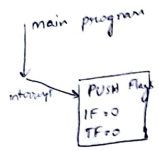
MPMC

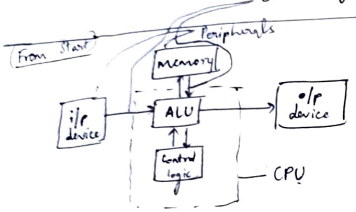
Interrupt Instructions

INT nn nn: 00 to 255



this topic
↳ Architechture
↳ Instruction set & programming
↳ I/O Interfacing

Interfacing



CPU can be constructed using discrete IC's like subtractor IC, .. etc
(or)
single IC for entire CPU

CPU built into a single IC
→ microprocessor

max memory capacity = 2^n , n = address lines.

8086 , n = 20 ⇒ 2^{20} = 1MB

Intel's first 16-bit µp

no. of data lines = 16

clock frequency of standard 8086 = 5 MHz

µp operations :

1) µp initiated operations
   a) Memory read
   b) Memory write
   c) i/o read
   d) i/o write
2) Internal operations
3) Peripherally initiated operations.

To perform these,

1) µp must identify memory or i/o with its address
   ↳ address bus (20 bit in 8086)
2) Provide timing & synchronisation signals

$\overline{RD}$ = 0 ⇒ Read operation
= 1 ⇒ Not a read

$\overline{WR}$ = 0 ⇒ write
= 1 ⇒ not write

M/$\overline{IO}$ = 1 ⇒ operation refers to memory
= 0 ⇒ IO operat^n

Memory Read

M/$\overline{IO}$	$\overline{RD}$	$\overline{MR}$
0	0	1
0	1	1
1	0	0
1	1	1

⇒ $\overline{MR}$ = M/$\overline{IO}$ + $\overline{RD}$

iii) perform data transfer
⇓
by data bus (16 bit for 8086)

2) Internal operations :

i) Store data : Registers
   ↳ General purpose registers

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

   ↳ Base & index registers (16-bit)
   ↳ segment

BP, SP } pointer registers (used in stack applic^n & address calc. app^n)
SI, DI } index register
(used in string operat^n)

Ax → 16 bit accumulator

ii) segment registers (16 bit)

CS
DS
SS
ES
IP

2^{20} = 1MB memory
each segment = 64 kb

1 kb = 1024 x 8 bits

eg

[CS] = 5000H
(1st 16-bits of starting address of code segment

eg: [IP] = 2345H

Physical address = [CS] x 10H + [IP]

Data segment : used to store data
64 kb.

Advantage of memory segmentat^{n}:

→ can access 4 segments x 64 kb ⇒ 256 kb of info; can be accessed simultaneously.

→ Same memory can be accessed by multiple users
8086 → multi-user config.

Stack segment : temp. storage extra segment :→ if data is excess of 64kb,

→ or if in multi-user config, if 2 users want to use same data, common data can be stored here

→ String operations.

ii) Perform arithmetic & logical operat^{n}
in 8086 [ALU] → 16-bit ∴ 8086 is called 16-bit µp
⇒ to perform any 16-bit operat^{n}, it will take only 1 clock cycle

iii) Testing for the conditions
↳ flag register → 16-bit
9 diff flags.

Flag registers

x	x	x	x	OF	DF	IF	TF	SF	ZF	x	ACF	x	PF	x	CF

Flag → basically a flip-flop ⇒ 2 states → can be set or reset

↳ Conditional → CF, PF, ACF, ZF, SF & OF
↳ Control → DF, IF, TF

① Carry flg:
During addition :- Suppose, result of 2 8-bit numbers > 8 bit
CF=1 (set)
otherwise CF=0 (reset)

During subtraction: Minuend - subtrahend, Sub > Min,
(i.e subtraction needs borrow) ⇒ CF=1

② Parity flag: in data communicat^{n} systems
(to detect error & correct)

If binary no. contains even 1's ⇒ even parity
odd 1's ⇒ odd parity.

