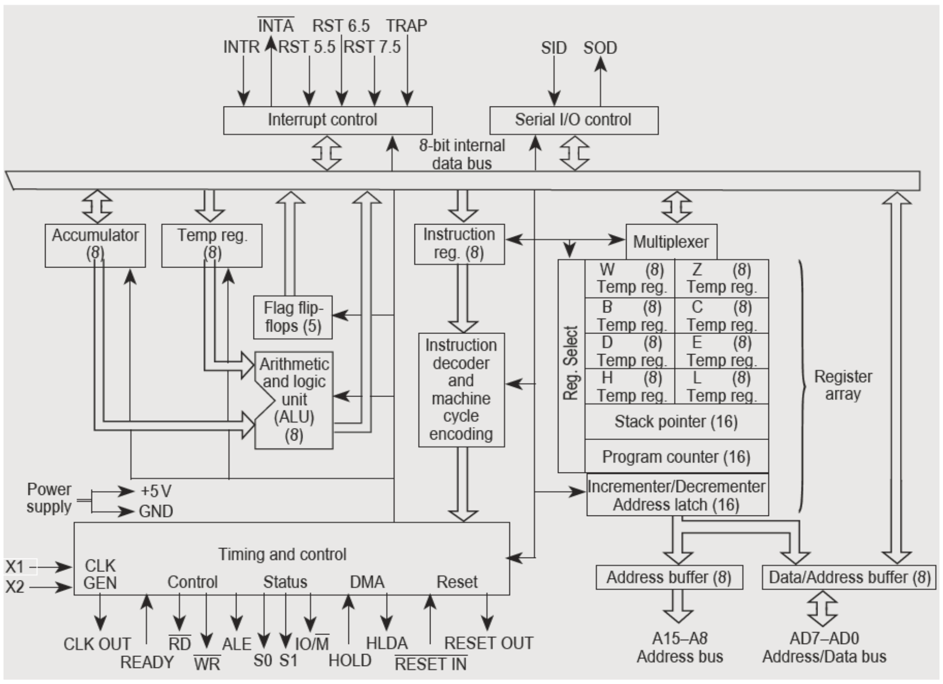
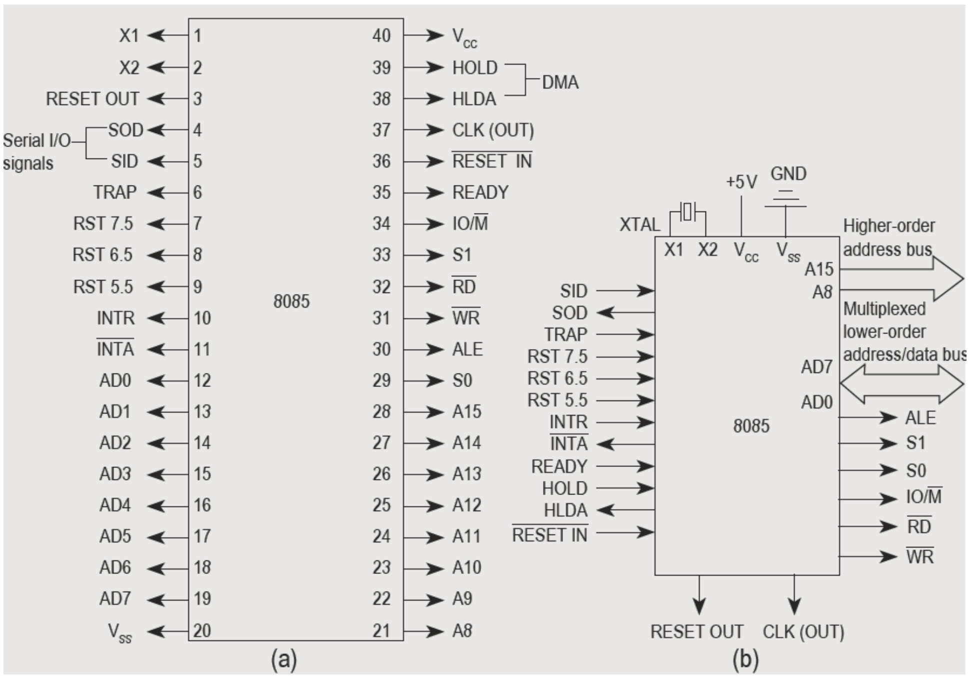


Figure 1.2: Architecture of 8085



Pin Diagram of 8085

Register Organization

The register organization of 8085 is shown in figure 1.3.

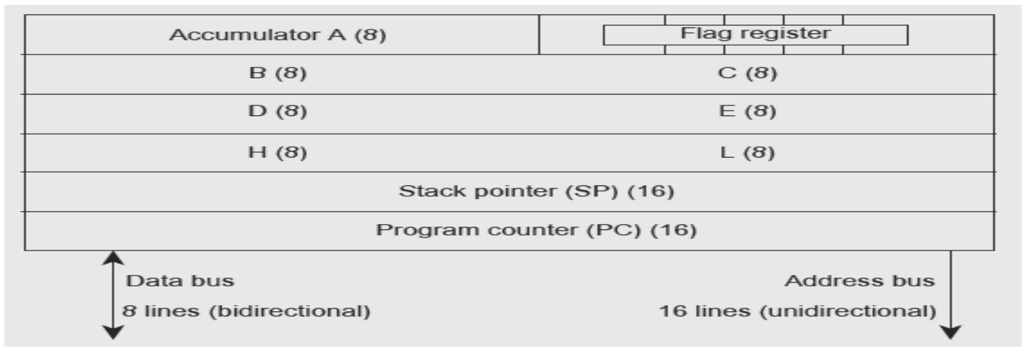


Figure1.3: Register Organization of 8085

Registers: The 8085 has six general purpose registers to store 8-bit data. They are identified as B, C, D, E, H and L. They can be combined as register pairs: B-C, D-E, and H-L to store and perform 16-bit operations.

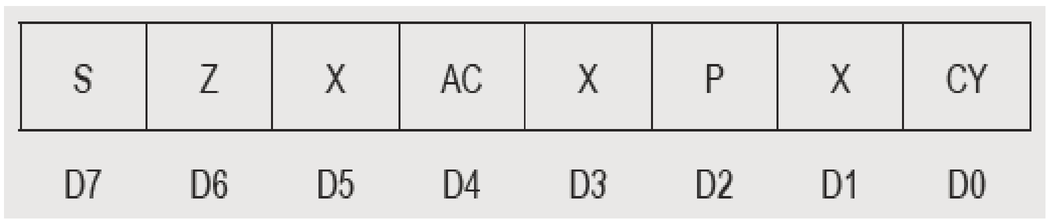
The 8085 special purpose registers are the Accumulator, Flag register, Program Counter (PC) and Stack Pointer (SP).

Accumulator: This is an 8 bit register that is part of arithmetic and logical unit (ALU).This is used to store 8-bit data and perform arithmetic and logical operations. The result of an operation is stored in the accumulator.

Program Counter (PC): This is a 16-bit register, which always points to the address of next instruction to be fetched.

Stack Pointer (SP): This is a 16-bit register that points to a memory location in R/W memory, called as stack.

Flags: The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of result in the accumulator. They are called as Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags .The flag register format of 8085 is shown in figure 1.4.



S	Z	X	AC	X	P	X	CY
D7	D6	D5	D4	D3	D2	D1	D0

Figure1.4: Flag Register Format of 8085

- Z-Zero: The zero flag is set if the result is zero, otherwise it is reset.
- CY-Carry: If an arithmetic operation results in a carry, the carry flag is set otherwise it is reset.
- S-Sign: The sign flag is set if bit D7 of the result is one, otherwise it is reset.
- AC- Auxiliary Carry: In a BCD arithmetic operation, when a carry results from digit D3 and passes on to digit D4, the AC flag is set.
- P-Parity: If the result has an even number of 1's, the parity flag is set. For odd number of 1's, the flag is reset.

Two additional 8-bit temporary registers W and Z are included in the register array. They are not programmable.

The ALU (arithmetic and logic unit)

The arithmetic logic unit of 8085 performs addition, subtraction, increment, decrement and comparison arithmetic operation and logical operations such as AND, OR, exclusive-OR, complement. The ALU includes the accumulator, the temporary register, the arithmetic logic circuits, and five flip-flops. The temporary register is used to hold data during arithmetic logic operation. The result is stored in accumulator, and flags are set or reset according to result of operation.

The 8085 is also called accumulator oriented processor as one of the data in the ALU operation is stored in the accumulator. The other data may be in memory or in register.

Timing and Control unit

The timing and control unit synchronizes all microprocessor operations with clock and generates control and status signals (RD, WR) for communication between microprocessor and peripherals.

Instruction Register and Decoder

The Instruction register/decoder is an 8-bit register that is part of the ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction. The instruction register is not programmable.

Pins and Signals

Intel 8085 has 40 pins, operates at 3MHz clock and requires +5V for power supply. The pin diagram of 8085 is shown in figure 1.6.The signals can be classified into six groups i.e. Address Bus, Data Bus, Control and Status Signals, Power supply and System Clock, Externally Initiated Signals, and Serial I/O signals.

Address and Data Buses

The 8085 has 16 bit (A15-A0) address lines which are unidirectional and 8-bit (D7-D0) data lines which are bidirectional.

A8 - A15 (Output-3 state: higher order address bus): The most significant 8 bits of the memory address or the 8 bits of the I/0 addresses, tri-stated during Hold and Halt modes.

AD0-AD7 (Input/output- 3 state: multiplexed address/data bus): Lower 8 bits of the memory address (or I/0 address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. Tri- stated during Hold and Halt modes.

Control and Status Signals

ALE (Output) Address Latch Enable:

This output signal indicates the availability of the valid address on the address/data lines. It occurs every time during the first clock cycle of the 8085 machine cycle operation. It is used to latch the low order address from multiplexed bus and generate a separate address (A7-A0).

RD (Output- 3 state) READ:

This is a Read control signal (active low).This indicates that the selected memory or I/O device is to be read and data are available on the data bus.

WR (Output- 3state) WRITE:

This is a Write control signal (active low).This indicates that the data on the data bus is to be written into the selected memory or I/O location.

IO/M (Output: 3 state):

This is a status signal used to differentiate between I/O and memory operation. When it is high, it indicates an I/O operation. When it is low, it indicates a memory operation.

SO, S1 (Output): These are status signals and identify various operations (Figure 1.5).

S1	S0	States
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

Figure1.5: 8085 Status Signals

Externally initiated signals

HOLD (Input):

It is an active high signal used in the direct transfer of data between a peripheral device and memory locations. This type of data transfer is called as direct memory access (DMA).During this transfer, the microprocessor loses control over the address and data buses and these buses are tri-stated.

Logic 1 on the Hold pin indicates that another controller, generally the DMA controller, is requesting the use of the address and data buses.

HLDA (Output): Hold Acknowledge

This is an active high signal, and acknowledges HOLD request.

INTR (Input): Interrupt Request

The INTR is used as a general purpose interrupt.

INTA (Output): Interrupt Acknowledge

This is an active low signal and is used to acknowledge an interrupt.

RST 7.5, RST 6.5, and RST 5.5: Restart Interrupts (input):

These three are hardware vectored interrupt signals.

TRAP (Input) :

This is a non-maskable interrupt and has highest priority.

RESET IN (Input):

When this goes low, the Program Counter is set to zero (0000H),the buses are tri-stated and reset the 8085.

RESET OUT (Output):

This indicates that 8085 is being reset. This can be used to reset other devices.

Power supply and Clock Frequency

X1, X2 (Input):

A Crystal or R-C or L-C network is connected to these two pins. The crystal frequency is internally divided by two to give the operating system frequency.So, to run the microprocessor at 3 MHz, a clock running at 6 MHz should be connected to the X1and X2 pins.

CLK (Output): Clock Output:

This output clock pin is used to provide the clock signal for other devices.

Power supplies:Vcc : +5 V supply; Vss : Ground Reference

Serial I/O Signals

SID and SOD implement serial data transmission.

SID (Input):

Serial input data line. The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

SOD (output):

Serial output data line. The accumulator bit 7 output on SOD line by the SIM instruction.

Instruction Timing and Execution:

An instruction is a command to a microprocessor to perform a specific task on data. Each instruction consists of two parts: one is operation code (OPCODE), and second one is Operand.

Instruction Cycle:

It is defined as the time taken by the processor to complete fetch and execution of an instruction. Instruction cycle consists of both fetch cycle and execute cycle. Instruction cycle consists of 1 to 6 machine cycles.

Fetch Cycle:

It is defined as the time taken by the processor to fetch an operation code from memory.

Execution Cycle:

It is defined as the time taken by the processor to decode and execute an instruction.

Machine Cycle:

The time required to complete one operation of accessing memory, I/O. or acknowledging an external request. The machine cycle consists of 3 to 6 T-states.

T state:

The T state is equal to one clock period. The T state is defined as one sub division of an operation performed in one clock period.

8085 Machine Cycles

The 8085 microprocessor has 5 (Five) basic machine cycles. They are Opcode Fetch Cycle (4T), Memory Read Cycle (3 T), Memory Write Cycle (3 T), I/O Read Cycle (3 T), and I/O Write Cycle (3 T).In Opcode Fetch Cycle processor get a Opcode from memory. The processor read data from memory in Memory Read Cycle. In Memory Write Cycle, the data is stored in memory. The processors get data from input device in I/O Read Cycle and send data to output device in I/O Write Cycle.

Timing Diagram:

The timing diagram of an instruction is obtained by drawing the binary levels on the various signals of 8085. It is drawn with respect to the clock input of the microprocessor. It explains the execution of the instruction with the basic machine cycles of that instruction, one by one in the order of execution.

Opcode Fetch Cycle (4T):

It is defined as the time taken by the processor to fetch an operation code from memory. The program counter places 16-bit memory address on the address bus. The 8085 sends memory read control signal to enable memory chip. The operation code from memory location is placed on the data bus. The operation code from data bus is stored in the instruction register to decode and execute.

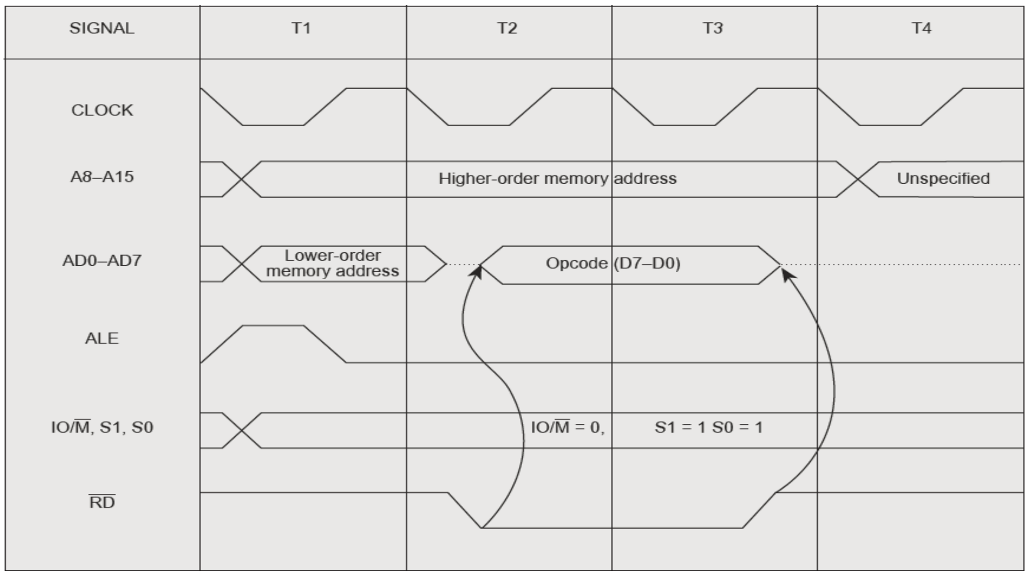

SIGNAL
T1
T2
T3
T4
CLOCK
A8–A15
Higher-order memory address
Unspecified
AD0–AD7
Lower-order memory address
Opcode (D7–D0)
ALE
IO/$\overline{M}$, S1, S0
IO/$\overline{M}$ = 0, S1 = 1 S0 = 1
$\overline{RD}$

Figure1.6: Timing Diagram of Opcode Fetch Cycle

- At T1, the high order 8 address bits are placed on the address lines A8 – A15 and the low order bits are placed on AD7–AD0. The ALE signal goes high to indicate that AD0 – AD8 are carrying an address. At exactly the same time, the IO/M signal goes low to indicate a memory access operation.
- At the beginning of the T2 cycle, the low order 8 address bits are removed from AD7– AD0 and the processor sends the Read (RD) signal to the memory. The RD signal remains low (active) for two clock periods to allow for reading slow devices. During T2, memory places the operation code byte from the memory location on the lines AD7– AD0.

- During T3 the RD signal is Disabled (goes high). This turns off the output Tri-state buffers in the memory. That makes the AD7– AD0 lines go to high impedance state.
- During T4, the opcode is decoded by the processor and necessary action or control is initiated for the execution of the instruction fetched.

Memory Read Cycle (3 T) :

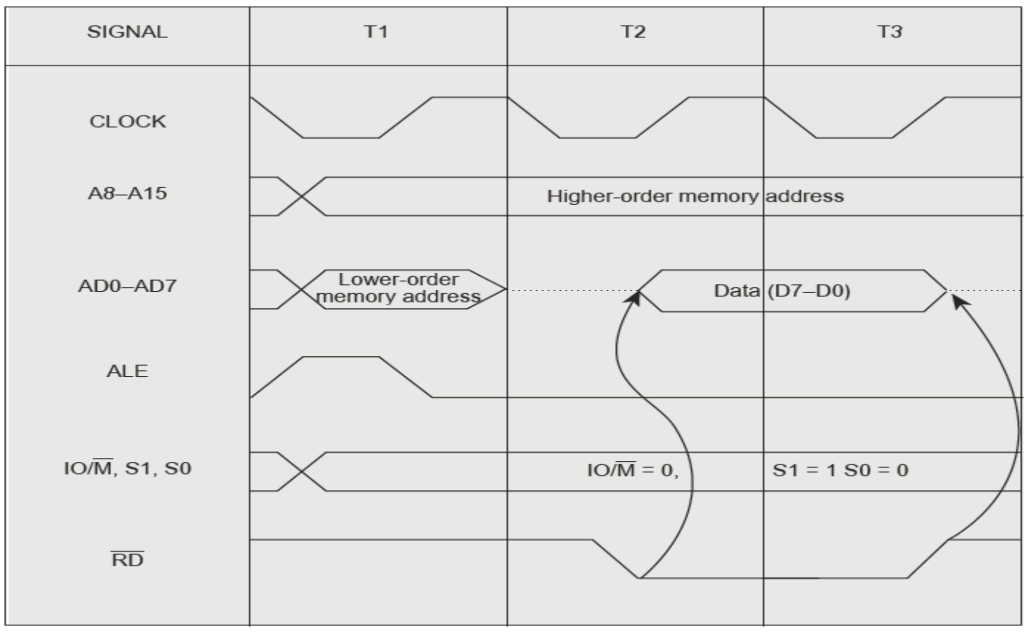


Figure1.7: Timing Diagram of Memory Read Cycle

The memory read cycle is similar as the opcode fetch cycle except: It only has 3 T-states. The S0 signal is 0. S1 is set to 1.In memory read cycle processor read a data byte from memory.

Instruction Set of 8085

An instruction is a command to a microprocessor to perform a specific task on data. The instruction set is the total number of instructions supported by processor.

Each instruction consists of two parts: one is operation code (OPCODE), and second one is Operand. The operation code specifies the type of operation to be performed. The operand is the data to be operated on. The operand includes 8-bit/16-bit data, an internal register, memory location, and 8-bit/16-bit address.

In the design of 8085, all the instructions, and registers are identified with a binary code.

Code	Registers
000	B
001	C
010	D
011	E
100	H
101	L
110	A
111	M

Code	Register Pair
00	B-C
01	D-E
10	H-L
11	AF or SP

Instruction Word Size

The word size of 8085 instruction can be

1-byte instruction: ADD B

The opcode and operand are in the same byte.

2-byte instruction: MVI A, 05H.

The first byte specifies opcode and second byte specify the operand.

3-byte instruction: LDA 2000H.

The first byte specifies opcode and the following two bytes specify 16-bit address.

Addressing Mode

The various techniques to specify data for instruction is called addressing mode.

Direct addressing: The address of the data is specified in the instruction itself. STA 2000H, IN 07H.

Register addressing: The operands are in general purpose register. MOV A, B , ADD B.

Register indirect addressing: The address of the operand is specified by register pair indirectly.

MOV A, M   , ADD M.

Immediate addressing: The operand is specified within the instruction itself.MVI A,03H, ADI 12H.

Implicit addressing: The operand is in the accumulator. CMA, RAL, RAR.

The entire group of instructions is called the instruction set, and this determines the functionalities the microprocessor can perform.

The 8085 Instructions can be classified into five functional categories: Data Transfer (copy) operations, Arithmetic operations, Logical operations, Branching operations, and Machine-control operations.

Data transfer instructions

This group of instructions copy data from a location called a source to another location called a destination without modifying the contents of the source. The various type of data transfer are:

Between registers: MOV B, D

Data byte to register/memory: MVI A, 45H,   MVI M, 34H.

Between memory and register: LDA 2100H, STA 2400H.

Between I/O device and accumulator: IN 16H, OUT 12H.

Arithmetic instruction

The arithmetic instructions are addition, subtraction, increment, and decrement.

Addition

Any 8-bit number, or the content of a register, or the content of a memory location can be added to the content of accumulator. The sum is stored in the accumulator.

The DAD instruction adds 16-bit numbers in register pairs.

Subtraction

Any 8-bit number, or the content of a register, or the content of a memory location can be subtracted from the content of accumulator. The result is stored in the accumulator. The subtraction is performed in 2's complement form.

Mnemonics	Tasks performed on execution	Addressing mode	Length of the Instruction	Example
ADI 8-bit	Add immediate to accumulator	Immediate	Two bytes	ADI 30H
ACI 8-bit	Add immediate to accumulator with carry	Immediate	Two bytes	ACI 4FH
SUI 8-bit	Subtract immediate from accumulator	Immediate	Two bytes	SUI 2AH
SBI 8-bit	Subtract immediate from accumulator with borrow	Immediate	Two bytes	SBI 5CH

ADD R	Add register content to accumulator	Direct	One byte	ADD C
ADC R	Add register content to accumulator with carry	Direct	One byte	ADC E
SUB R	Subtract register content from accumulator	Direct	One byte	SUB B
SBB R	Subtract register content and borrow from accumulator	Direct	One byte	SBB C
DAD Rp	Add register pair to H and L registers	Direct	One byte	DAD B
INR R	Increment register by 1	Direct	One byte	INR B
INX Rp	Increment register pair by 1	Direct	One byte	INX B
DCR R	Decrement register by 1	Direct	One byte	DCR E
DCX Rp	Decrement register pair by 1	Direct	One byte	DCX D
ADD M	Add memory content pointed by HL register pair to accumulator	Indirect	One byte	ADD C
ADC M	Add memory content pointed by HL register pair to accumulator	Indirect	One byte	ADC E
SUB M	Subtract memory content pointed by HL register pair from accumulator	Indirect	One byte	SUB B
SBB M	Subtract memory content pointed by HL pair and borrow from accumulator	Indirect	One byte	SBB C
INR M	Increment a memory content pointed by HL register pair once	Indirect	One byte	INR M
DCR M	Increment a memory content pointed by HL register pair once	Indirect	One byte	INR M
DAA	Decimal adjust accumulator	Implicit	One byte	DAA

Figure1.8: Arithmetic Instructions of 8085

Logical instruction

These instructions perform logical operation with content of accumulator. These instructions are AND, OR, Ex-OR, rotate, compare, and complement.

Logical AND

Any 8-bit number, or the content of a register, or the content of a memory location can be logically ANDed with the content of accumulator. The result is stored in the accumulator.

Mnemonics	Tasks performed on execution	Addressing mode	Length of the Instruction	Example
ANI 8-bit	Logical AND immediate with accumulator	Immediate	Two bytes	ANI 0FH
XRI 8-bit	Exclusive OR immediate with accumulator	Immediate	Two bytes	XRI 01H
ORI 8-bit	Logical OR immediate with accumulator	Immediate	Two bytes	ORI 80H
ANA R	Logical AND register or memory with accumulator	Direct	One byte	ANA C
XRA R	Exclusive OR register or memory with accumulator	Direct	One byte	XRA D
ORA R	Logical OR register or memory with accumulator	Direct	One byte	ORA E
ANA M	Logical AND memory pointed by HL register pair with accumulator	Indirect	One byte	ANA M
XRA M	Logical XOR memory pointed by HL register pair with accumulator	Indirect	One byte	XRA M
ORA M	Logical OR memory pointed by HL register pair with accumulator	Indirect	One byte	ORA M
RLC	Rotate accumulator left	Implicit	One byte	RLC
RRC	Rotate accumulator right	Implicit	One byte	RRC
RAL	Rotate accumulator left through carry	Implicit	One byte	RAL
RAR	Rotate accumulator right through carry	Implicit	One byte	RAR
CPI 8-bit	Compare immediate with accumulator	Immediate	Two bytes	CPI FFH
CMP R	Compare register or memory with accumulator	Direct	One byte	CMP B
CMP M	Compare memory pointed by HL register pair with accumulator	Indirect	One byte	CMP M
CMA	Complement accumulator	Implicit	One byte	CMA
CMC	Complement carry	Implicit	One byte	CMC
STC	Set carry	Implicit	One byte	STC

Figure 1.9: Logical Instructions of 8085

Branching Instructions

This instruction alters the sequence of program execution either conditionally or unconditionally. These instructions are jump, call, return, and restart.