⇒ if result of any AL operat^{n} contains even 1's, PF set
otherwise PF=0

eg: 32H → 0011 0010
59H → 0101 1001
1000 1011 → contains 4 1's ⇒ PF=1

③ AF: Normally used in BCD arithmatic
lower order 4 bits are inp
if carry from lower nibble to upper nibble, AF=1
otherwise AF=0

or in subtract^{n},
if lower nibble needs borrow from upper nibble

eg: 39H + 5AH = 93H
ACF=1

A+9 = A+5+4
= F+4 = F+1+4
= 10+3
= 13H

A+5 = F

(or) can be done in binary

39H 0011 1001
5AH 0101 1010

① Zero flag: if result of any AL operat^n is 0 , ZF = 1
otherwise ⇒ 0

eg: FF H
+ 01 H
neglect ① 00 ⇒ ZF = 1

② Sign flag: -ve no. MSB = 1
+ve MSB = 0

signed no.s
i) sign-magnitude
ii) signed 1's comp
iii) signed 2's comp → in 8086 this is used.

MSB → signed bit
n-1 → magnitude bits

-2^{n-1} to $2^{n-1}-1$

eg: n = 8 ⇒ (-128 to 127) using signed 2's comp.
(0 → 255) using unsigned 2's comp

If result of any arithematic operat^n is -ve (neglecting carry)
ie MSB = 1 ⇒ SF = 1
otherwise SF = 0

eg: F2 H → 1111 0010
9A H → 1001 1010
① 100 D 1100
carry MSB

③ Overflow flag: (-2^{n-1}) to $(2^{n-1}-1)$
If result of any A operat^n is outside above range OF = 1
eg: $(100)_D + (50)_D = (150)_D$ ⇒ OF = 1 (range -128 to 127)

Control flags: flags can be programmed by user.

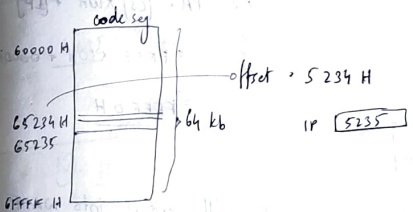
i) Direct^n flag:
If DF = 1 ⇒ (In) string operat^n will be operated in auto decrement mode
DF = 0 → auto increment mode

ii) Interrupt-enable flag:
IF = 1 ⇒ µP recognises the interrupts
IF = 0 ⇒ µP does not recognise

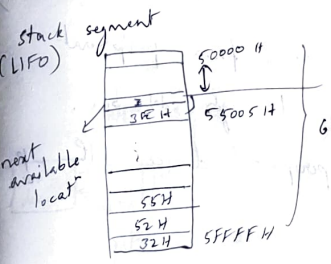
iii) Trap flag: used for debugging the program
If TF = 1 by programmer, µP operates in single-stepping mode ie after each instruct^n is executed, µP will be interrupted
Drawback: takes lot of time
So in interrupt service subroutine you can write instruct^{n}s to display contents of registers & memory locat^{n}s (basically use like console to debug like in js)

another method to debug:
Break-point method: programmer can set like after 10 instruct^{n}s (using assembler directives)

iv) Sequence the execution of instructions
↳ IP → instruction pointer. (holds address of instruction to be executed next) (effective or offset address)



v) Store the data temporarily during execut^n in a defined R/W locat^n called stack



stack top: locat^n upto which data is full

stack pointer: holds the offset of stack top
here 5005

Need of Stack:

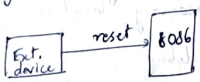
① (During subroutines (or) procedures)
whenever subroutine is called, before jumping there contents of IP (which holds the next address (of after sub)) must be pushed to stack.

or If subroutine uses registers, old content of registers → to be pushed to stack

---

③ Externally or Peripherally initiated operations:

i) Reset (pin)



if RESET = 1 ⇒ μP suspends all internal operations & resets all registers, flags to 0.

DS, SS, ES, IP = 0, CS = FFFFH

If RESET = 0 ⇒ μP fetches the instruction from CS

i.e PA = [CS] X10H + [IP]

= FFFF X10H + 0000H

= FFFF0H

ii) READY

if READY = 0 ⇒ μP will enter into WAIT state

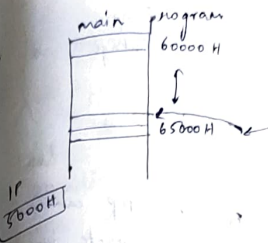
(suppose peripheral is slower compared to μP, until data is ready, μP can be made to wait)

if READY = 1 ⇒ μP reads data from peripheral through data bus.