Mnemonics	Tasks performed on execution	Addressing mode	Length of the Instruction	Example
JMP 16-bit	Jump unconditionally	Immediate	Three bytes	JMP 9500
JC 16-bit	Jump if carry is set	Immediate	Three bytes	JC 9500
JNC 16-bit	Jump on no carry	Immediate	Three bytes	JNC 9500
JP 16-bit	Jump on positive	Immediate	Three bytes	JP 9500
JM 16-bit	Jump on minus	Immediate	Three bytes	JM 9500
JZ 16-bit	Jump on zero	Immediate	Three bytes	JZ 9500
JNZ16-bit	Jump on no zero	Immediate	Three bytes	JNZ 9500
JPE 16-bit	Jump on parity even	Immediate	Three bytes	JPE 9500
JPO 16-bit	Jump on parity odd	Immediate	Three bytes	JPO 9500
CALL 16-bit	Call unconditionally	Immediate	Three bytes	CALL 9500
CC 16-bit	Call on carry	Immediate	Three bytes	CC 9500
CNC 16-bit	Call on no carry	Immediate	Three bytes	CNC 9500
CP 16-bit	Call on positive	Immediate	Three bytes	CP 9500
CM 16-bit	Call on minus	Immediate	Three bytes	CM 9500
CZ 16-bit	Call on zero	Immediate	Three bytes	CZ 9500
CNZ 16-bit	Call on no zero	Immediate	Three bytes	CNZ 9500
CPE 16-bit	Call on parity even	Immediate	Three bytes	CPE 9500
CPO 16-bit	Call on parity odd	Immediate	Three bytes	CPO 9500
RET	Return unconditionally	Implicit	One byte	RET
RC	Return on carry	Implicit	One byte	RC
RNC	Return on no carry	Implicit	One byte	RNC
RP	Return on positive	Implicit	One byte	RP

RM	Return on minus	Implicit	One byte	RM
RZ	Return on zero	Implicit	One byte	RZ
RNZ	Return on no zero	Implicit	One byte	RNZ
RPE	Return on parity even	Implicit	One byte	RPE
RPO	Return on parity odd	Implicit	One byte	RPO
PCHL	Copy HL contents to the program	Implicit	One byte	PCHL
RST 0/1/2/3/4/5/6/7	Restart	Implicit	One byte	RST 5

Figure1.10: Branch Instructions of 8085

Machine control operation

These instruction are halt, interrupt, and do nothing etc.

Mnemonics	**Tasks performed on execution**	**Addressing mode**	**Length of the Instruction**
NOP	No operation	Implicit	One byte
HLT	Halt the microprocessor execution	Implicit	One byte
DI	Disable interrupts	Implicit	One byte
EI	Enable interrupts	Implicit	One byte
RIM	Read interrupt mask	Implicit	One byte
SIM	Set interrupt mask	Implicit	One byte

Figure1.11: Machine Instructions of 8085

Assembly Language Programming using 8085 instruction set

Statement: Add the content of memory location 4000H with content of memory location 4001H and place the result in memory location 4002H.

(4000H) = 14H

(4001H) = 89H

Result = 14H + 89H = 9DH

Source program

LXI H 4000H : HL points 4000H

MOV A, M : Get first operand

INX H : HL points 4001H

ADD M : Add second operand

INX H : HL points 4002H

MOV M, A : Store result at 4002H

HLT : Terminate program execution

Statement: Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

Program - 4: Subtract two 8-bit numbers

Sample problem:

(4000H) = 51H

(4001H) = 19H

Result = 51H - 19H = 38H

LXI H, 4000H : HL points 4000H

MOV A, M : Get first operand

INX H : HL points 4001H

SUB M : Subtract second operand

INX H : HL points 4002H

MOV M, A : Store result at 4002H.

HLT : Terminate program execution

MODULE-2

Memory interfacing

Introduction: RAM, ROM, EPROM

The programs and data which are executed by the microprocessor have to be stored in ROM/EPROM and RAM which are basically semiconductor memory chips.

The programs and data which are stored in ROM/EPROM are not erased even-though the power supply to the ROM/EPROM chip is removed. Hence the ROM/EPROM are called non-volatile memory and we can use them to store permanent programs such as monitor program and data such as look up table, which are needed in microprocessor based systems. The difference between ROM and EPROM is that ROM chip is programmable only one time whereas an EPROM chip can be programmed many times.

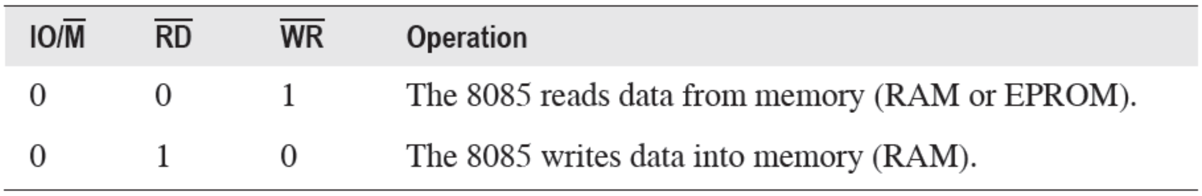
The programs and data which are stored in RAM are erased when the power supply to the RAM chip is removed and hence RAM is called volatile memory. The program written during the learning of assembly language programming and data entered while testing the above programs are stored in RAM.

Interfacing memory chips with 8085

8085 has 16 address lines namely (A15 to A0), a maximum of 64Kbytes (=2^{16}) memory can be interfaced with 8085. The 64 KB memory address space of 8085 has the value from 0000H to FFFFH when represented in hexadecimal form.

The 8085 access memory to read the instructions and data stored in memory and also to store the result in to memory.

The 8085 initiates a set of signals and when it wants to read from and write into memory and the memory chip has certain signals such as chip Enable or Chip Select, or Output Enable or Read and or Write Enable or Write.



IO/$\overline{M}$	$\overline{RD}$	$\overline{WR}$	Operation
0	0	1	The 8085 reads data from memory (RAM or EPROM).
0	1	0	The 8085 writes data into memory (RAM).

Figure2.1: Generation of control signal for memory

Interfacing 2764 EPROM chip with 8085

There are 13 address lines (Since 2^{13} =8K) namely A12 to A0 present in IC 2764 (Fig. 2.1) where A0 is the least significant bit of the address and A12 is the most significant bit of the address.

In order to read the content of a memory location in EPROM chip, the following steps are done

- The address of the memory location from where data has to be read is placed in the address lines of EPROM.
- CE signal is made logic low (i.e. 0)
- OE signal is made logic low(i.e. 0)
- Now the data in the selected memory location will be available in the data lines (D7-D0) of EPROM.

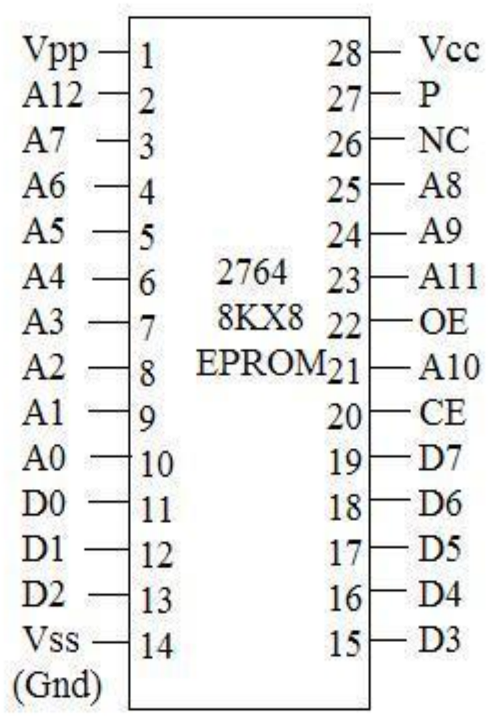


Figure 2.2: Pin diagram of IC 2764

Whenever many number of same capacity memory chips have to be interfaced with 8085, decoder IC with active low outputs such as 74LS138 is very useful.

By using a single 74LS138 IC, maximum of eight memory chips (RAM and EPROM) can be interfaced with 8085.

Partial address decoding:

If the entire address bus of 8085 (A15 to A0) is used to interface memory chips with 8085 than this technique is called absolute address decoding. Using the absolute address decoding method, maximum of 64 Kbytes of memory can be interfaced with 8085.

There is another method known as partial address decoding which is used when the amount of memory needed in an 8085 based system is less than 64 Kbytes such as 8K or 16K or 32K.

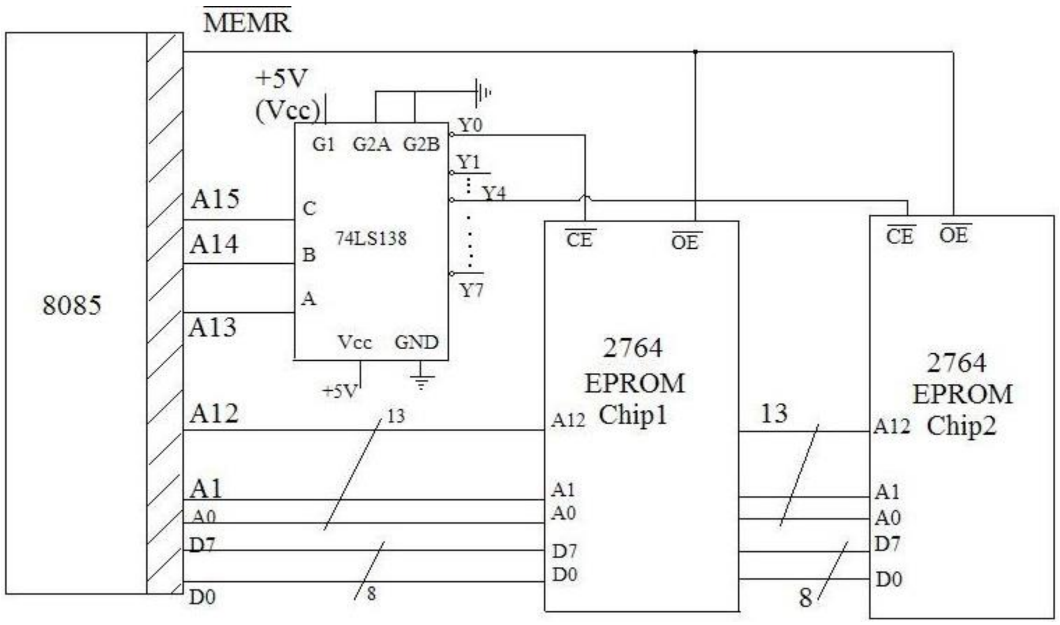


Figure2.3: Interfacing EPROM chips using 74LS138 Decoder

Interfacing 6264 RAM chip with 8085

The interfacing of RAM chip with 8085 is same as that of EPROM chip except that one more signals namely write of 8085 is used.6264 RAM has 13 address lines.

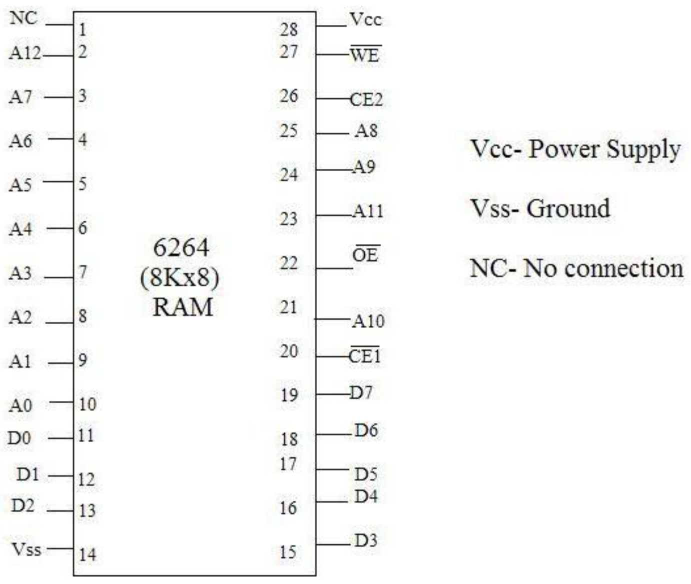


Figure2.4: Pin diagram of IC 6264 RAM

Stack and Subroutines

The stack is an area of R/W memory identified by the programmer for temporary storage of information.

- The stack is a Last in First out (LIFO) structure.
- The stack normally grows backwards into memory.

In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.

Given that the stack grows backwards into memory, it is customary to place the bottom of the stack at the end of memory to keep it as far away from user programs as possible.

In the 8085, the stack is defined by setting the SP (Stack Pointer) register.

```
LXI SP, FFFFH
```

This sets the Stack Pointer to location FFFFH (end of memory for the 8085).

The Size of the stack is limited only by the available memory

Saving Information on the Stack

Information is saved on the stack by PUSH ing it on. It is retrieved from the stack by POPing it off.

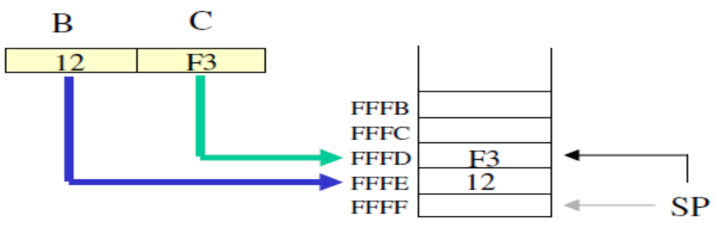
The 8085 provides two instructions: PUSH and POP for storing information on the stack and retrieving it back. Both PUSH and POP work with register pairs only.

The PUSH Instruction

PUSH B (1 Byte Instruction).

Decrement SP, Copy the contents of register B to the memory location pointed to by SP.

Decrement SP, Copy the contents of register C to the memory location pointed to by SP

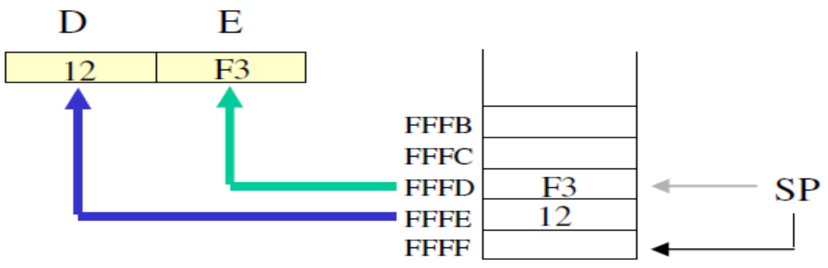


The POP Instruction

POP D (1 Byte Instruction).

Copy the contents of the memory location pointed to by the SP to register E, Increment SP.

Copy the contents of the memory location pointed to by the SP to register D, Increment SP.



Operation of the Stack

During pushing, the stack operates in a “decrement then store” style.

The stack pointer is decremented first, and then the information is placed on the stack.

During popping, the stack operates in a “use then increment” style.

The information is retrieved from the top of the, the stack and then the pointer is incremented.

The SP pointer always points to “the top of the stack”.

LIFO

The order of PUSHs and POPs must be opposite of each other in order to retrieve information back into its original location.

PUSH B

PUSH D

...

POP D

POP B

Reversing the order of the POP instructions will result in the exchange of the contents of BC and DE.

The PSW Register Pair

The 8085 recognizes one additional register pair called the PSW (Program Status Word).

This register pair is made up of the Accumulator and the Flags register.

It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack.

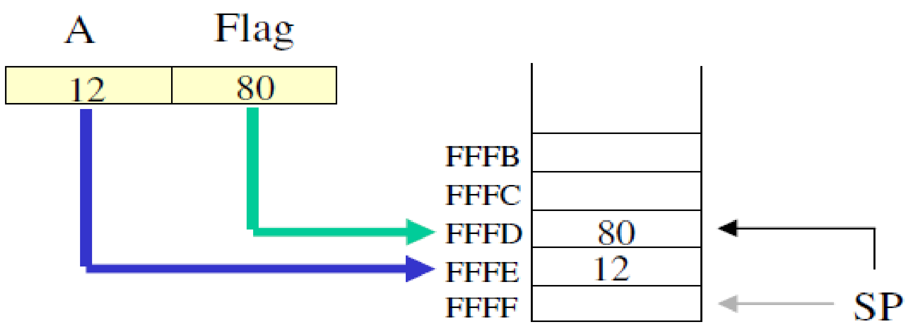
The result is that the contents of the Accumulator and the status of the Flags are returned to what they were before the operations were executed.

PUSH PSW Register Pair

PUSH PSW (1 Byte Instruction).

Decrement SP, Copy the contents of register A to the memory location pointed to by SP.

Decrement SP, Copy the contents of Flag register to the memory location pointed to by SP.

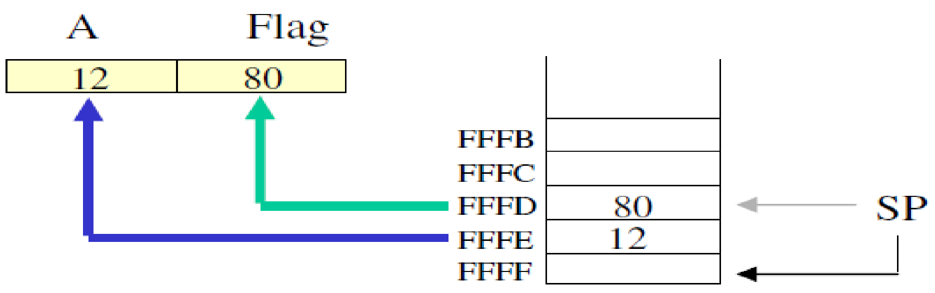


Pop PSW Register Pair

POP PSW (1 Byte Instruction).

Increment SP, Copy the contents of the memory location pointed to by the SP to Flag register

Increment SP, Copy the contents of the memory location pointed to by the SP to register A



Subroutines

A subroutine is a group of instructions that will be used repeatedly in different locations of the program.

Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.

In Assembly language, a subroutine can exist anywhere in the code. However, it is customary to place subroutines separately from the main program.

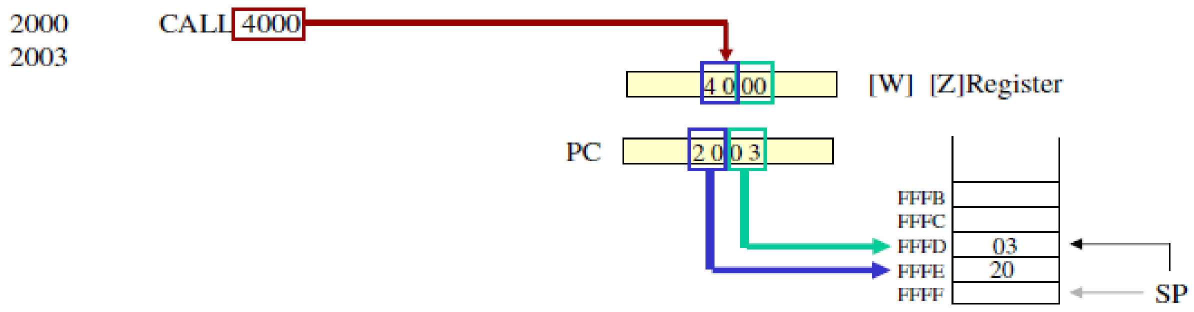
The 8085 has two instructions for dealing with subroutines.

The CALL instruction is used to redirect program execution to the subroutine.

The RET instruction is used to return the execution to the calling routine.

CALL 4000H (3 byte instruction)

When CALL instruction is fetched, the Processor knows that the next two memory location contains 16bit subroutine address in the memory.



The Processor Reads the subroutine address from the next two memory location and stores the higher order 8-bit of the address in the W register and stores the lower order 8-bit of the address in the Z register

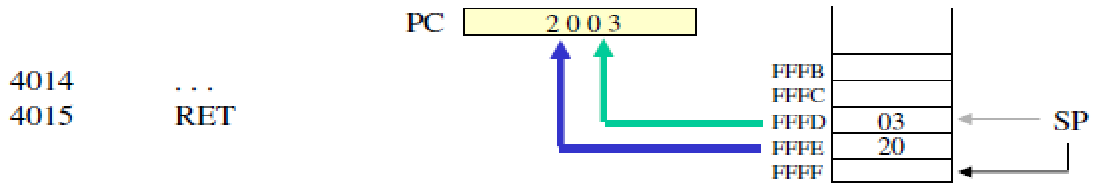
Push the address of the instruction immediately following the CALL onto the stack.

Loads the program counter with the 16-bit address supplied with the CALL instruction from WZ register.

RET (1 byte instruction)

Retrieve the return address from the top of the stack

Load the program counter with the return address.



Subroutine provides flexibility in writing program and uses memory efficiently.

Nesting:

The programming technique of a subroutine calling another subroutine is called nesting of subroutine. This process is limited only by available stack locations.

Recursive Subroutine:

A subroutine which is called by itself is called recursive subroutine.

Reentrant Subroutine:

In nested subroutine, if a latter subroutine calls an earlier subroutine, it is known as reentrant subroutine.

Multiple Ending Subroutines:

The subroutine which has more than one return is called multiple ending subroutines.

Multiple Calling of subroutine:

Calling a subroutine more than once by main program are called multiple calling subroutines.

Parameter passing:

It is necessary pass data and address variable of main program to subroutine. This passing of data and address is called parameter passing. Parameter passing can be done by using register, pointer, memory and stack.

8085 Interrupts

Introduction

Interrupts are mainly used for interrupt driven data transfer between processor and slower input/ output device. The primary function of the microprocessor is to accept data from input devices such as keyboards and A/D converters, read instruction from memory, process data according to the instructions, and send the results to output device such as LEDs, printers and video monitors. The 8085 has five number of interrupt pins: TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.

When the 8085 is interrupted by external devices, it is called hardware interrupt. In software interrupt, the program execution is interrupted by using software instruction.

Types of Interrupts

- Vectored and Non-vectored Interrupts
- Maskable and Non-maskable Interrupts
- Software and Hardware Interrupt

Interrupt Handling Procedure

The Processor will have to store the information about the current program when an interrupt signal is recognized before executing the ISR. The processor checks for the Interrupt request signals at the end of every instruction execution. If the interrupt is masked, then the interrupt will not be recognized until interrupts are re-enabled. The sequence of operations that take place when an interrupt signal is recognized is as followed. The CPU responds to an interrupt request by a transfer of control to another program in a manner similar to a subroutine call. Save the PC (Program Counter) contents (address of the next instruction) and supplementary information about current state (flags, registers, etc.) to the stack. Load PC with the beginning address of an Interrupt Service Routine (ISR) and start to execute it. Finish ISR when return instruction is executed. Return to the interrupted program, exactly to the same point from which it left.

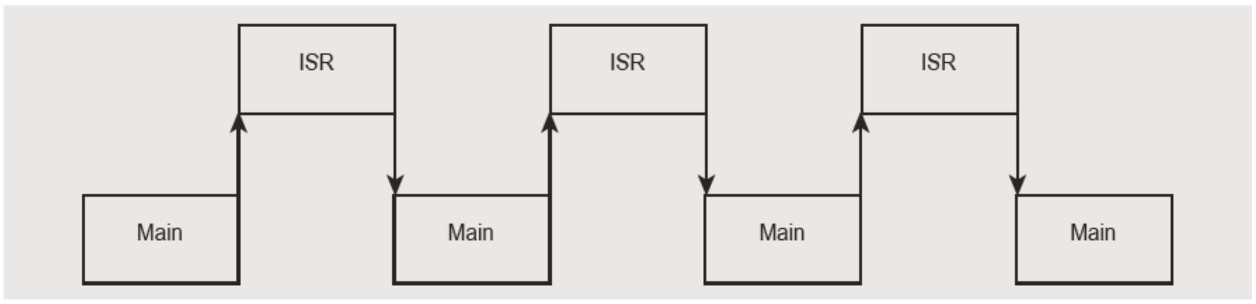


Figure2.5: Transfer of control from Main memory to ISR

Interrupt Sources and their Vector Addresses in 8085

Intel 8085 has the facility for both software and hardware interrupts. The software interrupts are in the form of instructions and the hardware interrupts are applied as signals from the external devices.

Hardware Interrupts and Priorities

Intel 8085 has 5 hardware interrupts: INTR, RT 5.5, RST 6.6, RST 7.5, and TRAP. The details of the five interrupts are given in the table 2.1. Five pins of 8085 are reserved for the five hardware interrupts. All the five interrupts are active high signals. This means that in order to apply an interrupt, logic 1 or high level signal should be applied at these pins. The processor checks the voltage on these pins after the execution of every instruction. If the signal level on any of these 5 pins is at logic 1 and the corresponding interrupt is not masked, then the processor will suspend the current program and execute the corresponding interrupt service routine. RST 7.5 interrupt alone is edge triggered. That means a logic 0 to 1 transition will be treated as an interrupt input on this line. The rising edge interrupt can be applied at any time and this will set a flip flop inside the processor. The processor will check this flip flop while checking the signal level on other hardware interrupts**.**

Table 2.1: Hardware interrupts in 8085

Interrupt	**Interrupt vector address**	**Maskable or Non Maskable**	**Edge-or Level-triggered**	**priority**
Trap	0024H	Non-maskable	Level-triggered	1
RST 7.5	003CH	Maskable	Rising edge triggered	2
RST 6.5	0034H	Maskable	Level-triggered	3
RST 5.5	002CH	Maskable	Level-triggered	4
INTR	Decided by Hardware	Maskable	Level-triggered	5

Software Interrupts

8085 microprocessor instruction set has eight software interrupts instruction called Restart (RST) instructions. These are one-byte call instruction that transfer program execution to subroutine at predefined address. Table2.2 gives the eight software instructions and their corresponding interrupt vector address.

The vector address for a software interrupt is calculated as follows:

Vector address = interrupt number x 8.

The vector address for RST 5 is calculated as

5x 8=40 $_{10}$=28H. So vector address of RST 5 is 0028H

Table 2.2: Software interrupts and their Vector address

Instruction	Machine hex Code	Interrupt Vector address
RST 0	C7	0000H
RST 1	CF	0008H
RST 2	D7	0010H
RST 3	DF	0018H
RST 4	E7	0020H
RST 5	EF	0028H
RST 6	F7	0030H
RST 7	FF	0038H

Masking of Interrupts

The 8085 interrupt structure is shown in figure 2.5.The maskable interrupts are by default masked by the RESET signal. So, any interrupt will not be recognized by the hardware reset. The interrupts can be enabled by the execution of the instruction, EI – Enable interrupts. The three RST interrupts can be selectively masked by having proper word in Accumulator and executing the SIM (Set Interrupt Mask) instruction. This is called software masking. All the maskable interrupts are disabled whenever an interrupt is recognized. So, it is necessary to execute EI instruction every time the interrupts are recognized and serviced by the processor. All the maskable interrupts can be disabled by executing an instruction DI – Disable Interrupts. This instruction will reset an interrupt enable flip flop in the processor and the interrupts will be disabled. To enable interrupts, EI instruction has to be executed.

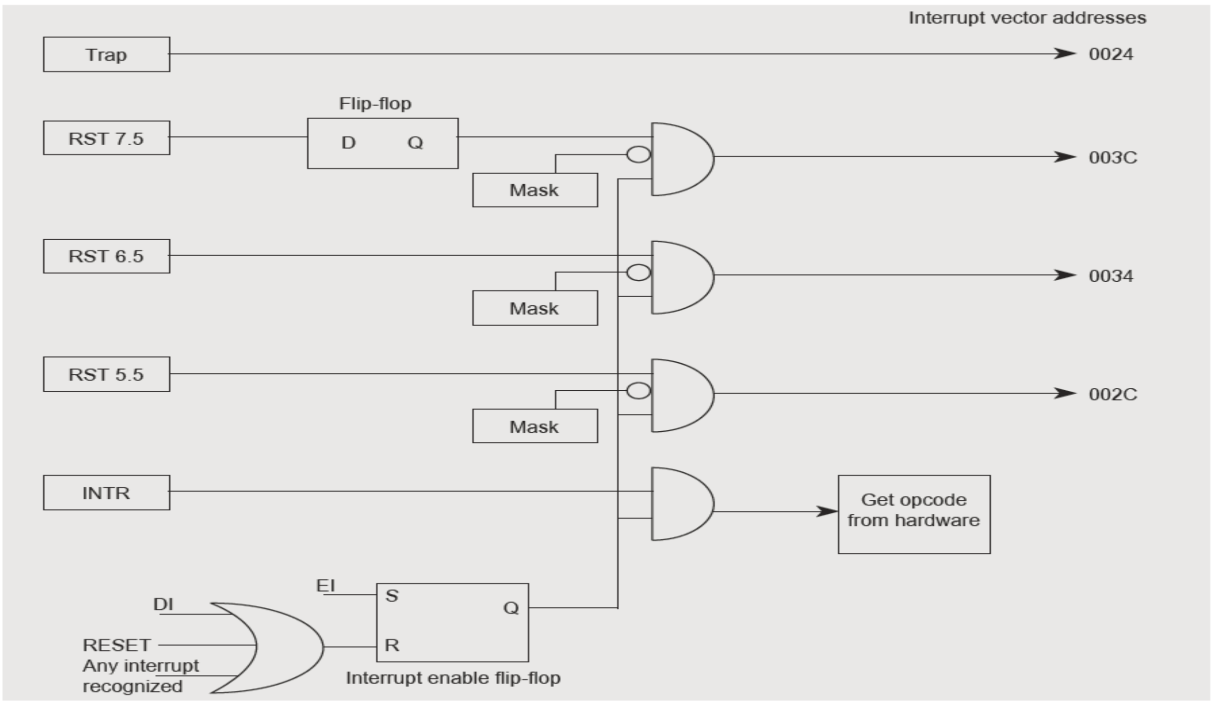
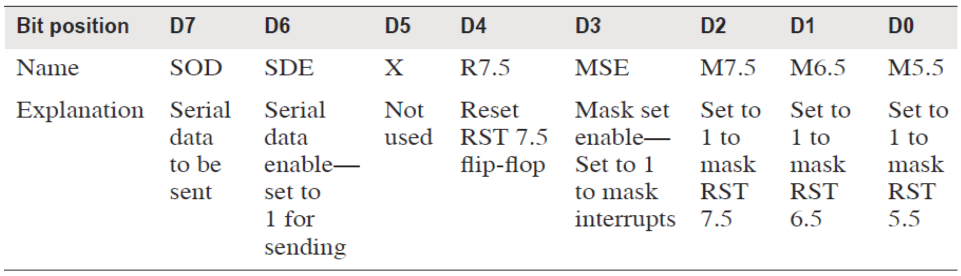


Figure2.6: interrupts structure of 8085

SIM Instruction

The SIM instruction is used to mask or unmask the restart RST hardware interrupts. Figure 2.7 shows the SIM instruction format. The SIM instruction when executed will read the contents of the accumulator and based on that will mask or unmask the interrupts. So, SIM instruction must be executed after storing having proper control word in accumulator. The format of the control word is to be stored in accumulator before executing SIM instruction.



Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SOD	SDE	X	R7.5	MSE	M7.5	M6.5	M5.5
Explanation	Serial data to be sent	Serial data enable—set to 1 for sending	Not used	Reset RST 7.5 flip-flop	Mask set enable—Set to 1 to mask interrupts	Set to 1 to mask RST 7.5	Set to 1 to mask RST 6.5	Set to 1 to mask RST 5.5

Figure2.7: SIM instruction Format

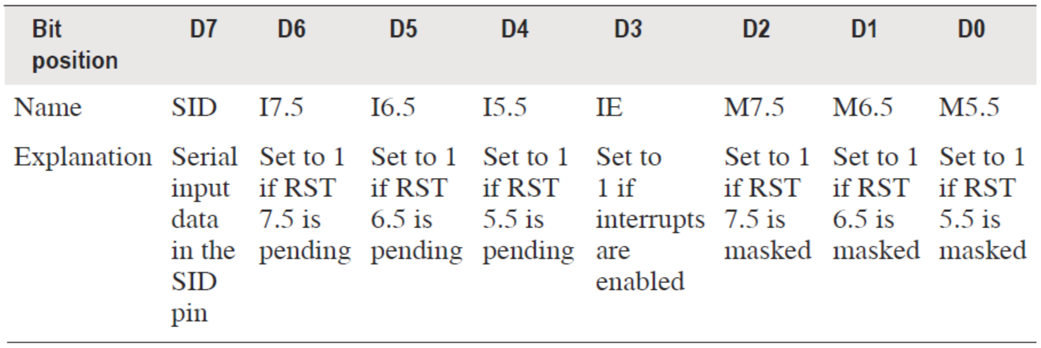
The instructions required for execution of SIM are

MVI A, control word

SIM

RIM Instruction

RIM stands for ‘Read Interrupt Mask’ and its format is shown in figure 2.8. When RIM instruction is executed, the status of serial input data (SID), pending interrupts and interrupt masks are loaded into accumulator.



Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
Explanation	Serial input data in the SID pin	Set to 1 if RST 7.5 is pending	Set to 1 if RST 6.5 is pending	Set to 1 if RST 5.5 is pending	Set to 1 if interrupts are enabled	Set to 1 if RST 7.5 is masked	Set to 1 if RST 6.5 is masked	Set to 1 if RST 5.5 is masked

Figure2.8: RIM instruction Format

The instructions required for execution of RIM are

MVI A, control word

RIM

MODULE-3

PROGRAMMABLE PERIPHERAL INTERFACE: 8255

Introduction

Intel 8085 microprocessor can transfer data between external devices such as input devices and output devices through ports. Normally a register can act as an I/O port. But having a separate register and configuring them for input and output operation becomes difficult and tedious. So, Intel has designed a separate IC 8255 with the objective of interfacing input and output devices with Intel microprocessors. The 8255 Programmable Peripheral Interface (PPI) is a very popular and versatile input / output chip that can be easily programmed to function in several different configurations. The common application of 8255 with 8085 include turning on a switch, to control movement by use of motors, to detect position etc.

Features of 8255

Each 8255 has three 8-bit TTL- compatible registers or ports which will allow the programmers to control digital outputs of up to 24 bits or to read 24 bit inputs, or control a combination of both input and output.

The common features of Intel 8255 IC are as followed.

- Three 8-bit ports named as Port A, Port B, Port C.
- Port C has been divided to two groups of 4 bits each as Port C upper (PCU) and Port C lower (PCL).
- It can operate in mode 0, mode1, mode2, and BSR (bit set reset) mode.
- The port can be programmed for simple I/O or Handshake I/O or bidirectional data bus for the data transfer in I/O modes.
- Each Port C bit can be set/reset individually in BSR mode.

Block Diagram of INTEL 8255

The block diagram of 8255 PPI is shown in figure 3.1. The block diagram consist of Port A, Port C upper Port B, Port C lower, data bus buffer, read write control logic.

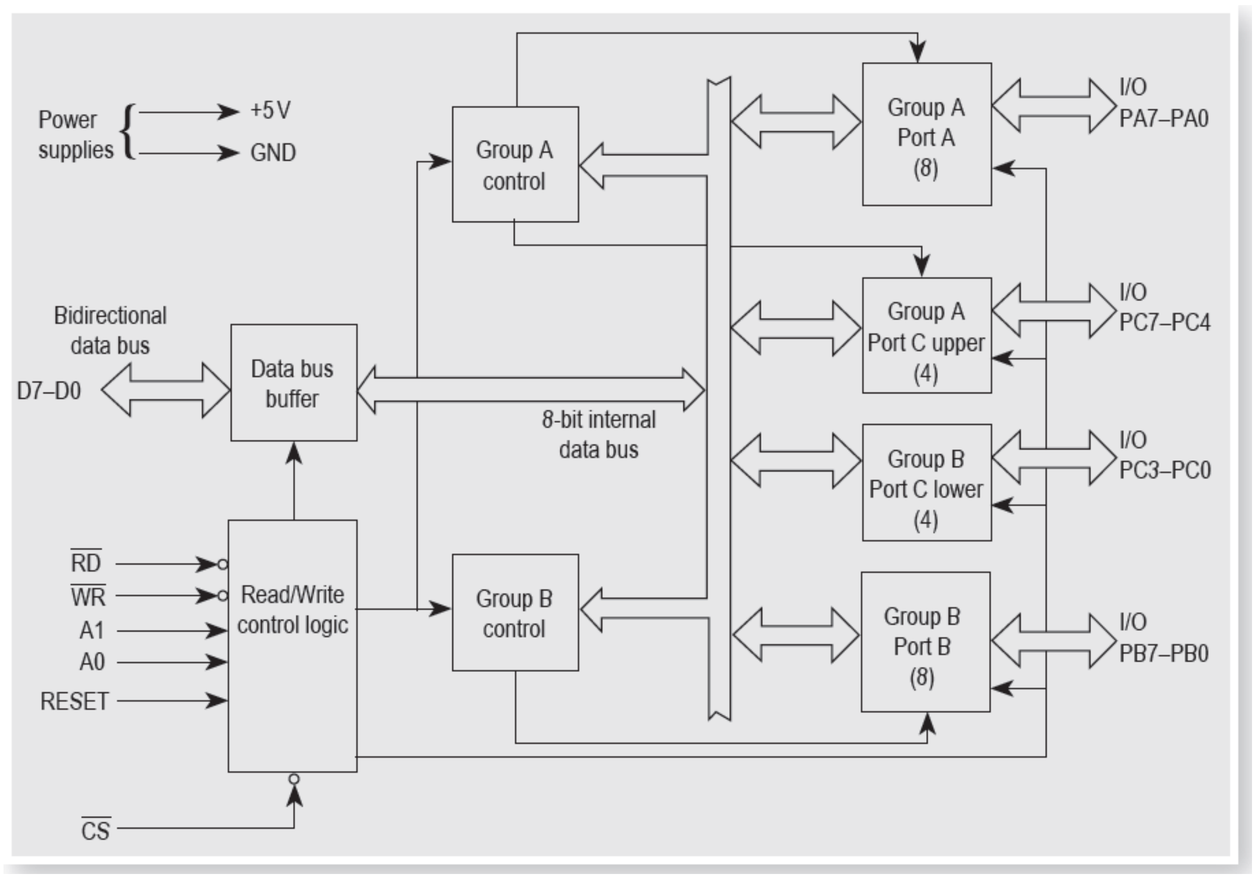
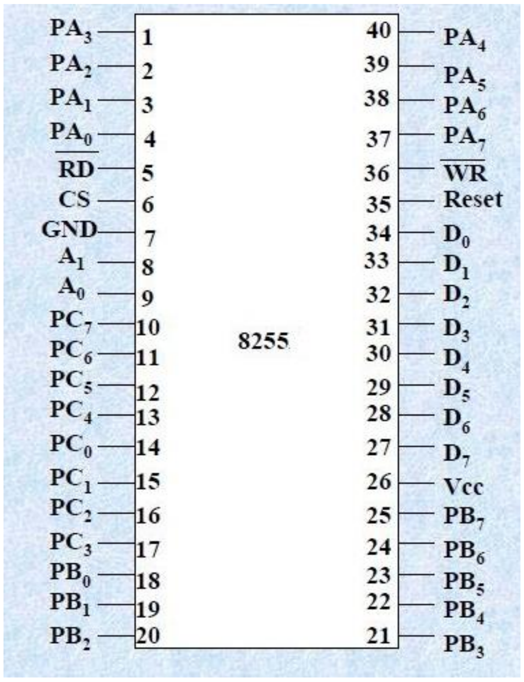


Figure 3.1: Block Diagram of 8255



Pin Diagram of8255

Description of Block diagram

- The block diagram of 8255 (Fig.3.1) has three basic registers of 8 bits each and are called as Port A, Port B and Port C.
- The port A and upper 4 bits of Port C are grouped and called as Group A.
- Similarly, Port B and lower 4 bits of Port C are grouped as Group B.
- The data bus buffer (D7-D0).
- In addition to three Ports, there is another register called control register.
- The contents written into the control register decides the operating modes of the three parallel ports.
- In order to identify Ports and control word register, 8255 uses two address lines A0 and A1.

The selection of the ports and control register (Table 3.1) are based on chip select (CS) signal and A0 and A1 is

Table 3.1: Port and Control Word Register selection

Chip Select	A1	A0	Selection
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Word Register

Control Logic

- RD (Read): When RD signal is low, processor reads data from selected I/O port of 8255.
- WR (write): When WR signal is low, processor writes into a selected I/O port or control register of 8255..
- Reset: when reset is high, it clears control register and sets all ports in the input mode.
- CS, A1 and A0: Chip Select (CS) is connected to decoded address.A1 and A0 of 8255 are connected to A1 and A0 of processor.

To communicate with peripherals through 8255, the following steps are necessary:

- Determine address of Port-A, Port-B, Port-C and control word register.
- Write a control word in control word register.
- Write input and output instruction.

Description of various Modes of 8255

I/O Mode:

Input/output mode is selected when bit D7=1 of the control word register.

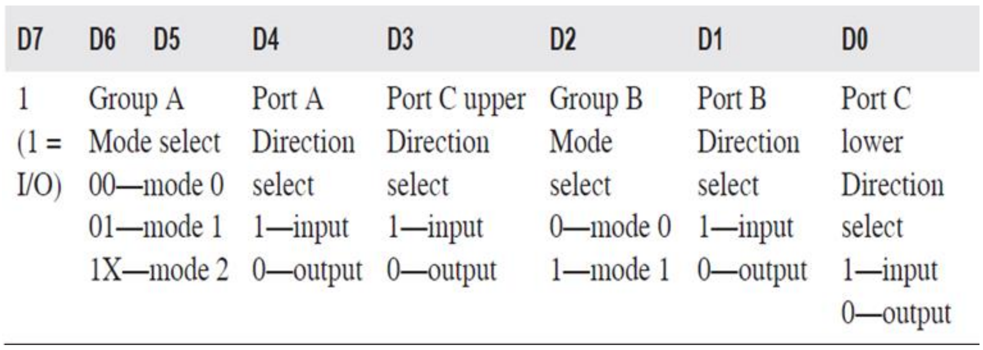
Mode 0: Simple input or output for Port A, Port B, and Port C.

- Outputs are latched.
- Inputs are not latched.
- Ports do not have handshake or interrupt capability.

Mode 1: Handshake I/O for Port A, Port B. Port-C bits is used as handshake signal.

- Inputs are latched.
- Handshake and interrupt capability.

Mode 2: Bi-directional Bus for Port A. Port-C bits is used as handshake signal.

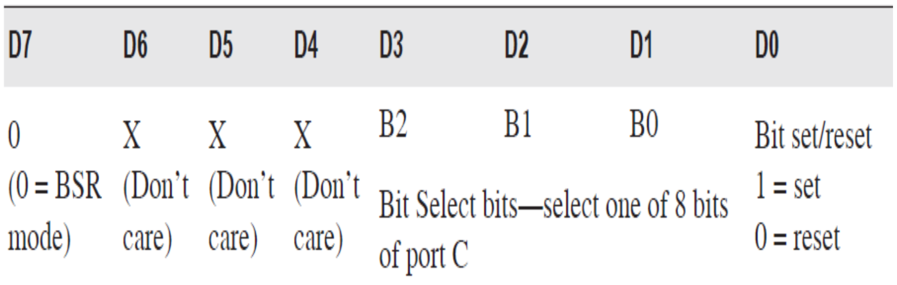


D7	D6 D5	D4	D3	D2	D1	D0
1 (1 = I/O)	Group A Mode select 00—mode 0 01—mode 1 1X—mode 2	Port A Direction select 1—input 0—output	Port C upper Direction select 1—input 0—output	Group B Mode select 0—mode 0 1—mode 1	Port B Direction select 1—input 0—output	Port C lower Direction select 1—input 0—output

Figure 3.2: I/O Control Word Format

BSR Mode:

BSR means bit set reset mode for Port C.BSR mode mode is selected when bit D7=0 of the control word register.



D7	D6	D5	D4	D3	D2	D1	D0
0 (0 = BSR mode)	X (Don't care)	X (Don't care)	X (Don't care)	B2	B1	B0	Bit set/reset 1 = set 0 = reset
				Bit Select bits—select one of 8 bits of port C			

Figure 3.3: BSR mode Control Word Format

- In BSR mode, any of the eight bits of Port C can be Set or Reset.
- It is used for control or on/off switch.
- BSR mode does not affect I/O mode.

Mode1 : INPUT CONTROL SIGNAL

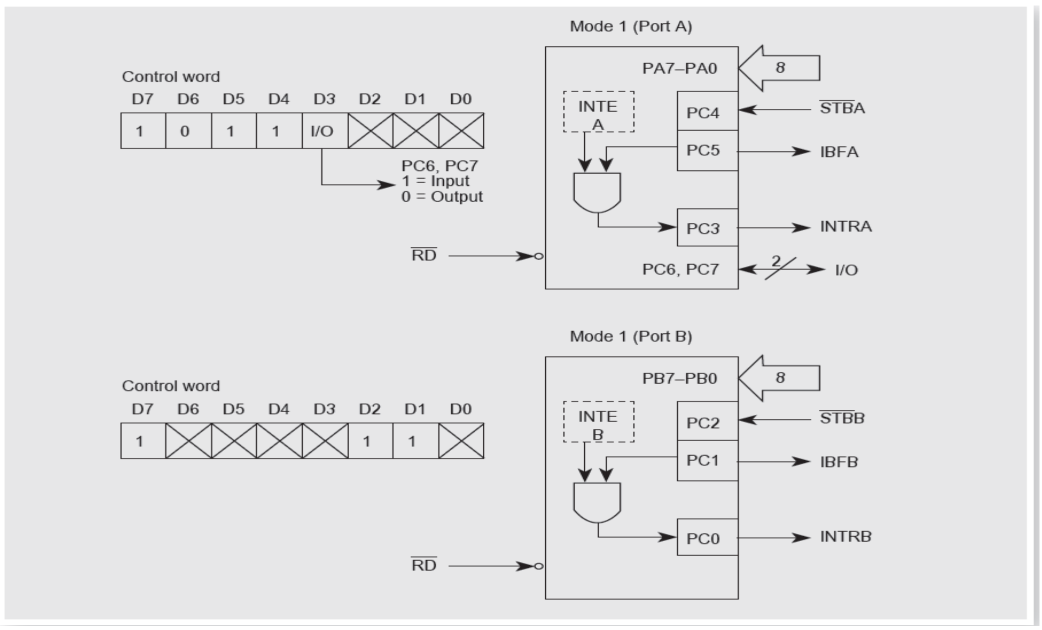


Figure 3.4: Mode1: Input Configuration

The operation of handshake signals for the input operation in 8255 is explained with the figure 3.4. The sequence of operations for the data input operation from an input device to microprocessor through 8255 is listed as follows.

- STB (Strobe Input): The input device places data in the data lines i.e., the Port A or Port B lines. This is communicated to 8255 by making STB low. STB is an active low signal applied to PC4 or PC2.
- IBF (Input Buffer Full): 8255 acknowledges the receipt of the data to the input by making IBF high. This also indicates that the data has been latched into the input port.
- INTR (Interrupt Request): 8255 then makes INTR output line high and applies an interrupt to the processor. INTR signal is generated only if STB, IBF, and INTE are all high at the same time.
- INTE (Interrupt Enable): INTE is an internal flip flop. INTE for Port A is controlled by bit set/reset of PC4 and INTE for port B is controlled by bit set/reset of PC2. PC2 and PC4 can be controlled using BSR mode.
- RD (Read): The processor in the interrupt service routine reads the data from the corresponding input port. Reading from the port is done by selecting the 8255 port and applying RD active low signal. When RD signal goes low, INTR signal is reset. IBF is reset by the rising edge of the RD input.

The entire procedure explained above for mode1 allows an input device to request service from the CPU by simply sending its data into the port and giving STB signal.

Mode1: OUTPUT CONTROL SIGNALS

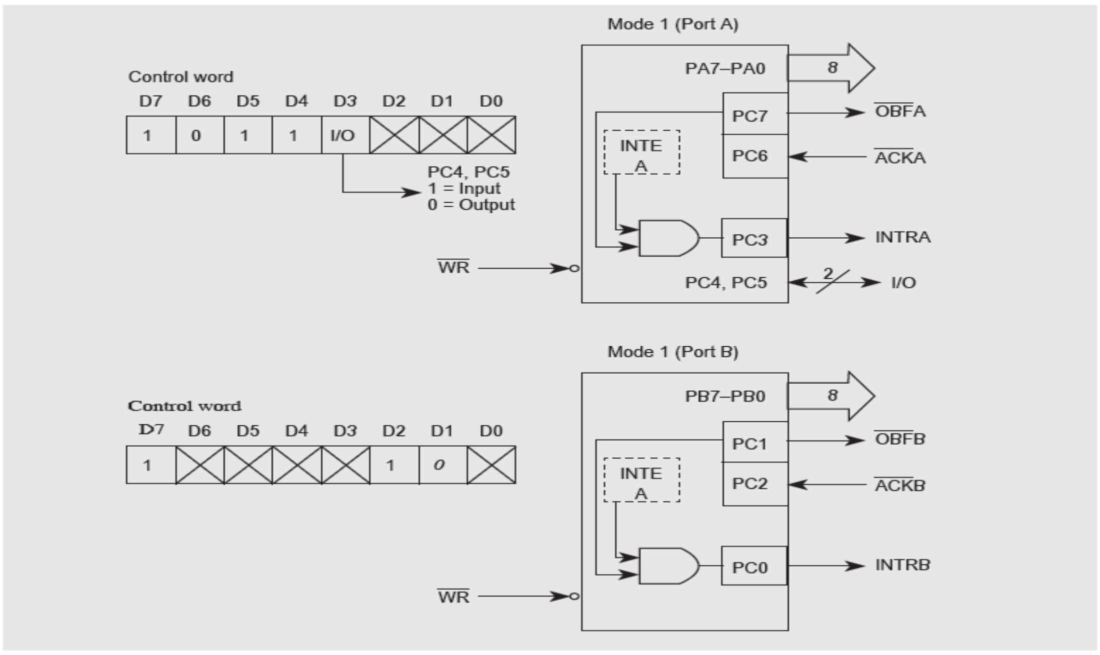


Figure 3.5: 8255 Mode1: Output Configuration

The control signals when Port A and Port B are configured as output ports in mode1 are shown in figure 3.5.

- WR (Write): The processor will initiate the data transmission by writing the data to be transmitted to the output device to the corresponding port of 8255. This is done by processor by sending the port address to 8255 and data on the data lines and then giving the active low WR signal.
- OBF (Output Buffer Full): This is an output signal that goes low when CPU writes data into the output latch of 8255.This signal indicates to an output peripherals that new data is ready to be read. It goes high again after the 8255 receives an acknowledge from peripherals.
- ACK (Acknowledge): This is an input signal from peripheral that goes low when peripheral receives data from 8255 ports.
- INTR (Interrupt Request): This is an output signal. This can be used to interrupt CPU to request the next data byte for output. The INTR is set when OBF.ACK and INTE are all one and reset by falling edge of WR.
- INTE (Interrupt Enable): This is an internal flip flop and set one to generate the INTR signal. $INTE_A$ and $INTE_B$ are controlled by bits PC_6 and PC_2respectively through BSR mode.

Mode 2 : Bidirectional Data Transfer

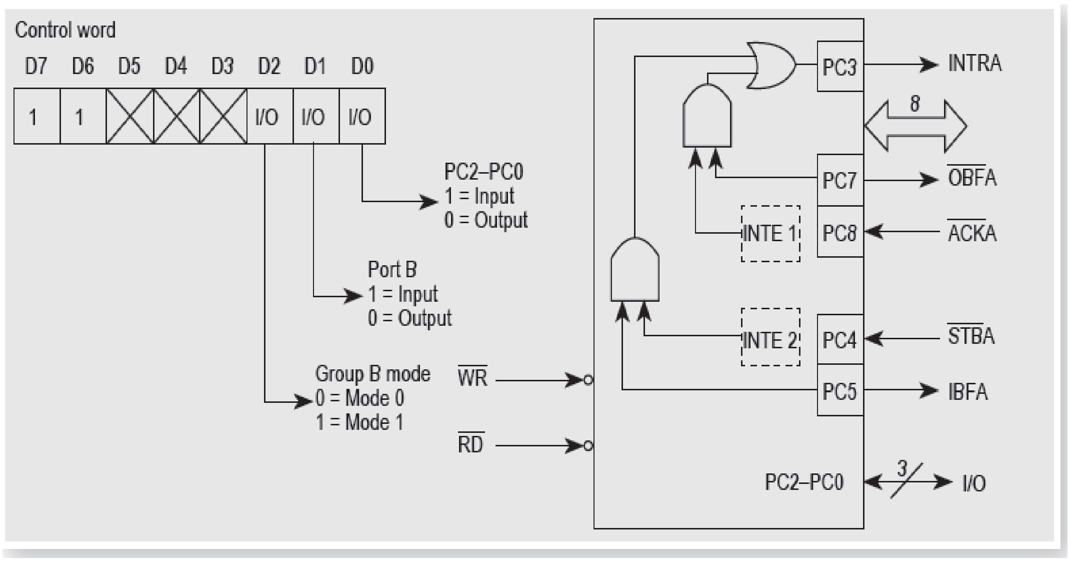


Figure 3.6: 8255 Mode 2: Bidirectional Input/output

Figure 3.6 shows mode2 bidirectional input/output port for Port A. This mode is used primarily in data transfer between two computers. Port A is configured as bidirectional port. Port A uses five signals from Port C as handshake signal for data transfer. The input and output operation of 8255 in mode2 is similar to the operation in mode 1 except that the Port A is bidirectional port. The data is transmitted and received through the Port A lines.