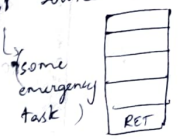
used to synchronise peripheral device with μP

---

iii) Interrupt: can be hardware or software.



if it receives interrupt,
① It will execute current instruct^n
② Then μP control goes to Interrupt Service Subroutine



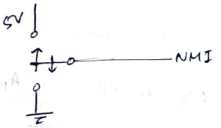
In 8086, 2 hardware interrupts:

INTR & NMI (Non-maskable interrupt)
(Maskable interrupt) — cannot be disabled

can be disabled by the programmer through some program

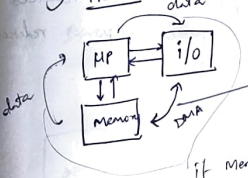
INTR: +ve level triggered interrupt

NMI: +ve edge triggered interrupt



---

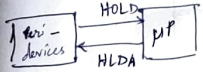
iv) HOLD:



DMA: Direct Memory Access
↳ control of buses need to be taken from μP.
⇒ peripheral devices do this using HOLD

if Memory to i/o ⇒ long & time consuming

HOLD signal = request to the μP to hand over control of address & data buses to the peripheral for DMA transfer

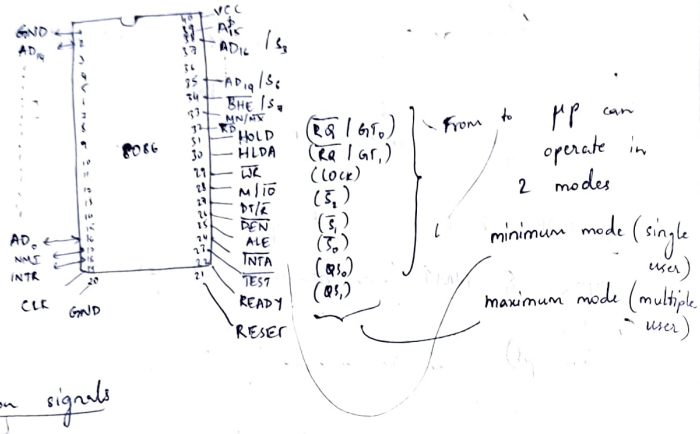


μP will acknowledge through HLDA (Hold acknowledge signal)

After transfer is over, control back to μP

PIN DIAGRAM

8086 → is a 40-pin IC available in 8 DIP · Dual Inline Package



From 8 to 40 µP can operate in 2 modes

- minimum mode (single user)
- maximum mode (multiple user)

Common signals

(32)

8 → Minimum mode exclusively
8 → Maximum mode
48 (in total)

- V_{cc} = 5V
- 2 grounds → to cancel noise
- 20 address lines
- 16 data lines

① AD_{15} to AD_0 & A_{19}/S_6 to A_{16}/S_3

→ lower order 16 address lines are multiplexed with data lines

AD_{15} to AD_0 ⇒ Multiplexed address data bus

reason ⇐ we can reduce no. of pins ⇒ area reduces ⇒ power reduces

$A_{15} \rightarrow A_0$ = lower order 16 bit of address bus

$D_{15} \rightarrow D_0$ = data bus

---

1 clock period of any µP ⇒ 1 T state

8086 bus cycle → 4 T states

① Identify device with its address
② generate control signal (Read / Write)
③ Transfer data

- In first T state AD_{15} to AD_0 is used to carry address
- remaining 4 bits of 20 bit are multiplexed with status signals i.e A_{19}/S_6 to A_{16}/S_3 ⇒ Multiplexed Address-status bus

total 20 address lines during T1.

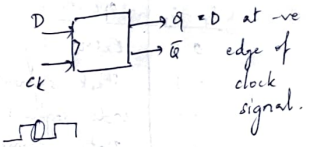
In subsequent T states AD_{15} to AD_0 carry data & A_{19}/S_6 to A_{16}/S_3 carry status signals

How will this be latched ?

⇒ ALE → Address Latch Enable (signal)
↳ used to demultiplex address-data bus & address-status bus

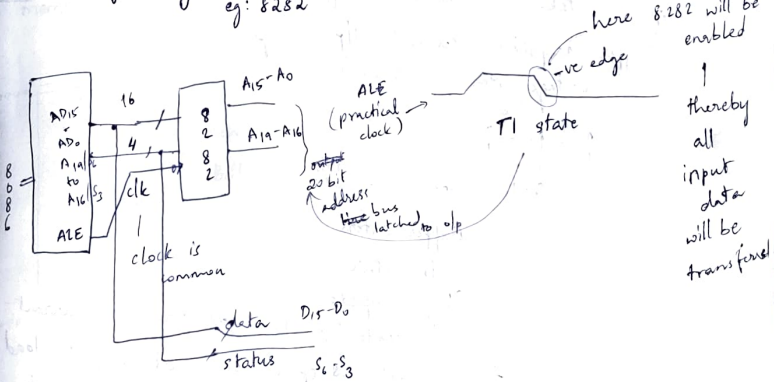
How exactly is it demultiplexed ?