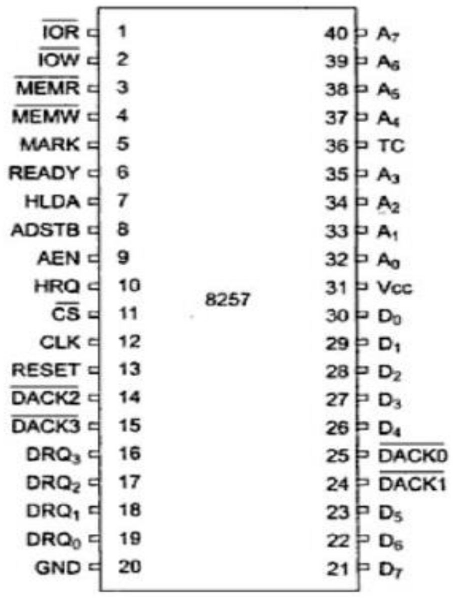
PROGRAMMABLE DMA CONTROLLER 8257

Introduction

8257 is Intel direct memory access controller (DMA) IC. Peripheral devices access memory directly by passing processor. So it is faster method of memory access over programmed data transfer for memory access. The features of 8257 are as follows:

- It is a four channel DMA controller IC.
- A maximum 16 KB data transfer can be done sequentially at a time.
- 8257 is initialized for each channel by starting address, number of bytes of data to be transferred, mode of operation.
- It executes 3 DMA cycles: DMA read, DMA write, DMA verify.
- It generates a TC signal to indicate the peripheral that the programmed numbers of data bytes have been transferred.
- It generates MARK signal to indicate the peripheral that 128 bytes have been transferred.
- It provide AEN signal that can be used to isolate CPU and other devices from the system bus.
- It operates in two modes: Master Mode, Slave Mode.

Figure 3.7 shows block diagram of 8257 DMA controller. The internal block diagram of 8257 consists of eight blocks: data bus buffer, read/write block, control logic and mode set register, priority resolver and four channel blocks.



Pin Diagram of 8257

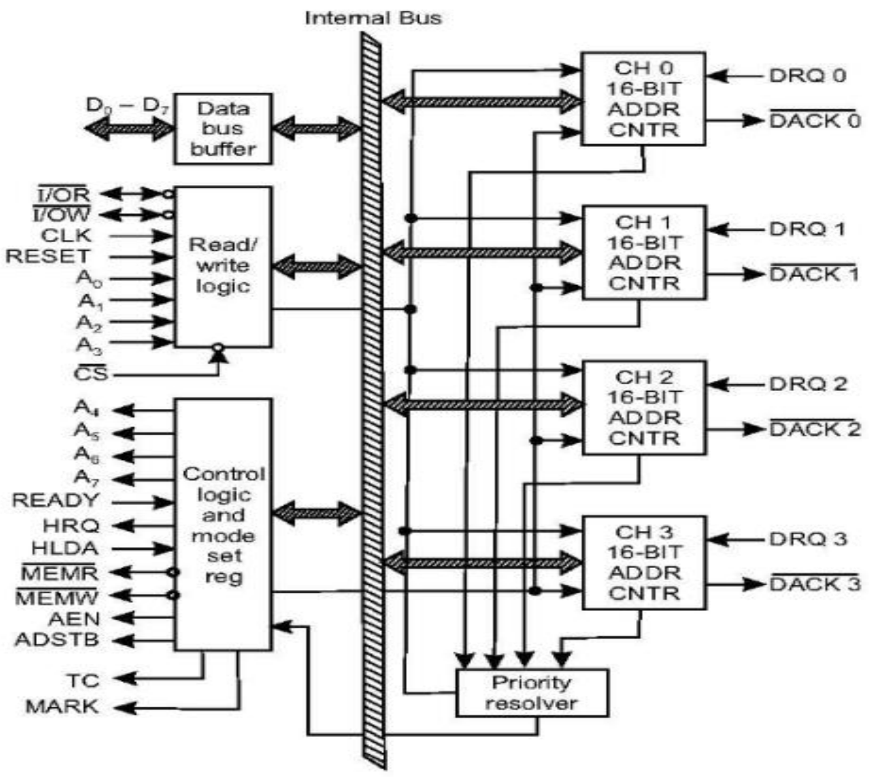


Figure 3.7: Block Diagram of 8257

- Each channel of 8257 Block diagram has two programmable 16-bit registers named as address register and count register.

- Address register is used to store the starting address of memory location for DMA data transfer.
- The address in the address register is automatically incremented after every read/write/verify transfer.
- The count register is used to count the number of byte or word transferred by DMA. The format of count register is shown in figure 3.8.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

0 0	Verify transfer
0 1	Write transfer
1 0	Read transfer
1 1	Illegal

Figure 3.8: Count Register Format

- 14-bits B0-B13 is used to count value and a 2-bits is used for indicate the type of DMA transfer (Read/Write/Veri1 transfer).
- In read transfer the data is transferred from memory to I/O device.
- In write transfer the data is transferred from I/O device to memory.
- Verification operations generate the DMA addresses without generating the DMA memory and I/O control signals.

The 8257 has two eight bit registers called mode set register and status register. The format of mode set register is shown in figure 3.9.

B7	B6	B5	B4	B3	B2	B1	B0
Auto-load	TC	EW	RP	CH-3	CH-2	CH-1	CH-0

Figure 3.9: Mode Set Register Format

The use of mode set register is

- Enable/disable a channel.
- Fixed/rotating priority
- Stop DMA on terminal count.
- Extended/normal writes time.
- Auto reloading of channel-2.

- The bits B0, B1, B2, and B3 of mode set register are used to enable/disable channel -0, 1, 2 and 3 respectively. A one in these bit position will enable a particular channel and a zero will disable it.
- If the bit B4 is set to one, then the channels will have rotating priority and if it zero then the channels wilt have fixed priority.
    1. In rotating priority after servicing a channel its priority is made as lowest.
    2. In fixed priority the channel-0 has highest priority and channel-2 has lowest priority.

- If the bit B5 is set to one, then the timing of low write signals (MEMW and IOW) will be extended.
- If the bit B6 is set to one then the DMA operation is stopped at the terminal count.
- The bit B7 is used to select the auto load feature for DMA channel-2.
- When bit B7 is set to one, then the content of channel-3 count and address registers are loaded in channel-2 count and address registers respectively whenever the channel-2 reaches terminal count. When this mode is activated the number of channels available for DMA reduces from four to three.

Figure 3.10 shows the eight bit status register format of 8257.It is used to read the status of terminal count of the four channels (CH0-CH3).

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	UF	CH-3	CH-2	CH-1	CH-0

Figure 3.10: Status Register Format

- The bit B0, B1, B2, and B3 of status register indicates the terminal count status of channel-0, 1,2 and 3 respectively. A one in these bit positions indicates that the particular channel has reached terminal count.
- These status bits are cleared after a read operation by microprocessor.
- The bit B4 of status register is called update flag and a one in this bit position indicates that the channel-2 register has been reloaded from channel-3 registers in the auto load mode of operation.

PROGRAMMABLE INTERRUPT CONTROLLER: 8259

Introduction

The Intel 8259 is a Programmable Interrupt Controller (PIC) designed for use with the 8085 and 8086 microprocessors. The 8085 has only five number of hardware interrupts: TRAP,RST 7.5.RST 6.5,RST 5.5 and INTR. The 8259 can be used for applications that use more than five numbers of interrupts from multiple sources.

Features and Architecture of 8259

The main features of 8259 are listed below.

- Manage eight levels of interrupts.
- Eight interrupts are spaced at the interval of four or eight locations.
- Resolve eight levels of priority in fully nested mode, automatic rotation mode or specific rotation mode.
- Mask each interrupt individually.
- Read the status of pending interrupt, in-service interrupt, and masked interrupt.
- Accept either the level triggered or edge triggered interrupt.
- Can be expanded to 64 priority levels by cascading additional interrupts.Eight slave 8259s may be cascaded to a master 8259 to provide up to 64 priority levels.

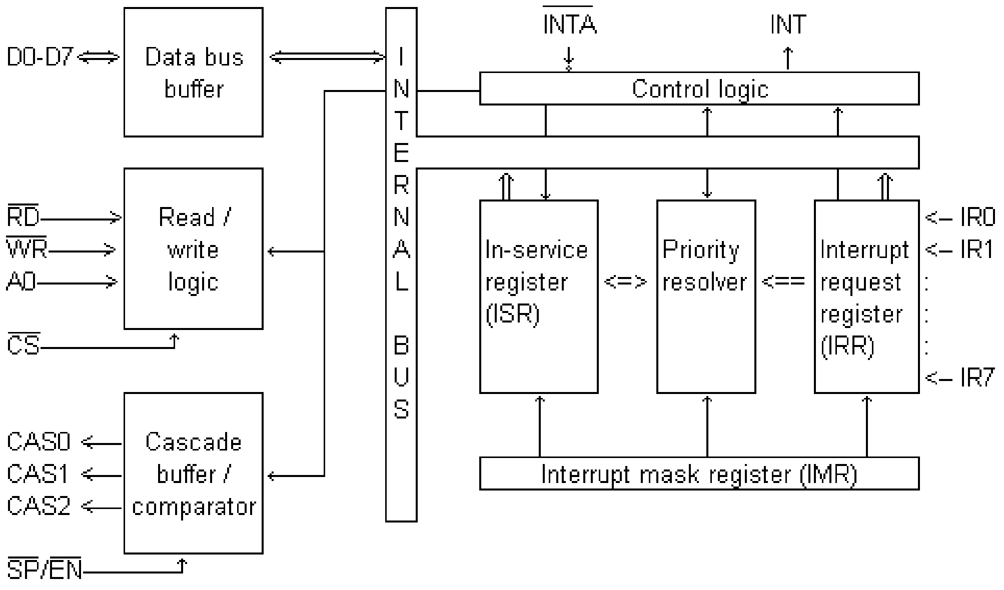
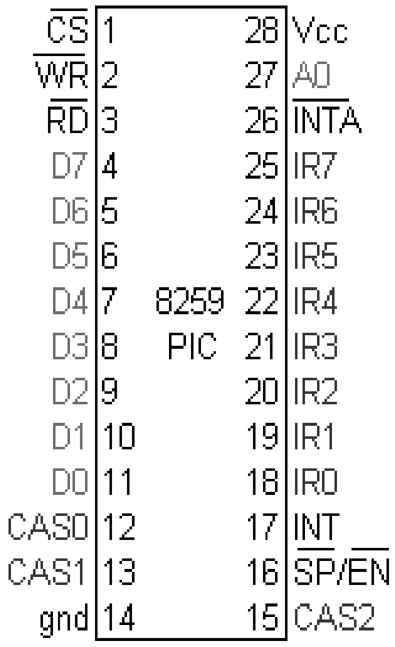


Figure 3.11: 8259 Internal Block Diagram

Figure 3.11 shows the internal block diagram of 8259. It includes eight blocks: control logic, read/write logic, data bus buffer, three registers (IRR, ISR, and IMR), priority resolver and cascade buffer.



Pin Diagram of 8259

- A0: Address input line, used to select control register.
- CAS 0 - CAS 2: Bi-directional, 3 bit cascade lines. In master mode, PIC places slave ID number on these lines. In slave mode, the PIC reads slave ID number from master on these lines. It may be regarded as slave-select.
- SP/EN (slave program / enable): In non-buffered mode, it is SP- input, used to distinguish master/slave PIC. In buffered mode, it is output line used to enable buffers
- INT: Interrupt line, connected to INTR of microprocessor.

Interrupt Operation

The Interrupt Enable (IE) flip-flop is enabled by writing the EI instruction. The 8259 is initialized by writing control words in the control register: Initialization Command Words (ICWs) and Operational Command Word (OCWs). ICWs are used to specify interrupt vector address. OCWs perform masking of interrupts, status read operation. After 8259 is initialized, the following sequence of events occurs when one interrupt request line go high:

- The IRR stores the request.
- The priority resolver checks three registers: the IRR, IMR, ISR. It resolves the priority and sets the INT pin high.
- The processor acknowledges the interrupt by sending INTA.
- After INTA is received, the priority bit in the ISR is set, the corresponding bit in IRR is reset. Then the opcode for the CALL instruction is placed on the data bus.
- When the processor decodes the CALL instruction, it places two more INTA signals on the data bus.
- When the 8259 receives the second INTA, it places the low order byte of the CALL address on the data bus. At the third INTA, it places the high-order byte on the data bus. This address is placed in the control register during initialization.
- During the third INTA, the ISR bit is reset.
- The program sequence is transferred to specified CALL address.

ICW1 (Initialization Command Word 1):

ICW1 specify lower byte of ISR CALL address.

A0
0

D7	D6	D5	D4	D3	D2	D1	D0
A7	A6	A5	1	LTIM	ADI	SNGL	IC4

D0: IC4: 0=no ICW4, 1=ICW4 required

D1: SNGL: 1=Single PIC, 0=Cascaded PIC

D2: ADI: Address interval used only in 8085, not 8086. 1=ISR's are 4 bytes apart (0200, 0204, etc)

0=ISR's are 8 byte apart (0200, 0208, etc)

D3: LTIM: level triggered interrupt mode: 1=All IR lines level triggered, 0=edge triggered

D4-D7: A5-A7: 8085 only. ISR address lower byte segment.

The lower byte is

A7	A6	A5	A4	A3	A2	A1	A0

of which A7, A6, A5 are provided by D7-D5 of ICW1 (if ADI=1), or A7, A6 are provided if ADI=0. A4-A0 (or A5-A0) is set by 8259 itself:

ADI=1 (spacing 4 bytes)

IRQ	A7	A6	A5	A4	A3	A2	A1	A0
IR0	A7	A6	A5	0	0	0	0	0
IR1	A7	A6	A5	0	0	1	0	0
IR2	A7	A6	A5	0	1	0	0	0
IR3	A7	A6	A5	0	1	1	0	0
IR4	A7	A6	A5	1	0	0	0	0
IR5	A7	A6	A5	1	0	1	0	0
IR6	A7	A6	A5	1	1	1	0	0
IR7	A7	A6	A5	1	1	1	0	0

ADI=0 (spacing 8 bytes)

IRQ	A7	A6	A5	A4	A3	A2	A1	A0
IR0	A7	A6	0	0	0	0	0	0
IR1	A7	A6	0	0	1	0	0	0
IR2	A7	A6	0	1	0	0	0	0
IR3	A7	A6	0	1	1	0	0	0
IR4	A7	A6	1	0	0	0	0	0
IR5	A7	A6	1	0	1	0	0	0
IR6	A7	A6	1	1	0	0	0	0
IR7	A7	A6	1	1	1	0	0	0

ICW2 (Initialization Command Word 2):

ICW2 specify higher byte of ISR CALL address (8085), or 8 bit vector address (8086).

A0
1

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

ICW3 (Initialization Command Word 3):

ICW3 is required only if several 8259's are used in cascaded form.ICW3 can operate in master mode and slave mode.

A0
1

	D7	D6	D5	D4	D3	D2	D1	D0
Master	S7	S6	S5	S4	S3	S2	S1	S0
Slave	0	0	0	0	0	ID3	ID2	ID1

- Master mode: 1 indicates slave is present on that interrupt, 0 indicates direct interrupt
- Slave mode: ID3-ID2-ID1 is the slave ID number. Slave 4 on IR4 has ICW3=04h (0000 0100)

ICW4 (Initialization Command Word 4)

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	SFNM	BUF	M/S	AEOI	Mode

- SFNM: 1=Special Fully Nested Mode, 0=FNM
- M/S: 1=Master, 0=Slave
- AEOI: 1=Auto End of Interrupt, 0=Normal
- Mode: 0=8085, 1=8086

OCW1 (Operational Command Word 1)

OCW1 specify masking of interrupts.

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

IRn is masked by setting Mn to 1; mask cleared by setting Mn to 0 (n=0..7)

OCW2 (Operational Command Word 2)

OCW2 control the rotate and End of interrupt modes.

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	R	SL	EOI	0	0	L3	L2	L1

	R	SL	EOI	Action
EOI	0	0	1	Non specific EOI (L3L2L1=000)
	0	1	1	Specific EOI command (Interrupt to clear given by L3L2L1)
Auto rotation of priorities (L3L2L1=000)	1	0	1	Rotate priorities on non-specific EOI
	1	0	0	Rotate priorities in auto EOI mode set
	0	0	0	Rotate priorities in auto EOI mode clear
Specific rotation of priorities (Lowest priority ISR=L3L2L1)	1	1	1	Rotate priority on specific EOI command (resets current ISR bit)
	1	1	0	Set priority (does not reset current ISR bit)
	0	1	0	No operation

OCW3 (Operational Command Word 3)

OCW3 read the status of register, set/reset special mask and polled modes.

A0
1

D7	D6	D5	D4	D3	D2	D1	D0
D7	ESMM	SMM	0	1	MODE	RIR	RIS

ESMM	SMM	Effect
0	X	No effect
1	0	Reset special mask
1	1	Set special mask

Priority Modes:

- Fully Nested Mode: All IRs are arranged from highest to lowest priority.
  IR0=highest priority, IR7=lowest priority
- Automatic Rotation Mode: A device after being serviced receives the lowest priority.
- Specific Rotation Mode: User selects any IR for lowest priority.

PROGRAMMABLE INTERVAL TIMER: 8253

INTRODUCTION

In software programming of 8085, it has been shown that a delay subroutine can be programmed to introduce a predefined time delay. The delay is achieved by decrementing a count value in a register using instructions. The disadvantage of this software approach is that the processor is locked in the delay loop and the precious processor time is wasted in just counting. This disadvantage can be overcome by using the hardware timer and interrupts. IC 555 can be used to generate the timing signals, but only at a fixed time interval. This can't be easily interfaced with the microprocessor. So, Intel has produced programmable timer devices namely IC 8253 and IC 8254. These devices can be programmed to generate different types of delay signals and also count external signals. Other counter/timer functions that are also common to be implemented with the 8253 are Programmable frequency square wave Generator, Event Counter, Real Time Clock, Digital One-Shot and Complex Motor Controller.

Features of 8253

Timer ICs 8253 and 8254 are manufactured by Intel with the similar operating functions. 8254 can be operated at frequency of up to 8MHz whereas 8253 can be operated only up to a maximum frequency of 2 MHZ.

- Generation of accurate time delay
- Three independent 16-bit down counters.
- Six different programmable operating modes

- Timer or counter operation.
- Can count in binary or BCD
- Can be used to interrupt the processor.
- Single +5V supply
- Can operate from DC to 2 MHZ.

Block Diagram of 8253

Figure 3.12 is the block diagram of 8253. It include three counters (counter 0, 1and 2), a data bus buffer, Read/write control logic and control register. Each counter has two input signals-clock (CLK) and GATE- and one output signal-OUT.

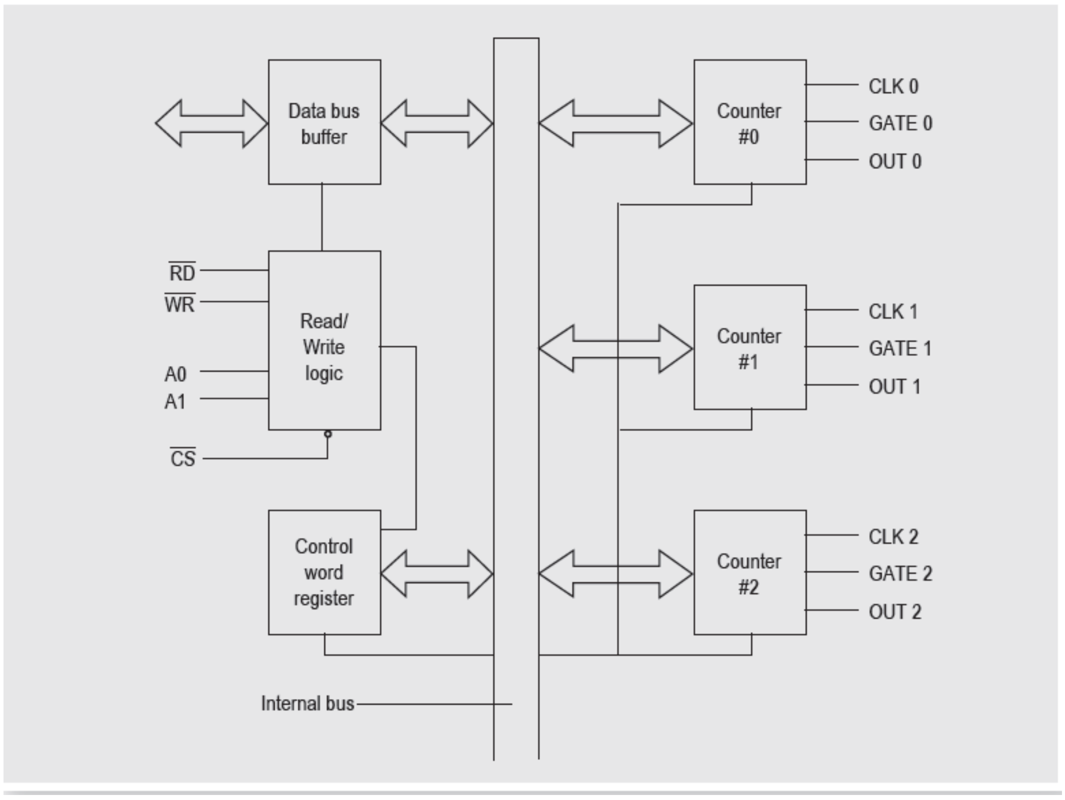
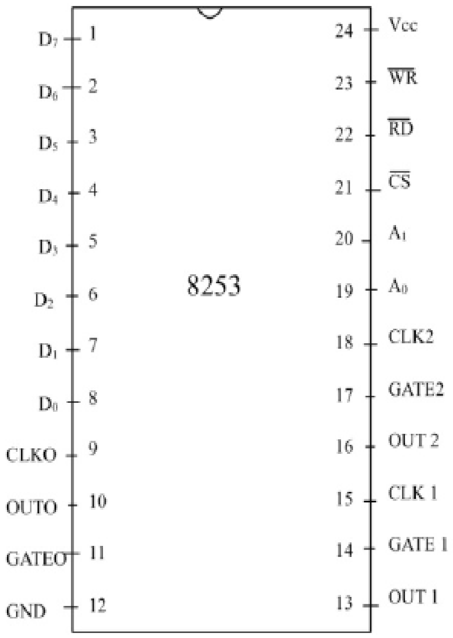


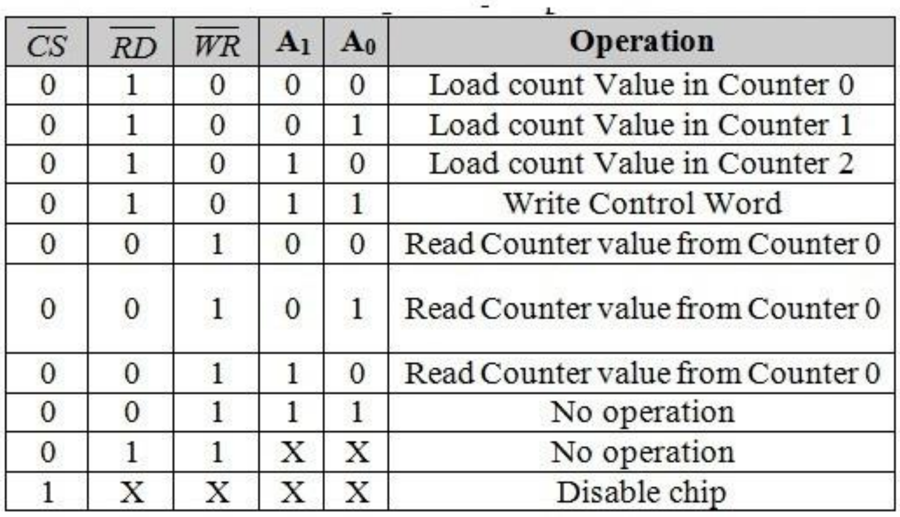
Figure 3.12: Block Diagram of 8253



Pin Diagram of 8253

The control word register and counters are selected according to the signals on lines A0 and A1 as follows:

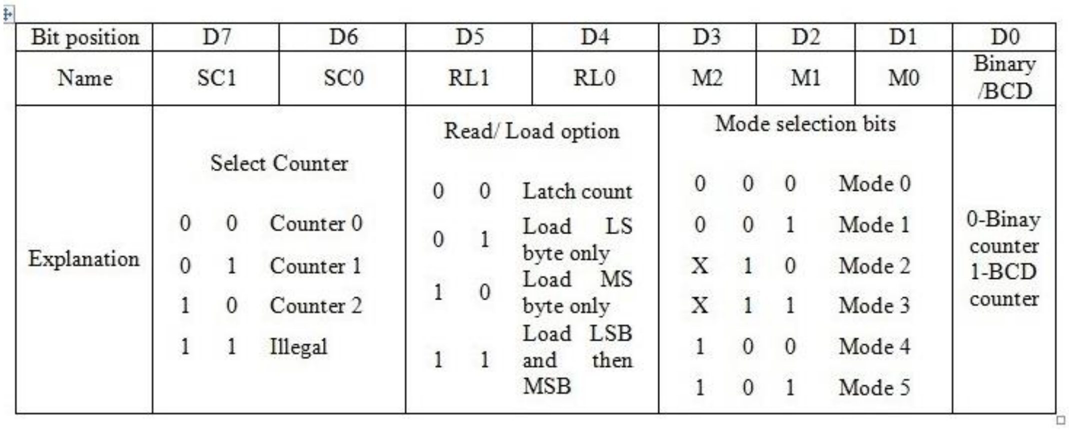
A1	A0	Selection
0	0	Counter0
0	1	Counter1
1	0	Counter2
1	1	Control Register



$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	**Operation**
0	1	0	0	0	Load count Value in Counter 0
0	1	0	0	1	Load count Value in Counter 1
0	1	0	1	0	Load count Value in Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read Counter value from Counter 0
0	0	1	0	1	Read Counter value from Counter 0
0	0	1	1	0	Read Counter value from Counter 0
0	0	1	1	1	No operation
0	1	1	X	X	No operation
1	X	X	X	X	Disable chip

Control Word Register

The control word format is shown in figure 3.13. This register is accessed when A0 and A1 at logic 1. It is used to write a command word which specifies the counter to be used, its mode, and either a Read or write operation.

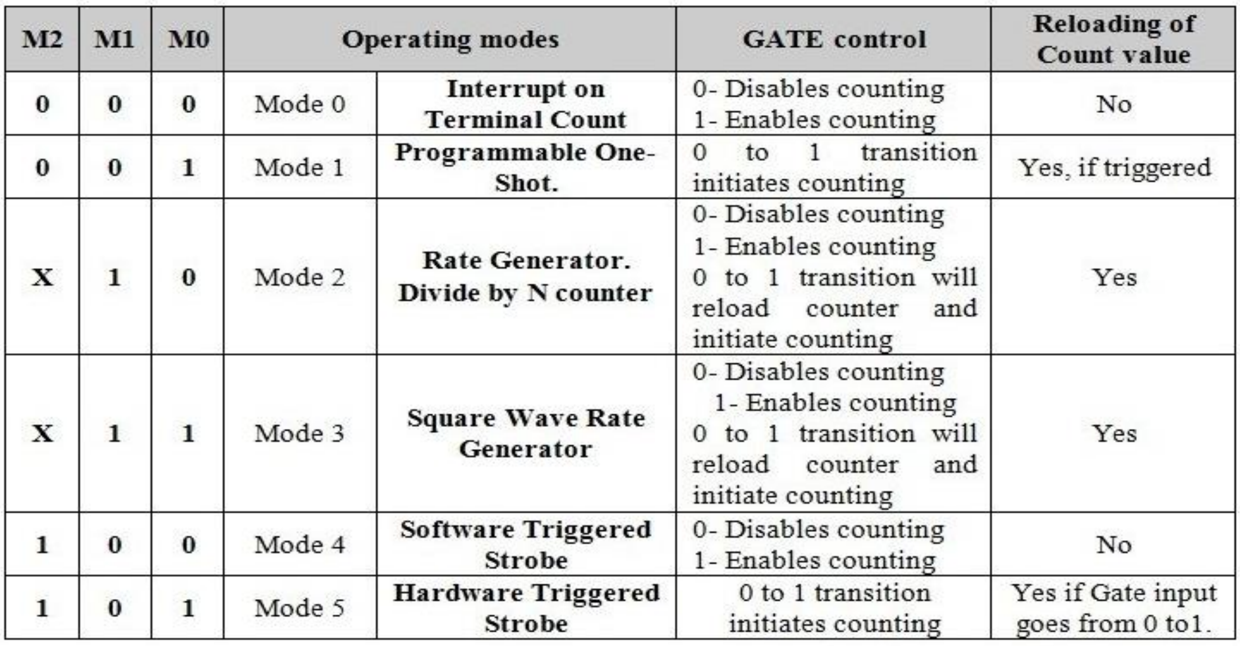


Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SC1	SC0	RL1	RL0	M2	M1	M0	Binary /BCD
Explanation	Select Counter 0 0 Counter 0 0 1 Counter 1 1 0 Counter 2 1 1 Illegal		Read/ Load option 0 0 Latch count 0 1 Load LS byte only 1 0 Load MS byte only 1 1 Load LSB and then MSB		Mode selection bits 0 0 0 Mode 0 0 0 1 Mode 1 X 1 0 Mode 2 X 1 1 Mode 3 1 0 0 Mode 4 1 0 1 Mode 5			0-Binay counter 1-BCD counter

Figure 3.13: 8253 Control Word Format

Mode:

Figure 3.14 shows the operating modes of 8253. It operates in 6 different modes: mode0, mode1, mode2, mode3, mode4, and mode5.



M2	M1	M0	Operating modes		GATE control	Reloading of Count value
0	0	0	Mode 0	**Interrupt on Terminal Count**	0- Disables counting 1- Enables counting	No
0	0	1	Mode 1	**Programmable One-Shot.**	0 to 1 transition initiates counting	Yes, if triggered
X	1	0	Mode 2	**Rate Generator. Divide by N counter**	0- Disables counting 1- Enables counting 0 to 1 transition will reload counter and initiate counting	Yes
X	1	1	Mode 3	**Square Wave Rate Generator**	0- Disables counting 1- Enables counting 0 to 1 transition will reload counter and initiate counting	Yes
1	0	0	Mode 4	**Software Triggered Strobe**	0- Disables counting 1- Enables counting	No
1	0	1	Mode 5	**Hardware Triggered Strobe**	0 to 1 transition initiates counting	Yes if Gate input goes from 0 to1.

Figure 3.14: Operating Mode of 8253

Programming the 8253

The 8253 can be programmed to provide various types of outputs through Write operation, or to check a count while counting through Read operations.

Write operation

To initialize a counter, the following steps are necessary

- Write a control word into the control register.
- Load the low –order byte of a count in the counter register.
- Load the high –order byte of a count in the counter register.

Read Operation

In event counters application, it is necessary to read count value in progress. This can be done by either two methods. One method involves reading a count after stopping the counter to be read. The second method involves reading a count while the count is in progress.

In the first method, counting is stopped by controlling the gate input or the clock input of the selected counter, and two I/O read operation are performed by CPU. The first I/O operation reads the lower byte and the second I/O operation reads the higher byte. In the second method an appropriate control word is written into the control register to latch a count in output latch, and two I/O read operation are performed by the CPU.

Mode 0: Interrupt on Terminal Count

In this mode, initially the OUT is low. Once a count is loaded in the register, the counter is decremented every cycle. When count reaches zero, the OUT goes high. This can be used as an interrupt. The OUT remains until a new count or a command is loaded.

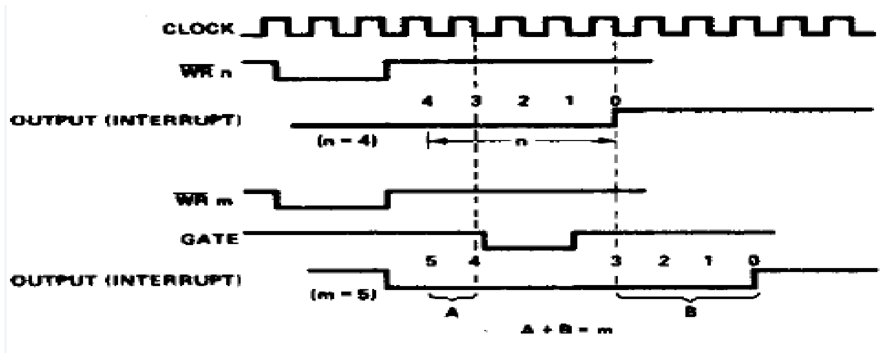


Figure 3.15: Mode 0: Interrupt on Terminal Count

Mode 1: Hardware-Triggered One Shot

In this mode, the OUT is initially high. When the Gate is triggered, the OUT goes low, and at the end of count, the OUT goes high again, generating one shot pulse.

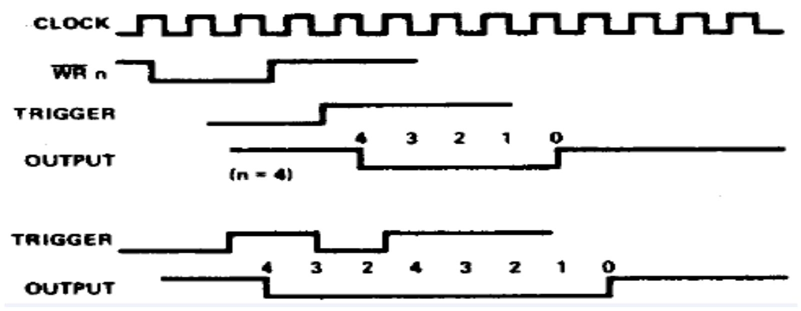


Figure 3.16: Mode 1: Programmable one shot

Mode 2: Rate Generator

This mode is used to generate a pulse equal to the clock period at a given interval. When a count is loaded, the OUT stays high until the count reaches 1, and then the count goes low for one clock period. The count is reloaded automatically, and the pulse is generated continuously. The count = 1 is illegal in this mode.

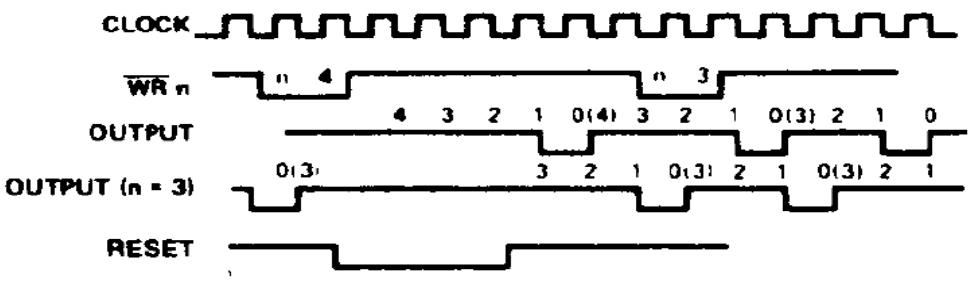


Figure 3.17: Mode 2: Rate Generator.

Mode 3: Square Wave Generator

In this mode, when count is loaded, the OUT is high. The count is decremented by two at every clock cycle. When it reaches zero, the OUT goes low, and the count is reloaded again. This is repeated continuously to generate a square wave with period equal to period of count generated. The frequency of the square wave is equal to the frequency of the clock divided by count.

- High for N/2 counts, and low for N/2 counts, if count N is even.
- High for (N+1)/2 counts, and low for (N-1)/2 counts, if N is odd.

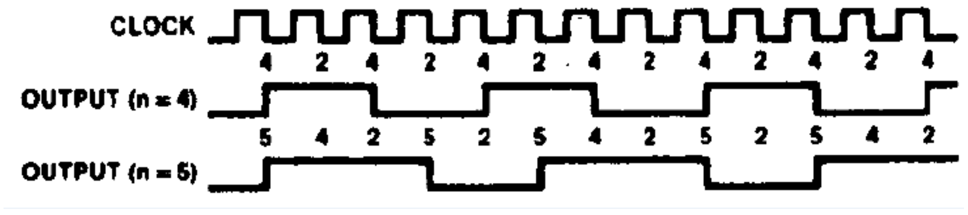


Figure 3.18: Mode 3: Square Wave Generator

Mode 4: Software Triggered Strobe

In this mode the OUT is initially high. It goes low for one clock period at the end of count. The count is reloaded for subsequent outputs.

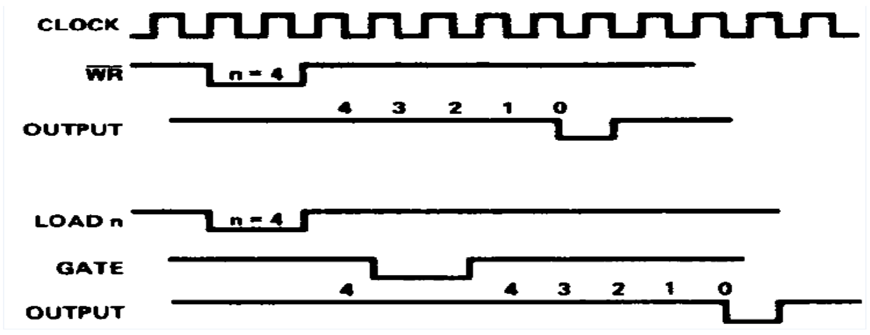


Figure 3.19: Mode 4: Software Triggered Strobe

Mode 5: Hardware Triggered Strobe

This mode is similar to mode 4, except that it is triggered by the rising edge of GATE input. Initially the OUT is low, and when the Gate pulse is triggered from low to high, the count begins. At the end of count, the OUT goes low for one clock period.

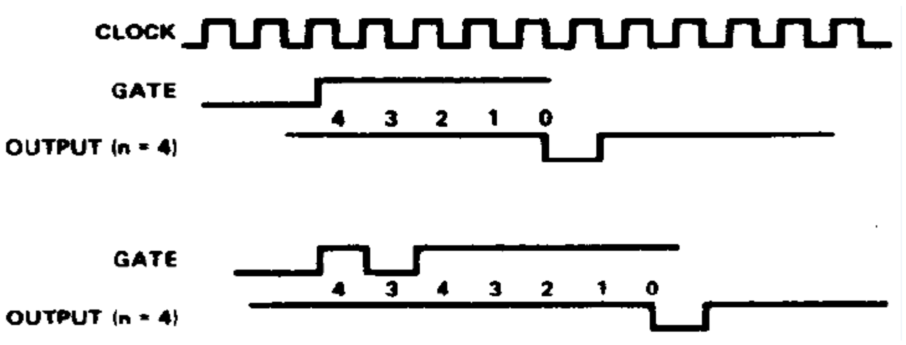


Figure 3.20: Mode5: Hardware Triggered Strobe

MODULE-4

Intel 8086 Introduction:

- 1978 - Intel released its first 16-bit microprocessor - 8086 - executes the instructions at 2.5 MIPS (i.e. 2.5 million Instruction per second).Execution time of one instruction - 400ns (=1/MIPS=1/(2.5x10^6))
- 8086 can also address one megabytes (1MB=2^{20} bytes) of memory. Another feature in 8086 - presence of a small 6-byte instruction queue. So instructions fetched from memory are placed in it before they are executed, higher execution speed &larger memory size.

Architecture of 8086

- It is subdivided into two units - The execution unit (EU) and the bus interface unit (BIU).
- The execution unit (EU) includes: ALU, eight 16-bit general purposes registers, a 16 bit flag register, and a control unit.
- The bus interface unit (BIU) includes: adder for address calculations, four 16-bit segments registers (CS, DS, SS and ES), a 16 bit instruction pointer (IP), a 6 byte instruction queue and bus control logic.

Execution Unit (EU)

- The EU consists of eight 16-bit general purpose registers - AX, BX, CX, DX, SP, BP, SI and DI.
- AX, BX, CX and DX - can be divided into two 8-bit registers – AH and AL, BH and BL, CH and CL, DH and DL.General purpose registers can be used to store 8 bit or 16 bit data during program execution.

Special functions of registers

- AX/AL - used as accumulator which multiply, divide, input/output (I/O) and some of the decimal and ASCII adjustment instructions.
- BX - holds the offset address of a location in memory - also used to refer the data in memory using lookup table technique with the help of XLAT instruction.
- CX - used to hold the count while executing repeated string instructions (REP/REPE/REPNE) and LOOP instruction - also used to hold the count while executing the shift and rotate instructions - count value indicates the number of times the same instructions has to be executed.
- DX - used to hold a part of the result during multiplication and part of the dividend before a division - also used to hold the I/O device address while executing IN and OUT instructions.
- SP - stack pointer - used to hold the offset address of the data stored at the top of stack segment - used along with SS register to decide the address at which data is pushed or popped during the execution of PUSH and POP instructions.

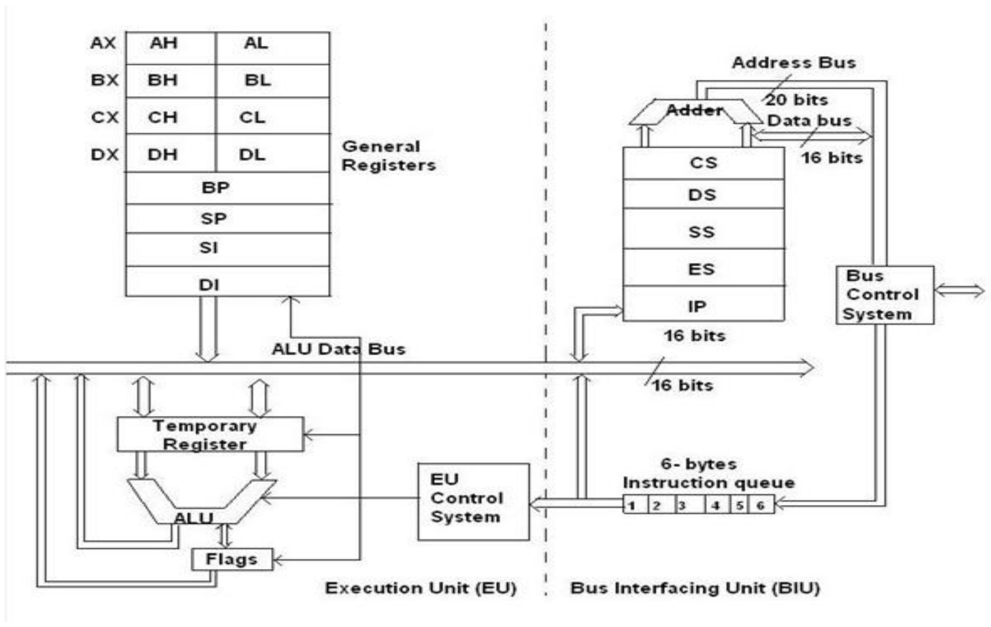


Figure 4.1: 8086 Block Diagram

- BP - Base Pointer - used to hold the offset address of the data to be read from or write into the stack segment
- SI - Source Index register - used to hold the offset address of source data in data segment while executing string instructions
- DI - Destination Index register - used to hold the offset address of destination data in extra segment while executing String instructions.
- NOTE: segment - a portion of memory where data for a program is stored - the maximum size of a segment can be 64 bytes - minimum size of a segment can be even one byte - segment begins in memory at a memory address which is divisible by 16.

Flag register of 8086

- The flags - classified into status flags and control flags
- CF, PF, AF, ZF, SF and OF are status flags. They indicate the status of the result that is obtained after the execution of arithmetic or logic instruction.
- DF, IF and TF are control flags. They can control the operation of CPU.

Function of different flags

- CF (Carry Flag) - holds the carry after 8 bit or 16 bit addition or the borrow after 8 bit or 16 bit subtraction operation.
- PF (Parity Flag) - If the lower 8 bit of the result is having odd parity (i.e. odd number of 1s) , PF is set to 0 . PF is set to 1 if the lower 8 bit of result is having even parity.

- AF (Auxiliary Carry Flag) - holds the carry after addition or the borrow after subtraction of the bits in bit position 3 (LSB is treated as bit position 0) .This is used by DAA and DAS instructions to adjust the value in AL after a BCD addition or subtraction, respectively.
- ZF (Zero Flag) - indicates that the result of an arithmetic or logic operation is zero , If Z=1. if Z=0, the result is not zero.
- SF (Sign flag) - holds the arithmetic sign of the result after an arithmetic or logic operation. If S = 0, the sign bit is 0 and the result is positive.
- TF (Trap Flag) - used to debug a program using single step technique. If T flag is set (i.e. TF = 1), 8086 gets interrupted (Trap or single step interrupt) after the execution of each instruction in the program. If TF is cleared (i.e. TF=0), the trapping or debugging feature is disabled.
- DF (Direction Flag) - selects either the increment or decrement made for the DI and/or SI register during the execution of string instructions. If D=0, registers are automatically incremented. If D=1, the register are automatically decremented. This flag can be set or cleared using the STD or CLD instruction respectively.
- IF (Interrupt Flag) - controls the operation of the 'INTR' interrupt pin of 8086 .If I=0 , INTR pin is disabled, if I=1, INTR pin is enabled. I flag can be set or cleared using the instruction STI or CLI respectively.
- OF (Overflow flag) - Signed numbers are represented in 2's complement form in microprocessor. When signed numbers are added or subtracted, overflow may occur, indicating that the result has exceeded the capacity of the machine. For example if the 8-bit signed data 7EH (= +126) is added with the 8-bit signed data 02H (= +2), the result is 80H (= -128 in 2's complement form). This result indicates an overflow condition; overflow flag is set during the above signed addition. In an 8-bit register, the minimum and maximum value of the signed number that can be stored is -128 (=80H) and +127 (=7FH) respectively. In a 16 bit register, the minimum and maximum value of the signed number that can be stored is -32768 (=8000H) and +32767 (=7FFFH) respectively. For operation an unsigned data, OF is ignored.

Bus Interface Unit (BIU)

- There are four segment registers - CS, DS, SS and ES.
- The function of the CS, DS, SS and ES register - indicate the starting address or base address of code segment, data segment, stack segment and extra segment in memory.
- The code segment contains the instructions of a program - data segment contains data for the program
- The stack segment holds the stack of a program which is needed while executing CALL and RET instructions to handle interrupts. Extra segment is the additional data segment used by some of the string instructions. Minimum size of a segment one byte, maximum size of a segment is 64 Kbytes.

- The base address can be obtained by adding four binary 0s to the right most portion of the content of corresponding segment register which is same as adding a hexadecimal digit 0 to the right most portion of a segment register.
- If the size of two different segments is less than 64Kbytes then it is possible for two segments to get overlapped (i.e. within 64 Kbytes allocated to a segment, another segment can start).
- Let a particular application in 8086 requires code segment of 1 Kbyte size and data segment of 2 Kbytes size. If the code segment is stored in memory from the address 20000H then it will end at the memory address 203FFH.
- The data segment can be stored from the address 20400H (which is the immediate next 16-byte boundary in memory).
- The CS and DS registers are loaded with the value 2000H and 2040H respectively for running this application in 8086.

Accessing memory locations

- Each address in physical memory (ROM/EPROM chips) is known as physical address. In order to access an operand (either data or instruction) from memory from a particular segment, 8086 has to first calculate the physical address of that operand.
- To find the physical address of that operand, the 8086 adds the base address of the corresponding segment with an offset address which may be either the content of a register or an 8 bit or 16 bit displacement given in the instruction or combination of both, depending upon the addressing mode used by the instruction.
- The 8086 designers have assigned certain register(s) as default offset register(s) for certain segment register. The default segment : offset registers includes CS : IP, DS : BX,SI,DI, SS : SP and BP, ES : DI.
- But this default assignment can be changed by using segment override prefix in the instruction.

Fetching of an instruction from memory

- Let us assume that the CS register is having the value 3000H and the IP register is having the value 2000H.To fetch an instruction from memory, the CPU calculates the memory address from where the next instruction is to be fetched, as shown below:
- CS X 10H =30000H + Base address of code segment
- IP = 2000H Offset address
- 32000H Memory address from where next instruction is taken by CPU.

Pin Details of 8086

- 8086 can operate in any one of the two modes namely minimum mode and maximum mode.In minimum mode, all the control signals for the memory and I/O are generated by the 8086.In maximum mode, some of the control signals must be externally generated. This requires the addition of an external bus controller such as 8288 with 8086.

- Some of the pins in 8086 have same function in both modes and some pins have different function in the two modes.

Function of pins common to minimum and maximum mode:

- AD15-AD0: These pins act as the multiplexed address and data bus of the microprocessor - Whenever ALE (Address Latch Enable) pin is high (i.e. 1), these pins carry address and when ALE pin is low (i.e. 0), these pins carry data .Using two external octal latches such as 74373 along with the ALE signal, these pins can be splitted into separate address bus (A15-A0) and data bus (D15-D0).

- A19|S6-A18|S3: These pins (address/status bus bits) are multiplexed to provide address signals A19-A16 and also status bits S6-S3 - When ALE=1, these pins carry address and when ALE=0, they carry the status lines - Using one external octal latch such as 74373 along with the ALE signal, these pins can be splitted into separate address bus (A19-A16) and status lines (S6-S3).

- NMI: Non-Maskable Interrupt (NMI) input is used to request a hardware interrupt. It cannot be disabled by software. It is a positive edge triggered interrupt and when it occurs, type2 interrupt occurs in 8086.

- INTR: Interrupt request (INTR) is a level triggered interrupt used to request a hardware interrupt. But INTR depends on the status of IF flag. When IF=1, if INTR is held high (i.e. logic 1), then 8086 gets interrupted. If IF=0, then INTR interrupt is disabled.

- CLK- The clock signal must have a duty cycle of 33% to provide proper internal timing for the 8086. Its maximum frequency can be 5 or 8 or 10 MHz for different versions of 8086 such as 8086, 8086-2 and 8086-1 respectively

- Vcc: This power supply pin provides a +5V signal to the 8086. The variation allowed in the power supply input is ±10%.

- MN/MX : The minimum/maximum mode pin is used to select either minimum mode or maximum mode operation for the 8086 by connecting this pin to either +5V directly or ground respectively.

- RD: Whenever the read signal ( ) is a logic 0, the 8086 reads data from memory or I/O device through the data bus.

- TEST: The test pin is an input that is tested by the WAIT instruction. If pin is at logic 0, then WAIT instruction functions as a NOP (No operation) instruction. If pin is at logic 1, then WAIT instruction waits for pin to become a logic 0. This pin is often connected to the BUSY input of 8087 numeric coprocessor to perform floating point operations.

- READY: This input is used to insert wait states into the timing of the 8086. If the READY pin is at logic 1 then it has no effect on the operation of the microprocessor. If the ready pin is at logic 0, the 8086 enters into wait state and remains idle. This pin is used to interface slowly operating peripherals with 8086.

- RESET: This input causes the 8086 to reset if it is held at logic 1 for a minimum of four clocking periods. Whenever the 8086 is reset, the CS and IP are initialized to FFFFH and 0000H respectively and all other registers are initialized with the value 0000H. This causes 8086 to begin executing instructions from the memory address FFFF0H.

- GND: The GND connection is the return for the power supply (Vcc). 8086 has two GND pins and both must be connected to ground for proper operation.

Function of pins used in minimum mode

- M/IO: The M/IO pin selects memory or I/O. This pin indicates whether the 8086 performs memory read or write operation (M/IO = 1) or I/O read or write operation (M/IO = 0).

- WR: The write signal indicates that 8086 is sending data to a memory or I/O device. When WR is at logic 0, the data bus contains valid data for memory or I/O.

- DT/ R: The data transmit/receive signal shows that the 8086 data bus is transmitting (DT/ R=1) or receiving (DT/R =0) data. This signal is used to control the data flow direction in external data bus buffers.

- DEN: Data bus enable signal activates external data bus buffers. When data is transferred through the data bus of 8086, this signal is at logic 0. When it is high, no data flows in the data bus.

- ALE: When Address Latch Enable (ALE) signal is high, it indicates that the 8086 multiplexed address/data bus (AD15-AD0) and multiplexed address/status bus (A19/S6-A16/S3) contain address which can be either memory address on I/O port address
- INTA: The interrupt acknowledge signal is a response to the INTR input pin. The INTA signal is used to place the interrupt type or vector number into the data bus in response to INTR interrupt.

- HOLD: The hold input requests a direct memory access (DMA) and is generated by DMA controller. If the HOLD signal is logic 1, the 8086 completes the execution the current instruction and places its address data and control bus at the high impedance state. If the HOLD pin is at logic 0, the 8086 executes the instructions normally.

- HLDA: Hold acknowledgement signal indicates that the 8086 has entered the hold state and is connected to HLDA input of DMA controller.

Function of pins used in Maximum mode

- S2, S1, S0: The status bits indicates the function of the current bus cycle. These signals are normally decoded by the 8288 (bus controller).

- LOCK: The lock output is used to lock peripherals off the system. This pin is activated by using the LOCK prefix on any instruction.
- RQ/GT0 and RQ/GT1 : The request/grant pins request DMA during maximum mode operation of 8086. These lines are both bi-directional and are used to request and grant for a DMA operation.
- QS1 and QS0: The queue status bits show the status of the internal instruction queue in 8086. These pins are provided for access by the numeric coprocessor (8087).

Re locatable program : A re locatable program is one which can be placed anywhere in the memory map of 8086 and executed without any modification in the program.

Instruction sets of 8086

Addressing modes in 8086

- Register addressing mode
- Immediate addressing mode
- Data memory addressing modes
- Program memory addressing modes
- Stack memory addressing modes

Register addressing mode

In this addressing mode, the data present in register is moved or manipulated and the result is stored in register.

Examples:

MOV AL, BL ; Move content of BL to AL

ADC BX, DX ; Add content of BX, carry flag and DX, and store result in BX

Immediate addressing mode

In this mode, the data is directly given in the instruction.

Examples

MOV AL, 50H ; Move data 50H to AL

Data memory addressing modes

The term Effective Address (EA) represents the offset address of the data within a segment which is obtained by different methods, depending upon the addressing mode that is used in the instruction.Let us assume that the various registers in 8086 have the following values stored in them.

Direct Addressing:

In this mode, the 16 bit offset address of the data within the segment is directly given in the instruction.

Examples:

MOV AL, [1000H]

EA is given within square bracket in the instruction in this addressing mode and hence EA=1000H in the above instruction. Since the destination is an 8-bit register (i.e. AL), a byte is taken from memory at the address given by DS x10H + EA=31000H and stored in AL.

Base Addressing

In this mode, the EA is the content of BX or BP register. When BX register is present in the instruction, data is taken from the data segment and if BP is present in the instruction, data is taken from the stack segment.

Examples:

MOV CL, [BX]

EA=(BX)=2000H

Memory address=DSx10+(BX)=32000H. The byte from the memory address 32000H is read and stored in CL.

Base Relative Addressing

In this mode, the EA is obtained by adding the content of the base register with an 8-bit or 16 bit displacement. The displacement is a signed number with negative values represented in two's complement form. The 16 bit displacement can have value from -32768 to +32767 and 8 bit displacement can have the value from -128 to +127.

Examples:

MOV AX, [BX+5]

EA= (BX)+5
Memory address=DSx10H+(BX)+5 = 30000H+2000H+5=32005H

The word from the memory address 32005H is taken and stored in AX.

Index Addressing

In this mode, the EA is the content of SI or DI register which is specified in the instruction. The data is taken from data segment.

Examples:

MOV BL,[SI]

EA=(SI)=1000H

Memory address=DSx10H+SI

=30000H+1000H=31000H

A byte from the memory address 31000H is taken and stored in BL.

Index relative Addressing

This mode is same as base relative addressing mode except that instead of BP or BX register, SI or DI register is used.

Example:

MOV BX, [SI-100H]

EA=(SI)-100H

Memory Address=DSx10H+(SI)-100H

= 30000H+1000H-100H =30F00H

A word from the memory address 30F00H is taken and stored in BX.

Base plus index Addressing

In this mode, the EA is obtained by adding the content of a base register and index register.

Example

MOV AX, [BX+SI]

EA = (BX)+(SI)

Memory address=DSx10H + (BX)+(SI)

=30000H+2000H+1000H=33000H

A word from the memory address 33000H is taken and stored in AX.

Base relative, index relative and base plus index addressing modes are used to access a byte or word type data from a table of data or an array of data stored in data segment one by one.

Base Relative plus index Addressing

In this mode, the EA is obtained by adding the content of a base register, an index and a displacement.

Example:

MOV CX, [BX+SI+50H]

EA= (BX) + (SI)+50H

Memory address=DSx10H+(BX)+(SI)+50H

=30000H+2000H+1000H+50H=33050H

A word from the memory address 33050H is taken and stored in CX.
Base relative plus index addressing is used to access a byte or a word in a particular record of a particular file in memory. A particular application program may process many files stored in the data segment.

Program memory addressing modes

- Program memory addressing modes are used with JMP and CALL instructions and consist of three distinct forms namely direct, relative and indirect.
- Direct Addressing: The direct program memory addressing stores both the segment and offset address where the control has to be transferred with the opcode. The above instruction is equivalent to JMP 32000H.
- When it is executed, the 16 bit offset value 2000H is loaded in IP register and the 16 bit segment value 3000H is loaded in CS. When the microprocessor calculates the memory address from where it has to fetch an instruction using the relation CSx10H+IP, the address 32000H will be obtained using the above CS and IP values.

Relative Addressing

- The term relative here means relative to the instruction pointer (IP). Relative JMP and CALL instructions contain either an 8 bit or a 16 bit signed displacement that is added to the current instruction pointer and based on the new value of IP thus obtained, the address of the next instruction to be executed is calculated using the relation CSx10H+IP.
- The 8-bit or 16 bit signed displacement which allows a forward memory reference or a reverse memory reference. A one byte displacement is used in short jump and call instructions, and a two byte displacement is used in near jump and call instructions. Both types are considered intra segment jumps since the program control is transferred anywhere within the current code segment.
- An 8 bit displacement has a jump range of between +127 and -128 bytes from the next instruction while a 16 bit displacement has a jump range of between -32768 and +32767 bytes from the next instruction following the jump instruction in the program. The opcode of relative short jump and near jump instructions are EBH and E9H respectively.
- While using assembler to develop 8086 program, the assembler directive SHORT and NEAR PTR is used to indicate short jump and near jump instruction respectively.
- Examples:

a) JMP SHORT OVER

b) JMP NEAR PTR FIND

In the above examples, OVER and FIND are the label of memory locations that are present in the same code segment in which the above instructions are present

Indirect Addressing

- The indirect jump or CALL instructions use either any 16 bit register (AX,BX,CX,DX,SP,BP,SI or DI) or any relative register ([BP],[BX],[DI] or [SI]) or any relative register with displacement. The opcode of indirect jump instruction is FFH. It can be either intersegment indirect jump or intra segment indirect jump instruction
- If a 16 bit register holds the jump address of in a indirect JMP instruction, the jump is near. If the CX register contains 2000H and JMP CX instruction present in a code segment is executed, the microprocessor jumps to offset address 2000H in the current code segment to take the next instruction for execution (This is done by loading the IP with the content of CX without changing the content of CS).
- When the instruction JMP [DI] is executed, the microprocessor first reads a word in the current data segment from the offset address specified by DI and puts that word in IP register. Now using this new value of IP, 8086 calculates the address of the memory location where it has to jump using the relation CSx10H+IP.

Stack memory addressing mode

- The stack holds data temporarily and also stores return address for procedures and interrupt service routines. The stack memory is a last-in, first-out (LIFO) memory. Data are placed into the stack using PUSH instruction and taken out from the stack using POP instruction. The CALL instruction uses the stack to hold the return address for procedures and RET instruction is used to remove return address from stack.
- The stack segment is maintained by two registers: the stack pointer (SP) and the stack segment register (SS). Always a word is entered into stack. Whenever a word of data is pushed into the stack, the higher-order 8 bits of the word are placed in the memory location specified by SP-1 (i.e. at address SSx10H + SP-1)and the lower-order 8 bits of the word are placed in the memory location specified by SP-2 in the current stack segment (SS) (i.e. at address SSx10H + SP-2). The SP is then decremented by 2. The data pushed into the stack may be either the content of a 16 bit register or segment register or 16 bit data in memory.
- Since SP gets decremented for every push operation, the stack segment is said to be growing downwards as for successive push operations, the data are stored in lower memory addresses in stack segment. Due to this, the SP is initialized with highest offset address according to the requirement, at the beginning of the program.

Segment Override Prefix:

- The segment override prefix which can be added to almost any instruction in any memory related addressing mode, allows the programmer to deviate from the default segment and offset register mechanism. The segment override prefix is an additional byte that appears the front of an instruction to select an alternate segment register. The jump and call instructions cannot be prefixed with the segment override prefix since they use only code segment register (CS) for address generation.

Example:

MOV AX, [BP] instruction accesses data within stack segment by default since BP is the offset register for stack segment. But if the programmer wants to get data from data segment using BP as offset register in the above instruction, then the instruction is modified as

MOV AX, DS: [BP]

Instruction Set of 8086

The instructions of 8086 are classified into data transfer, arithmetic, logical, flag manipulation, control transfer, shift/rotate, string and machine control instructions.

Data transfer instructions:

The data transfer instructions include MOV, PUSH, POP, XCHG, XLAT, IN, OUT, LEA, LDS, LES, LSS, LAHF and SAHF

MOV: MOV instruction copies a word or byte of data from a specified source to a specified destination. The destination can be a register or a memory location. The source can be a register or a memory location or an immediate number. The general format of MOV instruction is

MOV Destination, Source

Examples:

MOV BL, 50H; Move immediate data 50H to BL

PUSH: PUSH instruction is used to store the word in a register or a memory location into stack as explained in stack addressing modes. SP is decremented by 2 after execution of PUSH.

Examples:

PUSH CX; PUSH CX content in stack

POP: POP instruction copies the top word from the stack to a destination specified in the instruction. The destination can be a general purpose register, a segment register or a memory location. After the word is copied to the specified destination, the SP is incremented by 2 to point to the next word in the stack.

Examples:

POP BX ; Pop BX content from the stack

XCHG: The XCHG instruction exchanges the contents of a register with the contents of a memory location. It cannot exchange directly the contents of two memory locations. The source and destination must both be words or they must both be bytes. The segment registers cannot be used in this instruction.

Examples:

XCHG AL, BL; Exchanges content of AL and BL

XCHG AX, [BX]; Exchanges content of AX with content of memory at [BX]

XLAT: The XLAT instruction is used to translate a byte in AL from one code to another code. The instruction replaces a byte in the AL register with a byte in memory at [BX], which is data in a lookup table present in memory.

Before XLAT is executed, the lookup table containing the desired codes must be put in data segment and the offset address of the starting location of the lookup table is stored in BX. The code byte to be translated is put in AL. When XLAT is now executed, it adds the content of the AL with BX to find the offset address of the data in the above lookup table and the byte in that offset address is copied to AL.

IN: The IN instruction copies data from a port to AL or AX register. If an 8-bit port is read, the data is stored in AL and if a 16 bit port is read, the data is stored in AX. The IN instruction has two formats namely fixed port and variable port.

For the fixed port type IN instruction, the 8-bit address of a port is specified directly in the instruction. With this form, anyone of 256 possible ports can be addressed.

Examples:

IN AL, 80H; Input a byte from the port with address 80H to AL

IN AX, 40H; Input a word from port with address 40H to AX

For the variable port type IN instruction, the port address is loaded into DX register before the IN instruction. Since DX is a 16 bit register, the port address can be any number between 0000H and FFFFH. Hence up to 65536 ports are addressable in this mode. The following example shows a part of the program having IN instruction and the operations done when the instructions are executed are given in the corresponding comment field.

Examples:

MOV DX, 0FE50H; Initialize DX with port address of FE50H

IN AL, DX ; Input a byte from 8-bit port with port address FE50H into AL

IN AX, DX ; Input a word from 16-bit port with port address FE50H into AX

OUT: The OUT instruction transfers a byte from AL or a word from AX to the specified port. Similar to IN instruction, OUT instruction has two forms namely fixed port and variable port.

LEA: Load Effective Address

The general format of LEA instruction is

LEA register, source

This instruction determines the offset address of the variable or memory location named as the source and puts this offset address in the indicated 16 bit register.

Examples:

LEA BX, COST; Load BX with offset address of COST in data segment where

COST is the name assigned to a memory location in data segment.

LEA CX, [BX][SI]; Load CX with the value equal to (BX)+(SI) where (BX) and

(SI) represents content of BX and SI respectively.

LDS: Load register and DS with words from memory

The general form of this instruction is

LDS register, memory address of first word

The LDS instruction copies a word from the memory location specified in the instruction into the register and then copies a word from the next memory location into the DS register.

LDS is useful for initializing SI and DS registers at the start of a string before using one of the String instructions.

Example:

LDS SI,[2000H]; Copy content of memory at offset address 2000H in data segment to lower byte of SI, content of 2001H to higher byte of SI. Copy content at offset address 2002H in data segment to lower byte of DS and 2003H to higher byte of DS.

LES, LSS: LES and LSS instructions are similar to LDS instruction except that instead of DS register, ES and SS registers are loaded respectively along with the register specified in the instruction.

LAHF: This instruction copies the low byte of flag register into AH.

SAHF: Store content of AH in the low byte of flag register.

Except SAHF and POPF instructions, all other data transfer instructions do not affect flag register.

Arithmetic and Logical instructions

ADD: The general format of ADD instruction is

ADD destination, source

The data from the source and destination are added and the result is placed in the destination. The source may be an immediate number, a register or a memory location. The destination can be a register or memory location. But the source and destination cannot both be memory locations. The data from the source and destination must be of the same type either bytes or words.

Examples:

ADD BL,80H; Add immediate data 80H to BL

ADD AX,CX; Add content of AX and CX and store result in AX

ADD AL,[BX]; Add content of AL and the byte from memory at [BX] and store result in AL.

The flags AF, CF, OF, PF, SF and ZF flags are affected by the execution of ADD instruction

ADC: This instruction adds the data in source and destination along with the content of carry flag and stores the result in the destination. The general format of this instruction is

ADC destination, source

SUB: The general form of subtract (SUB) instruction is

SUB destination, source

It subtracts the number in the source from the number in the destination and stores the result in destination. The source may be an immediate number, a register or a memory location. The destination can be a register or memory location. But the source and destination cannot both be memory locations. The data from the source and destination must be of the same type either bytes or words.

For subtraction, the carry flag (CF) functions as borrow flag. If the result is negative after subtraction, CF is set, otherwise it is reset. The rules for source and destination are same as that of ADD instruction. The flags AF, CF, OF, PF, SF and ZF are affected by SUB instruction.

SBB: Subtract with Borrow

The general form of this instruction is

SBB destination, source

SBB instruction subtracts the content of source and content of carry flag from the content of destination and stores the result in destination. The rules for the source and destination are same as that of SUB instruction. AF, CF, OF, PF, SF and ZF are affected by this instruction.

INC: The increment (INC) instruction adds 1 to a specified register or to a memory location. The data incremented may be a byte or word. Carry flag is not affected by this instruction. AF, OF, PF, SF and ZF flags are affected.

Examples:

INC CL; Increment content of CL by 1

INC AX; Increment content of AX by 1

INC BYTE PTR [BX]; Increment byte in memory at [BX] by 1

INC WORD PTR [SI]; Increment word in memory at [SI] by 1

In the above examples, the term BYTE PTR and WORD PTR are assembler directives which are used to specify the type of data (byte or word respectively) to be incremented in memory.

DEC: The decrement (DEC) instruction subtracts 1 from the specified register or memory location. The data decremented may be a byte or word. CF is not affected and AF,OF, PF, SF and ZF flags are affected by this instruction.

NEG: The negate (NEG) instruction replaces the byte or word in the specified register or memory location by its 2's complement (i.e. changing the sign of the data). CF,AF, SF, PF, ZF and OF flags are affected by this instruction.

Examples:

NEG AL; Take 2's complement of the data in AL and store it in AL

NEG BYTE PTR [BX]; Take 2's complement of the byte in memory at [BX] and store result in the same place.

CMP: The general form of compare (CMP) instruction is given below:

CMP destination, source

This instruction compares a byte or word in the source with a byte or word in the destination and affects only the flags according to the result. The content of source and destination are not affected by the execution of this instruction. The comparison is done by subtracting the content of source from destination.

AF, OF, SF, ZF, PF and CF flags are affected by the instruction. The rules for source and destination are same as that of SUB instruction.

Example:

After the instruction CMP AX, DX is executed, the status of CF, ZF and SF will be as follows:

	CF	ZF	SF
If AX=DX	0	1	0
If AX>DX	0	0	0
If AX<DX	1	0	1

MUL: The multiply (MUL) instruction is used for multiplying two unsigned bytes or words. The general form of MUL instruction is

MUL Source

The source can be byte or word from a register or memory location which is considered as the multiplier. The multiplicand is taken by default from AL or AX for byte or word type data respectively. The result of multiplication is stored in AX or DX-AX (i.e. Most significant word of result in DX and least significant word of result in AX) for byte or word type data respectively. (Note: Multiplying two 8 bit data gives 16 bit result and multiplying two 16 bit data gives 32 bit result).

Examples:

MUL CH; Multiply AL and CH and store result in AX

MUL BX; Multiply AX and BX and store result in DX-AX

MUL BYTE PTR [BX]; multiply AL with the byte in memory at [B X] and store result in DX-AX

If the most significant byte of the 16 bit result is 00H or the most significant word of a 32 bit result is 0000h, both CF and OF will both be 0s. Checking these flags allows us to decide whether the leading 0s in the result have to be discarded or not. AF, PF, SF and ZF flags are undefined (i.e. some random number will be stored in these bits) after the execution of MUL instruction

IMUL: The IMUL instruction is used for multiplying signed byte or word in a register or memory location with AL or AX respectively and stores the result in AX or DX-AX respectively. If the magnitude of the result does not require all the bits of the destination, the unused bits are filled with copies of the sign bit. If the upper byte of a 16 bit result or upper word of a 32 bit result contains only copies of the sign bit (all 0s or all 1s) then CF and OF will both be 0 otherwise both will be 1. AF, PF, SF and ZF are undefined after IMUL. To multiply a signed byte by a signed word, the byte is moved into a word location and the upper byte of the word is filled with the copies of the sign bit. If the byte is moved into AL, by using the CBW (Convert Byte to Word) instruction, the sign bit in AL is extended into all the bits of AH. Thus AX contains the 16 bit sign extended word.

Examples:

IMUL BL; multiply AL with BL and store result in AX

IMUL AX; multiply AX and AX and store result in DX-AX

IMUL BYTE PTR [BX]; multiply AL with byte from memory at [BX] and store result in AX

IMUL WORD PTR [SI]; Multiply AX with word from memory at [SI] and store result in DX-AX

DIV: The divide (DIV) instruction is used for dividing unsigned data. The general form of DIV instruction is

DIV source

Where source is the divisor and it can be a byte or word in a register or memory location. The dividend is taken by default from AX and DX-AX for byte or word type data division respectively.

Examples

DIV DL; Divide word in AX by byte in DL. Quotient is stored in AL and remainder is stored in AH

DIV CX; Divide double word (32 bits) in DX-AX by word in CX. Quotient is stored in AX and remainder is stored in DX

DIV BYTE PTR [BX]; Divide word in AX by byte from memory at [BX]. Quotient is stored in AL and remainder is stored in AH.

IDIV: The IDIV instruction is used for dividing signed data. The general form and the rules for IDIV instruction are same as DIV instruction. The quotient will be a signed number and the sign of the remainder is same as the sign of the dividend.

To divide a signed byte by a signed byte, the dividend byte is put in AL and using CBW (Convert Byte to Word) instruction, the sign bit of the data in AL is extended to AH and thereby the byte in AL is converted to signed word in AX. To divide a signed word by a signed word, the dividend byte is put in AX and using CWD (Convert Word to Double word) instruction, the sign bit of the data in AX is extended to DX and thereby the word in AX is converted to signed double word in DX-AX.

If an attempt is made to divide by 0 or if the quotient is too large or two low to fit in AL or AX for 8 or 16 bit division respectively (i.e. either when the result is greater than +127 decimal in 8 bit division or +32767 decimal in 16 bit division or if the result is less than -128 decimal in 8 bit division or -32767 decimal in 16 bit division), the 8086 automatically generate a type 0 interrupt. All flags are undefined after a DIV instruction.

DAA: Decimal Adjust AL after BCD addition

This instruction is used to get the result of adding two packed BCD numbers (two decimal digits are represented in 8 bits) to be a BCD number. The result of addition must be in AL for DAA to work correctly. If the lower nibble (4 bits) in AL is greater than 9 after addition or AF flag is set by the addition then the DAA will add 6 to the lower nibble in AL. If the result in the upper nibble of AL is now greater than 9 or the carry flag is set by the addition, then the DAA will add 60H to AL.

Examples:

Let AL=0101 1000=58 BCD

CL=0011 0101=35 BCD

Consider the execution of the following instructions:

ADD AL, CL; AL=10001101=8DH and AF=0 after execution

DAA - Add 0110 (decimal 6) to AL since lower nibble in AL is greater than 9

AL=10010011= 93 BCD and CF=0

Therefore the result of addition is 93 BCD.

DAS: Decimal Adjust after BCD subtraction

DAS is used to get the result is in correct packed BCD form after subtracting two packed BCD numbers. The result of the subtraction must be in AL for DAS to work correctly. If the lower nibble in AL after a subtraction is greater than 9 or the AF was set by subtraction then the DAS will subtract 6 from the lower nibble of AL. If the result in the upper nibble is now greater than 9 or if the carry flag was set, the DAS will subtract 60H from AL.

Examples:

Let AL=86 BCD=1000 0110

CH=57 BCD=0101 0111

Consider the execution of the following instructions:

SUB AL, CH; AL=0010 1111=2FH and CF=0 after execution

DAS; Lower nibble of result is 1111, so DAS subtracts 06H from AL to make

AL=0010 1001=29 BCD and CF=0 to indicate there is no borrow.

The result is 29 BCD.

AAA:

AAA (ASCII Adjust after Addition) instruction must always follow the addition of two unpacked BCD operands in AL. When AAA is executed, the content of AL is changed to a valid unpacked BCD number and clears the top 4 bits of AL. The CF is set and AH is incremented if a decimal carry out from AL is generated.

Example:

Let AL=05 decimal=0000 0101

BH=06 decimal=0000 0100

AH=00H

Consider the execution of the following instructions:

```
ADD AL, BH       ; AL=0BH=11 decimal and CF=0

AAA              ; AL=01 and AH=01 and CF=1
```

Since 05+06=11(decimal)=0101 H stored in AX in unpacked BCD form. When this result is to be sent to the printer, the ASCII code of each decimal digit is easily formed by adding 30H to each byte.

AAS: ASCII Adjust after Subtraction

This instruction always follows the subtraction of one unpacked BCD operand from another unpacked BCD operand in AL. It changes the content of AL to a valid unpacked BCD number and clears the top 4 bits of AL. The CF is set and AH is decremented if a decimal carry occurred.

Example:

Let AL=09 BCD=0000 1001

CL=05 BCD =0000 0101

AH=00H

Consider the execution of the following instructions:

SUB AL, CL; AL=04 BCD

AAS ; AL=04 BCD and CF=0

; AH=00H

AAA and AAS affect AF and CF flags and OF, PF, SF and ZF are left undefined. Another salient feature of the above two instructions are that the input data used in the addition or subtraction can be even in ASCII form of the unpacked decimal number and still we get the result in ordinary unpacked decimal number form and by adding 30H to the result , again we get ASCII form of the result.

AAD: The ASCII adjust AX before Division instruction modifies the dividend in AH and AL, to prepare for the division of two valid unpacked BCD operands. After the execution of AAD, AH will be cleared and AL will contain the binary equivalent of the original unpacked two digit numbers. Initially AH contains the most significant unpacked digit and AL contains the least significant unpacked digit.

Example: To perform the operation 32 decimal / 08 decimal

Let AH=03H; upper decimal digit in the dividend

AL=02H; lower decimal digit in the dividend

CL=08H; divisor

Consider the execution of the following instructions:

AAD; AX=0020H (binary equivalent of 32 decimal in 16 bit form)

DIV CL; Divide AX by CL; AL will contain the quotient and AH will contain the remainder.

AAD affects PF, SF and ZF flags. AF, CF and OF are undefined after execution of AAD.

AAM: The ASCII Adjust AX after Multiplication instruction corrects the value of a multiplication of two valid unpacked decimal numbers. The higher order digit is placed in AH and the low order digit in AL.

Example:

Let AL=05 decimal

CL=09 decimal

Consider the execution of the following instructions:

MUL CH; AX=002DH=45 decimal

AAM; AH=04 and AL=05 (unpacked BCD form decimal number of 45)

OR AX, 3030H; To get ASCII code of the result in AH and AL (Note: this instruction is used only when it is needed). AAM affects flags same as that of AAD.

AND: The AND instruction perform logical AND operation between the corresponding bits in the source and destination and stores the result in the destination. Both the data can be either bytes or words. The general form of AND instruction is

AND destination, Source

The rules for destination and source for AND instruction are same as that of ADD instruction. CF and OF are both 0 after AND. PF, SF and ZF are updated after execution of AND instruction. AF is undefined. PF has meaning only for ANDing 8-bit operand.

OR: The OR instruction perform logical OR operation between the corresponding bits in the source and destination and stores the result in the destination. Both the data can be either bytes or words. The general form of OR instruction is

OR destination, Source

The rules for the source and destination and the way flags are affected for OR instruction are same as that of AND instruction.

XOR: The XOR instruction performs logical XOR operation between the corresponding bits in the source and destination and stores the result in the destination. Both the data can be either bytes or words. The general form of XOR instruction is

XOR destination, source

The rules for the same source and destination and the way flags are affected for XOR instruction are same as that of AND instruction.

NOT: The Not instruction inverts each bit (forms the 1's complement) of the byte or word at the specified destination. The destination can be a register or a memory location. No flags are affected by the NOT instruction.

Example:

NOT AL; Take 1's complement of AL

NOT BX; Take 1's complement of BX

NOT [SI]; Take 1’s complement of data in memory at [SI]

TEST: This instruction ANDs the content of a source byte or word with the content of the specified destination byte or word respectively. Flags are updated, but neither operand is changed. The TEST instruction is often used to set flags before a conditional jump instruction. The general form of TEST instruction is

TEST destination, source

The rules for the source and destination are same as that of AND instruction and the way flag are affected is also same as that of AND instruction.

Example:

Let AL=0111 1111 =7FH

TEST AL, 80H; AL=7FH (unchanged)

ZF=1 since (AL) AND (80H)=00H; SF=0; PF=1

Flag manipulations instructions

• **Mnemonics**	• **Function**
• LAHF	• Load low byte of flag register into AH
• SAHF	• Store AH into the low byte of flag register
• PUSHF	• Push flag register’s content into stack
• POPF	• Pop top word of stack into flag register
• CMC	• Complement carry flag (CF = complement of CF)
• CLC	• Clear carry flag (CF= 0)
• STC	• Set carry flag (CF= 1)
• CLD	• Clear direction flag (DF= 0)
• STD	• Set direction flag (DF= 1)
• CLI	• Clear interrupt flag (IF= 0)
• STI	• Set interrupt flag (IF=1)

Control transfer:

Mnemonics	Descripton
JMP addr	Jump unconditionally to addr
CALL addr	Call procedure or subroutine starting at addr
RET	Return from procedure or subroutine
JA addr	Jump if above to addr (jump if CF = ZF =0)
JAE addr	Jump if above or equal to addr (jump if CF=0)
JB addr	Jump if below to addr (jump if CF=1)
JBE addr	Jump if below or equal to addr (Jump if CF =1 or ZF = 1)
JC addr	jump if carry to addr (jump if CF = 1)
JCXZ addr	Jump if CX = 0
JE addr	Jump if equal (jump if ZF = 1)

Mnemonics	Descripton
JMP addr	Jump unconditionally to addr
CALL addr	Call procedure or subroutine starting at addr
RET	Return from procedure or subroutine
JA addr	Jump if above to addr (jump if CF = ZF =0)
JAE addr	Jump if above or equal to addr (jump if CF=0)
JB addr	Jump if below to addr (jump if CF=1)
JBE addr	Jump if below or equal to addr (Jump if CF =1 or ZF = 1)
JC addr	jump if carry to addr (jump if CF = 1)

JCXZ addr	Jump if CX = 0
JE addr	Jump if equal (jump if ZF = 1)

JL addr	**Jump if not less (Jump if SF=OF)**
JNLE addr	Jump if not less or equal (Jump if ZF = 0 and SF = OF)
JNO addr	Jump if not overflow (Jump if OF = 0)
JNP addr	Jump if not parity (Jump if PF = 0)
JNS addr	Jump if not sign (jump if SF=0)
JNZ addr	Jump if not zero (jump if ZF=0)
JO addr	Jump if overflow (jump if OF=1)
JP addr	Jump if parity (jump if PF=1)
JPE addr	Jump if parity even (jump if PF=1)
JPO addr	Jump if parity odd (jump if PF=0)
JS addr	Jump if sign (jump if SF=1)
JZ addr	Jump if zero (jump if ZF=1)

Conditional jump instructions

Shift and Rotate instructions: The Shift instructions perform logical left shift and right shift, and arithmetic left shift and right shift operation. The arithmetic left shift (SAL) and logical left shift (SHL) have the same function.

SAL/SHL: The general format of SAL/SHL instruction is

SAL/SHL Destination, Count

The destination can be a register or a memory location and it can be a byte or a word. This instruction shifts each bit in the specified destination some number of bit positions to the left. As a bit is shifted out of the LSB position, a 0 is put in the LSB position. The MSB will be shifted into carry flag (CF).

If the number of shifts to be done is 1 then it can be directly specified in the instruction with count equal to 1. For shifts of more than one bit position, the desired number of shifts is loaded into the CL register and CL is put in the count position of the instruction. CF, SF and ZF are affected according to the result. PF has meaning only when AL is used as destination. SAL instruction can be used to multiply an unsigned number by a power of 2. Doing one bit or two bits left shift of a number multiplies the number by 2 or 4 respectively and so on.

Examples:

SAL AX,1; Shift left the content of AX by 1 bit

SAL BL,1; Shift left the content of BL by 1 bit

SAL BYTE PTR [SI],1; Shift left the byte content of memory at [SI] by 1 bit

SAR: The general format of SAR instruction is

SAR Destination, Count

The destination can be a register or a memory location and it can be a byte or a word. This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position (i.e. the sign bit is copied into the MSB). The LSB will be shifted into carry flag (CF).

The rules for the Count in the instruction are same as that of the SAL instruction. CF, SF and ZF are affected according to the result. PF has meaning only when AL is used as destination.

SHR:

The general format of SHR instruction is

SHR Destination, Count

The destination can be a register or a memory location and it can be a byte or a word. This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a 0 is put in the MSB position. The LSB will be shifted into carry flag (CF).

The rules for the Count in the instruction are same as that of the SHL instruction. CF, SF and ZF are affected according to the result. PF has meaning only when 8 bit destination is used.

ROR: This instruction rotates all the bits of the specified byte or word by some number of bit positions to the right. The operation done when ROR is executed is shown below.

The general format of ROR instruction is

ROR Destination, Count

The data bit moved out of the LSB is also copied into CF. ROR affects only CF and OF. For single bit rotate, OF will be 1 after ROR, if the MSB is changed by the rotate. ROR is used to swap nibbles in a byte or to swap two bytes within a word. It can also be used to rotate a bit in a byte or word into CF, where it can be checked and acted upon by the JC and JNC instruction. CF will contain the bit most recently rotated out of LSB in the case of multiple bit rotate.

The rules for count are same as that of the shift instruction which is discussed above.

Examples:

ROR CH, 1 : rotate right byte in CH by one bit position.

ROR BX, CL : Rotate right word in BX by number of bit positions given by CL.

ROL: ROL rotates all the bits in a byte or word in the destination to the left either by 1 bit position and more than 1 bit positions using CL as shown below:

RCR: Rotate the byte or word in the destination right through carry flag (CF) either by one bit position or the number of bit positions given by the CL as shown below.The flag are affected by similar to ROR.

RCL: Rotate the byte or word in the destination left through carry flag (CF) either by one bit position or the number of bit positions given by the CL as shown below.The flags are affected similar to ROL.

String Instructions

The string instructions operate on element of strings of bytes or word. Registers SI and DI contain the offset address within a segment of an element (Byte or Word) in the source string and the destination string respectively. The source string is in the data segment at the offset address given by SI and destination string is in the extra segment at the offset address given by DI.After each string operation, SI and/or DI are automatically incremented or decremented by 1 or 2 (for byte or word operation) according to the D flag in the flag register. If D=0, SI and/or DI are automatically incremented and if D=1, SI and/or DI are automatically decremented.

MNEMONICS	**FUNCTION**
MOVSB	Move string byte from DS:[SI] to ES:[DI]
MOVSW	Move string word from DS:[SI] to ES:[DI]
CMPSB	Compare string byte (Done by subtracting byte at ES:[DI] from the byte at DS:[SI affected and the content of bytes compared is unaffected.
CMPSW	Compare string word (Done by subtracting word at ES:[DI] from the word at DS:[SI affected and the content of words compared is unaffected.
LODSB	Load string byte at DS:[SI] into AL
LODSW	Load string word at DS:[SI] into AX
STOSB	Store string byte in AL at ES:[DI]

STOSW	Store string word in AX at ES:[DI]
SCASB	Compare string byte (Done by subtracting byte at ES:[DI] from the byte at). Only flags the content of bytes compared is unaffected.
SCASW	Compare string word (Done by subtracting word at ES:[DI] from the byte at AX). Only and the content of words compared is unaffected.
REP	Decrement CX and Repeat the following string operation if CX ≠ 0.
REPE or REPZ	Decrement CX and Repeat the following string operation if CX ≠ 0 and ZF=1.
REPNE or REPNZ	Decrement CX and Repeat the following string operation if CX ≠ 0 and ZF=0.

Machine or processor control instructions

HLT: The halt instruction stops the execution of all instructions and places the processor in a halt state. An interrupt or a reset signal will cause the processor to resume execution from the halt state.

LOCK: The lock instruction asserts for the processor an exclusive hold on the use of the system bus. It activates an external locking signal ( ) of the processor and it is placed as a prefix in front of the instruction for which a lock is to be asserted. The lock will function only with the XCHG, ADD, OR, ADC, SBB, AND, SUB, XOR, NOT, NEG, INC and DEC instructions, when they involve a memory operand. An undefined opcode trap interrupt will be generated if a LOCK prefix is used with any instruction not listed above.

NOP: No operation. This instruction is used to insert delay in software delay programs.

ESC: This instruction is used to pass instructions to a coprocessor such as the 8087, which shares the address and data bus with an 8086. Instructions for the coprocessor are represented by a 6- bit code embedded in the escape instruction.

As the 8086 fetches instruction bytes from memory the coprocessor also catches these bytes from the data bus and puts them in a queue. However the coprocessor treats all the normal 8086 instructions as NOP instruction. When the 8086 fetches an ESC instruction, the coprocessor decodes the instruction and carries out the action specified by the 6- bit code specified in the instruction.

WAIT: When this instruction is executed, the 8086 checks the status of its input pin and if the input is high, it enters an idle condition in which it does not do any processing. The 8086 will remain in this state until the 8086's input pin is made low or an interrupt signal is received on the INTR or NMI pins. If a valid interrupt occurs while the 8086 is in this idle state, it will return to the idle state after the interrupt service routine is executed. WAIT instruction does not affect flags. It is used to synchronize 8086 with external hardware such as 8087 coprocessor.

8086 Assembly Language Programming:

Intel 8086 Interrupts

An 8086 interrupt can come from any one of the following three sources:

a) An external signal applied to the non-maskable interrupt (NMI) pin or to the interrupt (INTR) pin. An interrupt caused by a signal applied to one of these inputs is called *hardware interrupt*.

b) The execution of the instruction INT n*,* where n is the interrupt type that can take any value between 00H and FFH. This is called *software interrupt*.

c) An error condition such as divide-by-0*,* which is produced in the 8086 by the execution of the DIV/IDIV instruction or the trap interrupt.

8086 Interrupt Processing: If an interrupt has been requested, the 8086 processes it by performing the following series of steps:

a) Pushes the content of the flag register onto the stack to preserve the status of the interrupt (IF) and trap flags (TF), by decrementing the stack pointer (SP) by 2

b) Disables the INTR interrupt by clearing IF in the flag register

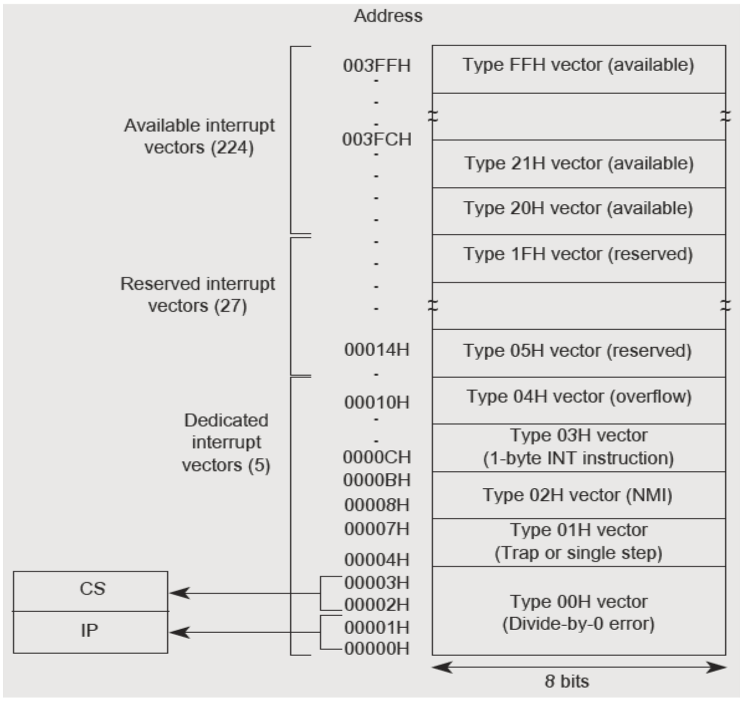
c) Resets TF in the flag register, to disable the single step or trap interrupt

d) Pushes the content of the code segment (CS) register onto the stack by decrementing SP by 2

e) Pushes the content of the instruction pointer (IP) onto the stack by decrementing SP by 2

f) Performs an indirect far jump to the start of the interrupt service routine (ISR) corresponding to the received interrupt

Interrupt vector table


Address
Available interrupt vectors (224)
Reserved interrupt vectors (27)
Dedicated interrupt vectors (5)
003FFH
003FCH
00014H
00010H
0000CH
0000BH
00008H
00007H
00004H
00003H
00002H
00001H
00000H
Type FFH vector (available)
Type 21H vector (available)
Type 20H vector (available)
Type 1FH vector (reserved)
Type 05H vector (reserved)
Type 04H vector (overflow)
Type 03H vector (1-byte INT instruction)
Type 02H vector (NMI)
Type 01H vector (Trap or single step)
Type 00H vector (Divide-by-0 error)
CS
IP
8 bits

Interrupt types in 8086

The lowest five interrupt types in the 8086 (i.e., types 00H–04H) are dedicated to

a) Type 00H or divide-by-0 interrupt

b) Type 01H or the single step (trap) interrupt

c) Type 02H or the NMI interrupt

d) Type 03H or the one-byte INT instruction interrupt

e) Type 04H or the overflow interrupt

Software Interrupts

- The INT instruction of the 8086 can be used to generate any one of the 256 possible interrupt types, which are called *software interrupts*. The desired interrupt type is specified as part of the INT instruction. e.g., the INT 21H instruction causes the 8086 to generate an interrupt of the type 21H.
- The IRET instruction at the end of the ISR makes the 8086 return to the main program to the instruction next to the INT n instruction, to continue the execution of the main program.

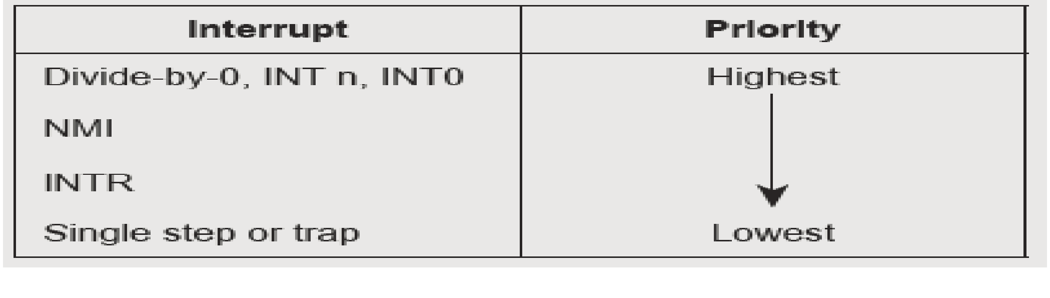
Software Interrupts – uses

Software interrupts produced by the INT instruction have the following uses:

a) Inserting break points in a program for debugging. The INT 03H instructions is used for this purpose.

b) Testing the function correctness of various ISRs. For example, the INT 02H instruction can be used to test the ISR for the NMI interrupt, without giving any input signal to the NMI pin of the 8086.

8086 Interrupt Priorities



Interrupt	Priority
Divide-by-0, INT n, INT0	Highest
NMI	↓
INTR	↓
Single step or trap	Lowest

Summary

- An interrupt is an external or internal event in a microprocessor that diverts it from the execution of the main program, to another program called the interrupt service routine (ISR).The interrupt can be either a hardware interrupt or a software interrupt. The 8086 has two hardware interrupts—NMI and INTR. The software interrupt is created in the 8086 using the INT instruction.
- There are 256 interrupt types available in the 8086 and the interrupt vector for each type, which is four bytes long, is stored in an interrupt vector table (IVT) from address 00000H in the memory.Whenever an interrupt is received, the 8086 saves the current value of IP, CS, and the flag register in the stack, clears TF and IF, and loads CS and IP with the interrupt vector corresponding to the received interrupt type. This causes the 8086 to start the execution of the ISR.
- The IRET instruction at the end of the ISR makes the 8086 return to the main program.There exist different levels of priority among the interrupts, and if two interrupts appear simultaneously in the 8086, the interrupt having higher priority is serviced first.
- BIOS function calls (also called BIOS interrupts) are stored in the system ROM and the video BIOS ROM present in the PC. These BIOS function calls directly control the I/O devices with/without the DOS (disk operating system) loaded in the system.

Architecture of 80386

80386 was Intel's first 32-bit microprocessor that contained 32-bit data bus and 32-bit address bus. Through the 32-bit address bus, the 80386 addresses upto 4G bytes (= 2^{32} bytes) of memory. The 32 bit data bus allows to read or write single precision floating number (32 bits) from or into memory in a single memory read or write cycle. This increases the speed of execution of any program that manipulates real numbers in 80386. Most high level language programs and database management systems use real numbers for data storage.

The internal architecture of 80386 (Figure 4.2) is divided to three units namely bus interface unit, memory management unit and central processing unit. The central processing unit is further divided into execution unit and instruction unit. The execution unit has eight general purpose and eight special purpose registers which are either used for handling data or calculation of the offset addresses. The instruction unit decodes the opcode bytes received from the 16-byte instruction queue and arranges them in a three decoded-instruction queue so as to pass it to the control section for deriving the necessary control signals. The barrel shifter increases the speed of execution of shift and rotates instructions. 32-bit multiplication can be executed within one microsecond by the multiply/divide logic.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. The segmentation unit allows the use of two address components, namely segment and offset for relocability and sharing of certain code and data by many programs. The maximum size of a segment is 4 Gbytes. The paging unit organizes the physical memory in terms of pages of 4Kbytes each. The paging unit works under the control of segmentation unit (i.e each segment is divided into pages). The virtual memory is also organized in terms of segments and pages by the MMU. The 80386 requires a single +5V power supply for its operation. The clock frequency used in different versions of 80386 is 16 MHz, 20 MHz, 25 MHz and 33 MHz.

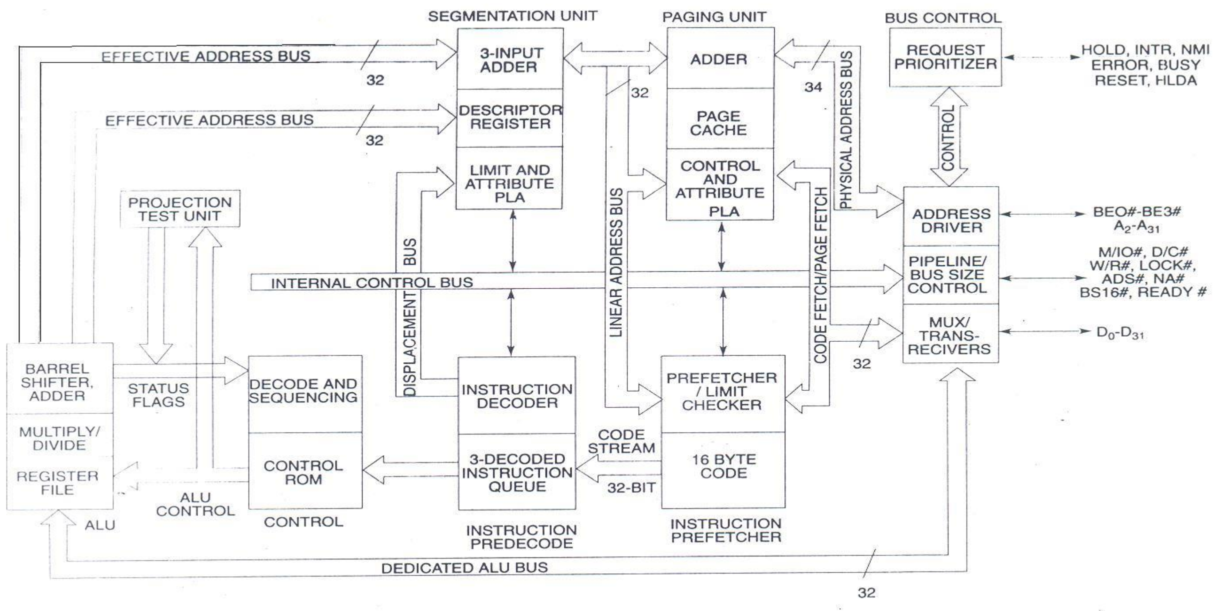


Figure 4.2: 80386 Architecture

Register organization of 80386

All the registers (Figure 4.3) in 80386 are 32-bits. The 32-bit register, known as an extended register is represented by the register name with prefix E. But 16-bit registers such as AX, BX, CX, etc. and 8-bit registers such as AH,AL,BH, etc. are also available in 80386 as in 8086. There are two additional segment registers such as FS and GS, which provides two additional segments which can be accessed by a program. The 80386 includes a memory management unit (MMU) that allows memory resources to be allocated and managed by the operating system. The segment descriptor registers are not available for the programmer rather they are internally used to store the segment descriptor information like base address, limit and attributes of different segments.

These registers are automatically loaded when the corresponding segment registers are loaded with new selectors. GDTR, IDTR, LDTR and TR are used to access the descriptor tables namely GDT, IDT, LDT and TSS descriptor respectively. The registers DR0 to DR3 (Figure 4.5) are used to store four program controllable breakpoint addresses at which execution of a program breaks, which is useful to debug a program using breakpoint technique easily. DR6 and DR7 hold break point status and break point control information respectively. The control registers (Figure 4.4) CR1 is reserved for use in future Intel processor. The bits PE and PG (bits 0 and 31) in CR0 are used to enable protected mode operation and paging respectively.CR3 is used to hold the base address of page directory in memory. CR2 is used to hold the linear address for which page fault (required page being not present in physical memory) has occurred and using this address the operating system can load the required page in physical memory from the secondary memory.

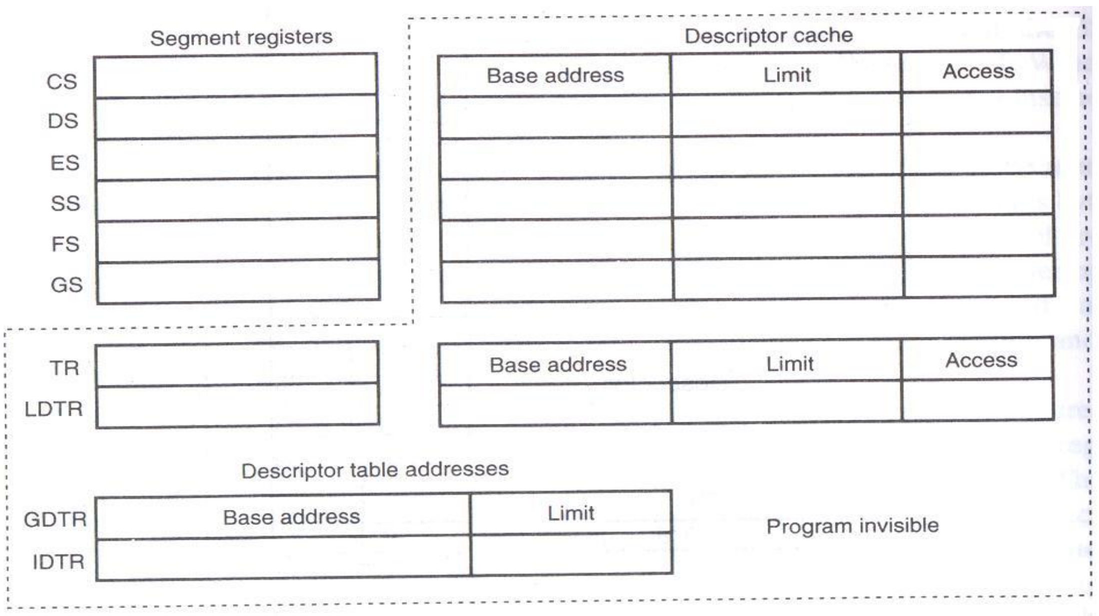


Figure 4.3: Segment registers and MMU registers

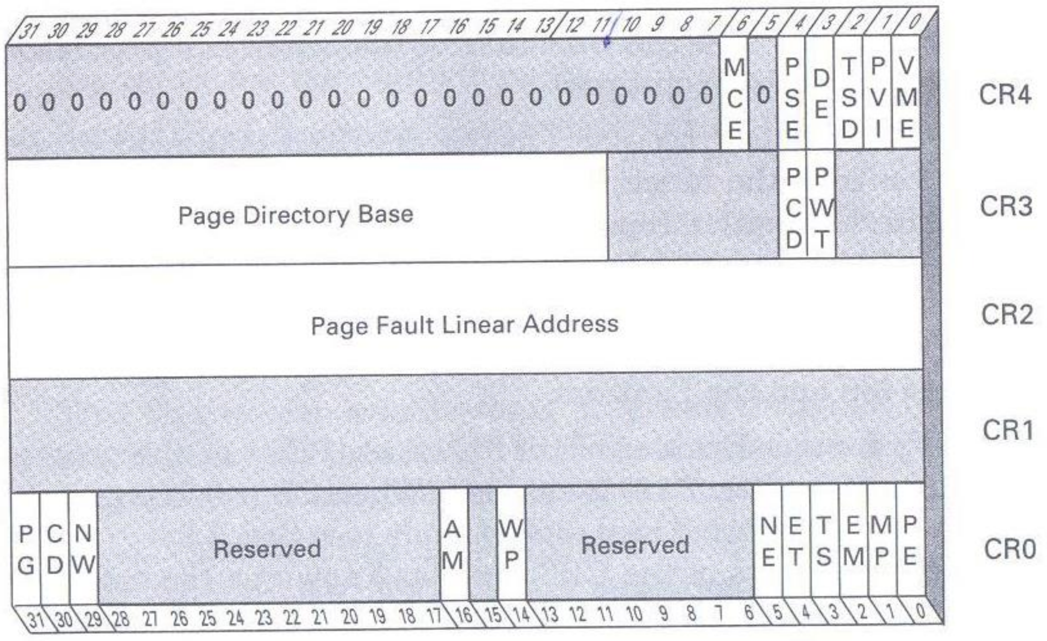


Figure 4.4: Control registers in 80386 to Pentium

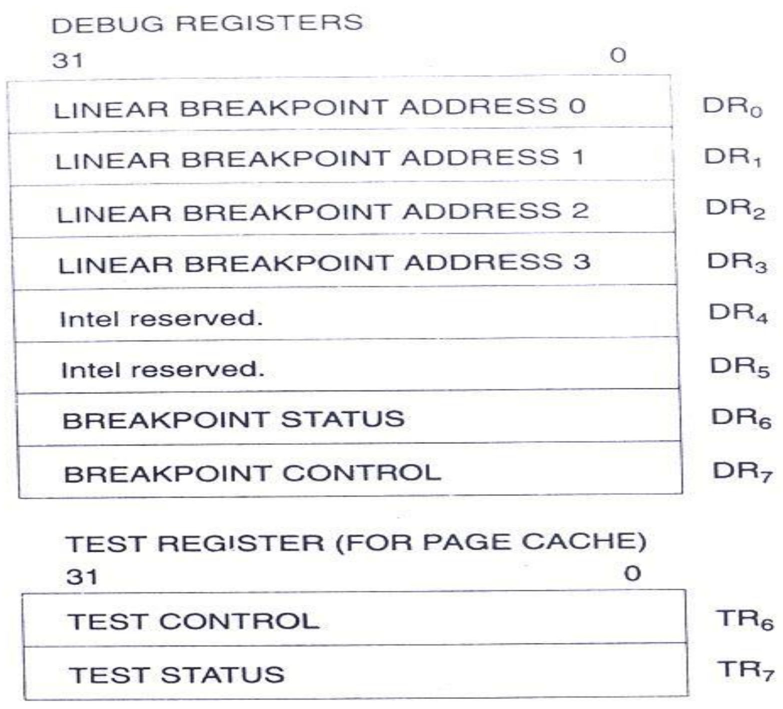
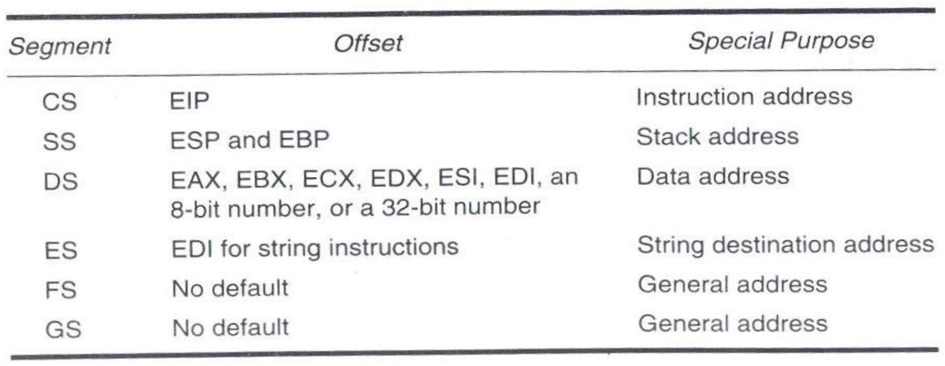


Figure 4.5: Debug registers and Test registers in 80386 to Pentium



Segment	Offset	Special Purpose
CS	EIP	Instruction address
SS	ESP and EBP	Stack address
DS	EAX, EBX, ECX, EDX, ESI, EDI, an 8-bit number, or a 32-bit number	Data address
ES	EDI for string instructions	String destination address
FS	No default	General address
GS	No default	General address

Figure 4.6: Segment registers and their default offset registers in 80386 to Pentium processors

Instruction set of 80386

The instruction set of the 80386 is upward compatible with the earlier 8086 and 80286 processors. The memory management instructions and techniques used by the 80386 are also compatible with the 80286. These features allowed 16-bit software which are written for 8086 and 80286, to be executed in 80386 also. There are few additional instructions included in 80386 which references the 32–bit registers and manages the memory system: BSF, BSR, BT, BTR, BTS, CDQ, CWDE, LFS, LGS, LSS, MOVSX, MOVZX, SET cc, SHLD, SHRD.

Addressing memory by 80386 in protected mode

The base address is 32 bits (B31-B0) and limit is 20 bits (L19-L0). If the granularity (G) bit is 1, then the limit field is multiplied by 4K to get the size of the segment in bytes and if G is 0 then the limit field itself gives the size of the segment in bytes. The available (AV) bit indicates whether the segment is available (AV=1) or not available (AV=0). The D bit indicates how the 80386 to Pentium 4 access register and memory data in the protected and real mode. If D is 0, the instructions are 16-bit instructions, compatible with 8086-80286 and these instructions use 16-bit registers and 16-bit offsets by default. If D is 1, the instructions are 32-bit instructions, compatible with 80386 and these instructions use 32-bit registers and 32-bit offsets by default.

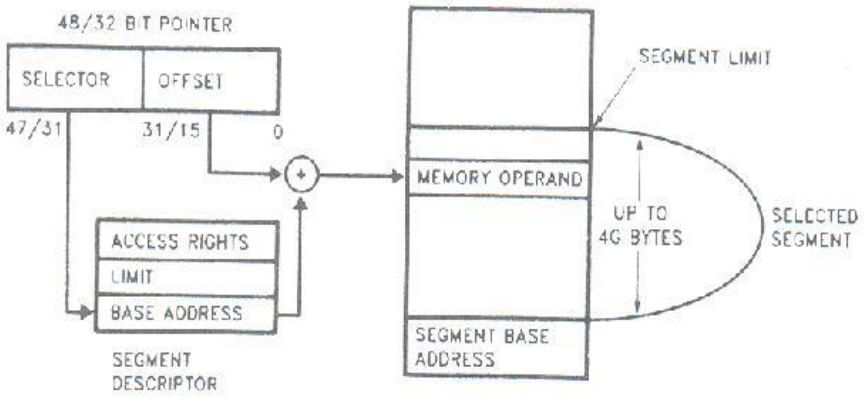


Figure 4.7: Addressing of memory when the 80386 – Pentium operates in protected mode

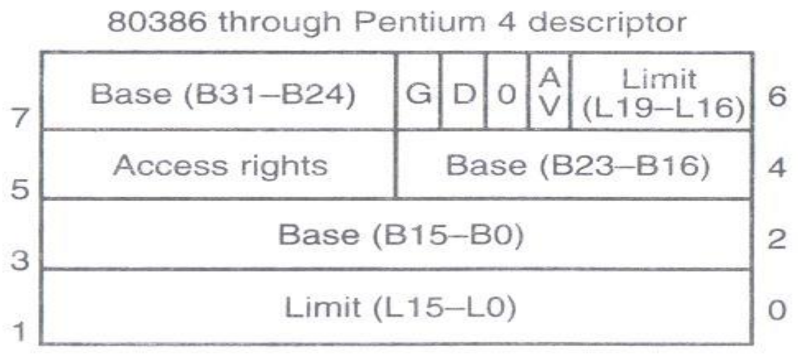


Figure 4.8: Format of segment descriptor in 80386 to Pentium

Accessing memory by different addressing modes in protected mode

The From 80386 onwards, Intel introduced a new addressing mode called scaled index addressing mode in which the content in index register can be multiplied by a factor of 1 or 2 or 4 or 8 while calculating the effective address (EA).

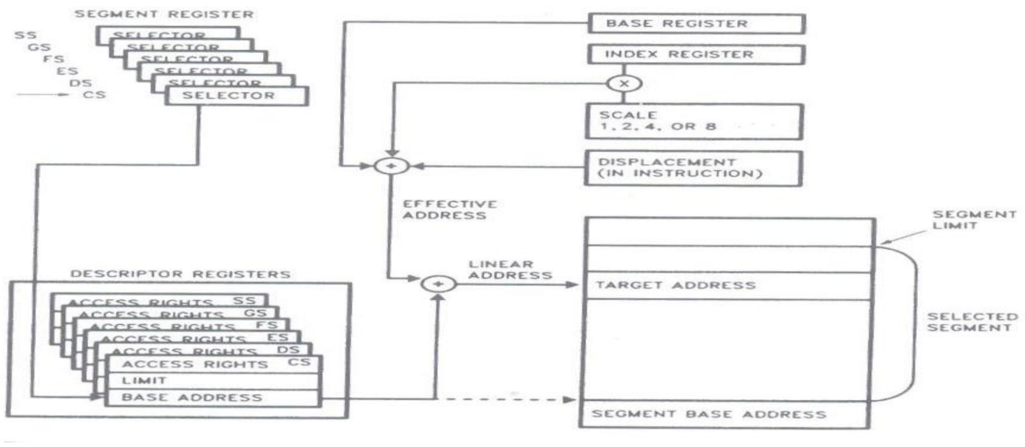


Figure 4.9: Accessing memory by different addressing modes in protected mode by 80386 to Pentium

Physical memory organization of 80386

The physical memory system of 80386 is 4G bytes. If virtual addressing is used using LDT and GDT, 64 Tera bytes (214 segment descriptors X 232 bytes/segment = 26 x 240 bytes) of virtual memory are mapped into the 4G bytes of the physical memory by the memory management unit and descriptors. The physical memory is divided into four 8-bit wide memory banks, each containing up to 1Gbyte of memory .This 32 bit wide memory organization allows bytes, words or double words of memory data to be accessed directly. The physical memory address ranges from 00000000H to FFFFFFFFH. The physical memory location with address 00000000H is in bank0, with address 00000001H is in bank1, with address 00000002H is in bank2, and with address 00000003H is in bank3, etc.

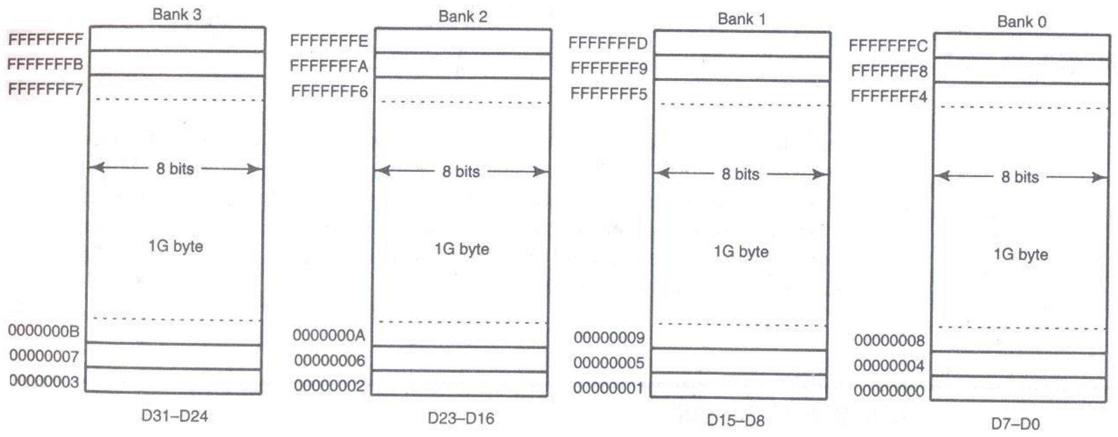


Figure 4.10:Physical memory system of 80386

Paging mechanism in 80386

The paging mechanism (Figure 4.11) provides an efficient way of handling virtual memory. The paging can be enabled or disabled be setting or clearing the PG bit in control register CR0 respectively. When paging is enabled, each segment is divided into fixed size pages of 4 Kbytes each and the address generated by the segmentation mechanism is known as linear address (Figure 4.12). The information about each page is stored in a page table in the form of a 4-byte entry known as PTE (Page Table Entry) Each page table can store a maximum of one thousand and twenty four (=210) PTEs and hence the size of a page table is 4 Kbytes. There can be a maximum of one thousand and twenty four (=210) page tables. The information about each page table is stored in a page directory in the form of a 4-byte entry known as PDE (Page Directory Entry). There is only one page directory and it can store a maximum of one thousand and twenty four (=210) PDEs and hence the size of a page directory is also 4 Kbytes.

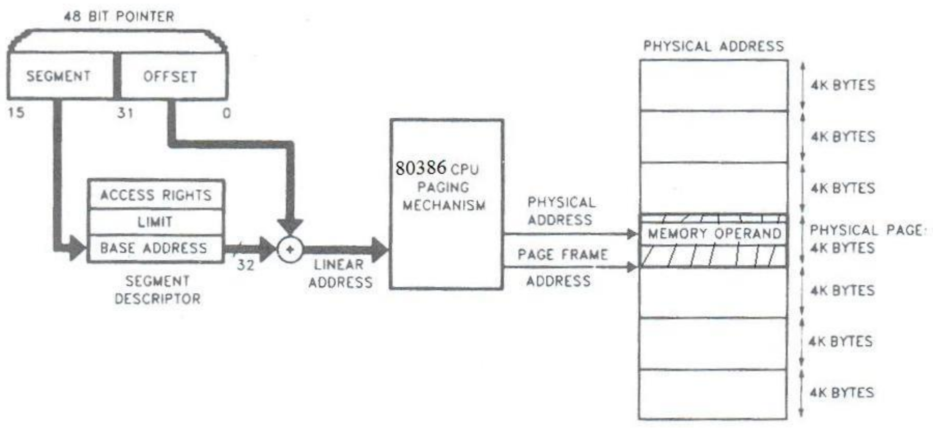


Figure 4.11: Paging mechanism of 80386

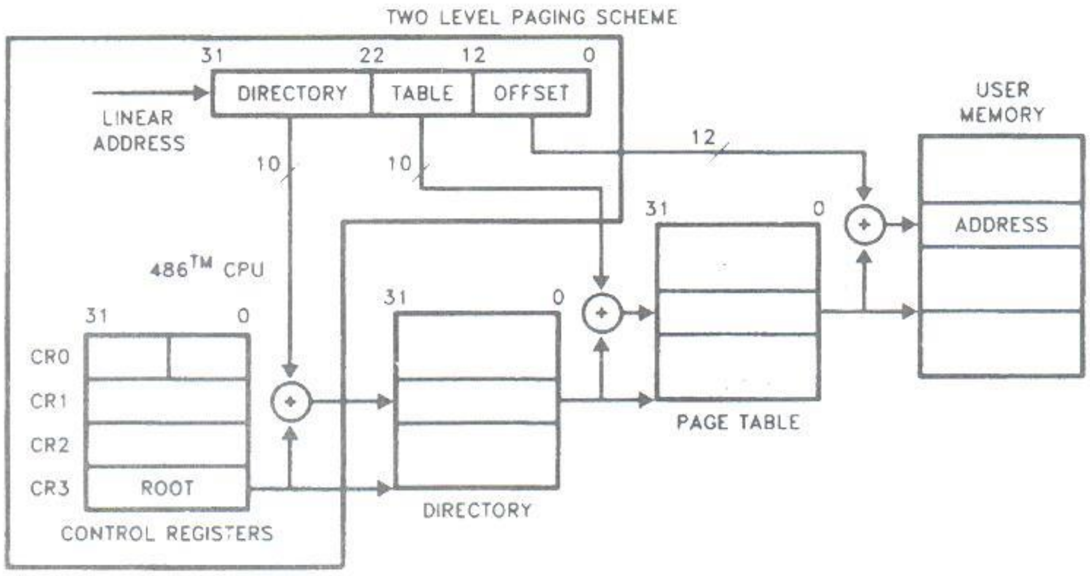


Figure 4.12: Conversion of linear address to physical address

80486 microprocessor

- The bus interface, connected to the external system bus and to the on-chip cache and prefetcher unit.
- The prefetcher, which includes a 32-byte queue of prefetched instructions and is connected to the bus interface, cache, instruction decoder and segmentation unit.
- The cache unit which includes an 8 Kbyte cache, storing both code and data, and cache management logic.
- It is connected to through a 64-bit interunit transfer (data) bus to the segmentation unit, ALU and FPU.
- The cache unit is also directly connected to the paging unit, bus interface and prefetcher through 128 lines, permitting the prefetching 16 bytes of instructions simultaneously.
- The cache is four way set-associative, write through, with 16 bytes/line.
- In write-through policy of cache, during a write operation when there is a hit, both the cache and main memory are updated together.
- The instruction decode unit, which receives three bytes of undecoded instructions from the prefetcher queue and transmits decoded instructions to the control and protection test unit.
- The control and protection test unit which generates micro instructions transmitted to other units and performs protection testing.
- The ALU, which includes general purpose register file, a barrel shifter, and registers for microcode use.
- The FPU which includes floating-point registers, an adder, a multiplier, and a shifter.
- The segmentation unit, which includes segmentation management logic, descriptor registers and break point logic.
- The paging unit, which includes paging management logic and a 32-entry TLB.

Pentium microprocessor

The Pentium (P5 or 80586) is the third among Intel's 32 bit microprocessors, released in the year 1993. It has the distinction of being the first complex Instruction set computer (CISC) type processor implementing instruction level parallelism (ILP). It is a two-issue superscalar processor, which means that two instructions can be simultaneously decoded and executed in it. The Pentium is manufactured using 0.8 micron Bipolar complementary metal-oxide-semiconductor (BiCMOS) technology and is available in a 273-pin grid array package. There are 3.1 million transistors inside the Pentium.

- The Pentium has a five stage integer pipeline (Figure 4.14), branching out into two paths U and V in the last three stages. The Pentium pipeline stages are as follows:
- PF- Prefetch: The CPU prefetches the code from the code cache and aligns the code to the initial byte of the next instruction to be decoded.
- D1- First decode: The CPU decodes the instruction to generate a control word. A single control word causes direct execution of an instruction. More complex instructions require micro coded control sequencing.
- D2- Second decode: The CPU decodes the control word, generated in stage D1, for subsequent use in the next execution (E) stage. In addition, addresses for data memory references are generated.

- E- Execute: The instruction is executed in the ALU. The barrel shifter or other operational units are used if necessary. The data cache is accessed at this stage if necessary.
- WB- Write back: The CPU stores the results and updates the flags at this stage.

Block diagram of Pentium

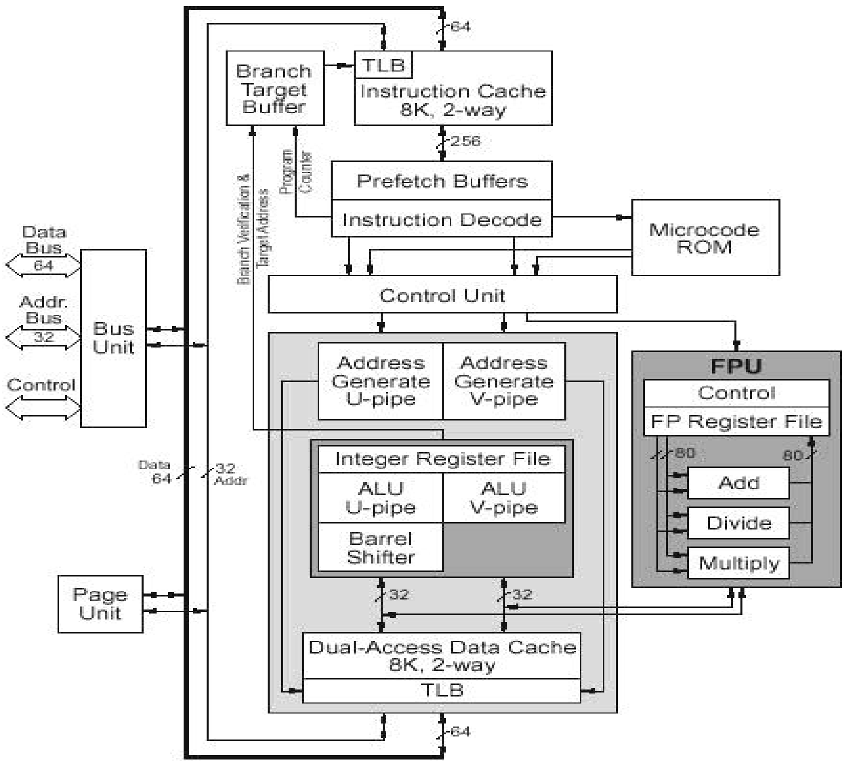


Figure 4.13: Pentium Architecture

The block diagram of the Pentium is shown in figure 4.13. The Pentium is a 32 bit processor with a double 64-bit data bus inside and outside of the chip. The internal and external address bus of Pentium is 32-bits wide. Hence, it can access 4 GB of memory. There are two separate 8 KB caches- one for code and for data. Each cache has a separate address translation TLB associated with it. The availability of dual cache and dual TLB permits the CPU to handle simultaneous instruction and data operand access, thus providing efficient handling of the pipeline. There are 256 lines between the codes cache and prefetch buffers, permitting the prefetching of 32 bytes of instructions.

8-stage floating-point pipeline

- PF- Prefetch: Prefetch instructions from the code cache
- D1- First decode: Same as in the integer pipeline.
- D2- Second decode: Same as in the integer pipeline.
- E- Operand fetch: operands are fetched either from the floating-point register file or the cache.
- X1- First execute: First step in the floating-point execution by the FPU (Floating-Point Unit).
- X2- Second execute: Second step in the floating-point execution by the FPU
- WF- Write float: The FPU completes the floating-point computation and writes the result into the floating-point register file.
- ER- Error reporting: The FPU reports internal special situations that might require additional processing to complete execution and updates the floating-point status word.

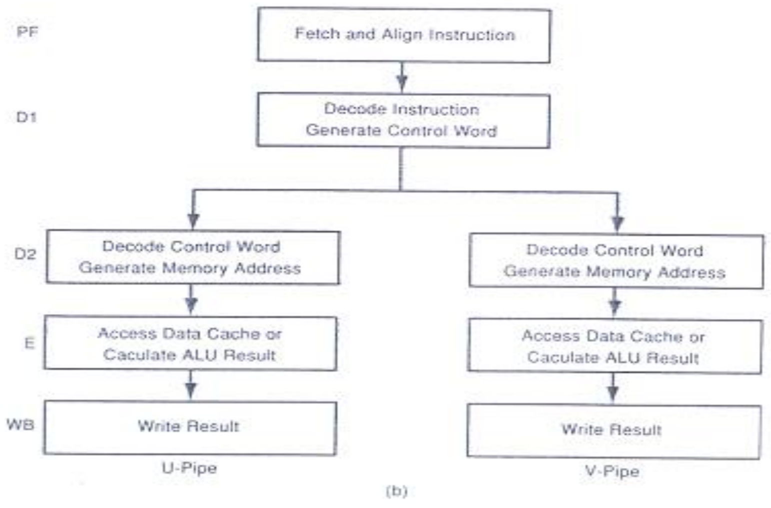


Figure 4.14: Integer pipeline stages in Pentium

Physical memory (4 GB) organization in Pentium

The physical memory organization in Pentium-based system is shown in figure 4.15. The eight bytes of 64-bit quad word are activated by eight bank enable signals BE7-BE0.

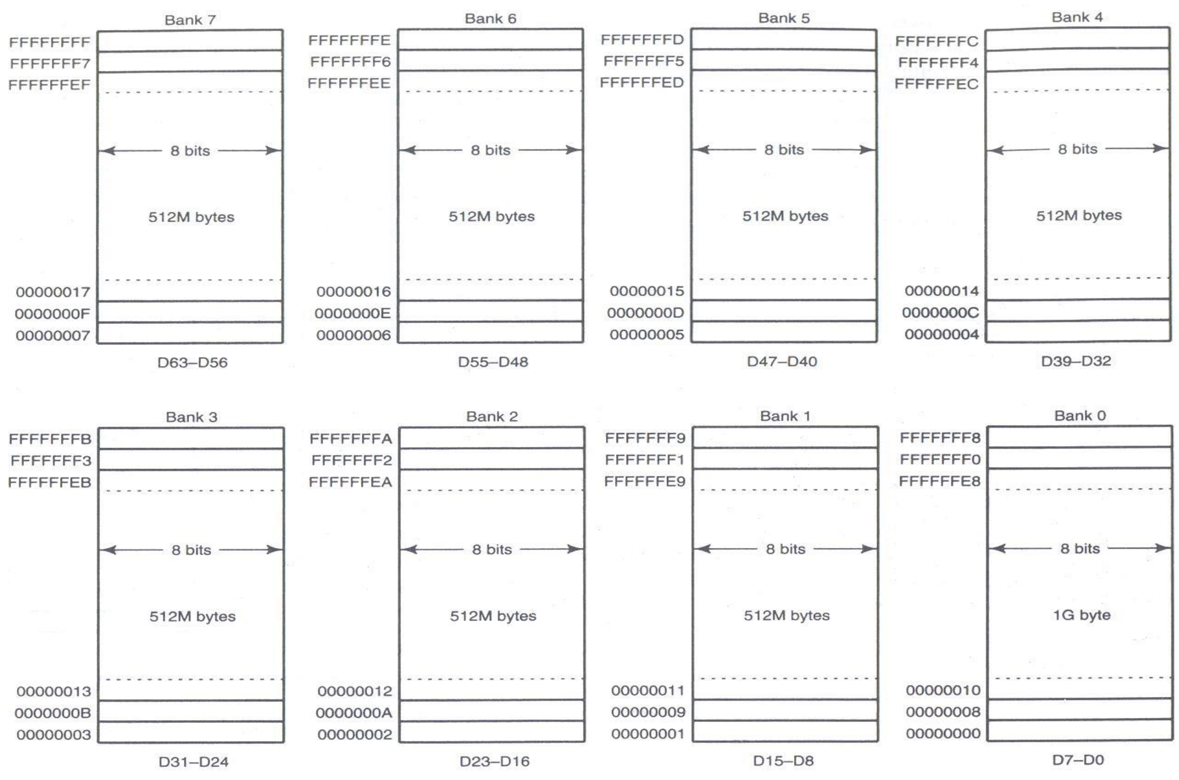


Figure 4.15: physical memory (4 GB) organization in the Pentium