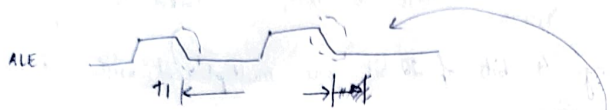
⇒ latches (D flip-flop) -ve edge



So, the only condition for latching this data into o/p is clock has to be changed from 1 to 0 ie -ve edge

So by using octal latch (with 8 latches inside an IC)
eg: 8282





So o/p will have 20 bit address bus until next high to low

In subsequent clock cycles only data & status can be extracted

Now address bus, data bus, status buses are available seperately.

- CLOCK (CLK) : This clock has to be generated using the external crystal generator (crystal oscillators are stable)

- $S_3 - S_6$

S_4	S_3	operation
0	0	Extra segment
0	1	Stack segment
1	0	code segment / no segment
1	1	Data segment

S_6 : (during HOLD operation)
S_6 is normally low when buses are under µP control.

once µP uses buses & is ready to give buses to memory, it makes S_6 go to high impedance state

Suppose, take a tri-state buffer (buffer amplifies current, i/p & o/p voltage remains same)

C = 1 ⇒ Normal operation

C = 0 ⇒ Buffer enters tri-state (High impedance state 'z')

⇒ almost acts as open circuit ⇒ draws no current ⇒ not going to load the device connected to o/p of buffer

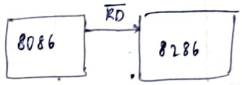
---

BUS CYCLES OF 8086

- $\overline{BHE}/S_7$ — Bus High Enable
  - carries $\overline{BHE}$ information during T_1 & in subsequent clock cycles, S_7.

- READ ($\overline{RD}$), $\overline{DEN}$, DT/$\overline{R}$, $\overline{WR}$

8286 → an external device (a tranceiver) (combination of transmitter & receiver)



µP can read or write into transceives.

$\overline{DEN}$ → Data Enable

→ during read operation, $\overline{DEN}$ is low

$\overline{RD} = 1$

DT/$\overline{R}$ → Data transmit / $\overline{\text{Receive}}$ → low / 0

→ during write operation, since µP is sending data to peripheral device, $\overline{DEN}$ will be low early as compared to

$\overline{WR} = 1$

DT/$\overline{R}$ → high / 1

- $\overline{BHE}$ → enables higher order 8 bits of data BUS i.e D_{15} to D_8 (lower is $D_7 - D_0$)

memory of 8086 = 1MB = 1024 KB

Half of this = 512 KB

→ each location capable of storing 8 bits

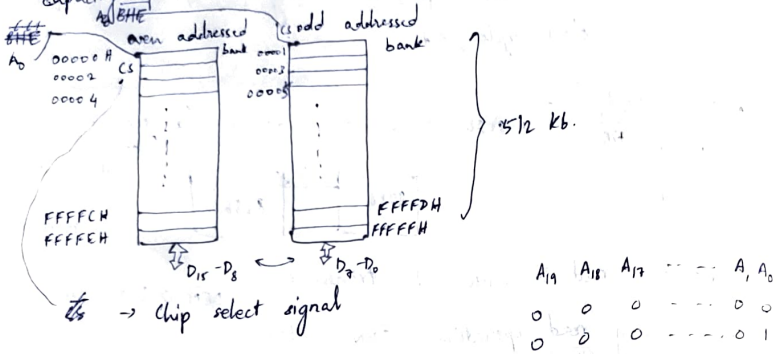
But µP 8086 is 16-bit. In some applications we need 16-bit data also

To enable 16-bit data into from memory we would require 2 clock cycles.

Instead,

To simultaneously or parallely read 16 bit, the memory is divided into 2 banks — odd & even bank.

capacity of each bank = 512 kb



$\overline{CS}$ → Chip select signal

A_{19}	A_{18}	A_{17}	...	A_1	A_0
0	0	0	...	0	0
0	0	0	...	0	1
0	0	0	...	1	0

A_0 = 0 for even addresses

$\overline{BHE}$	A_0	Memory operation
0	0	Two byte (word) operation (D_{15}-D_0)
0	1	one byte from odd bank (D_{15}-D_8)
1	0	" " " even bank (D_7-D_0)
1	1	No data (No bus)

- S_7 : low during interrupt acknowledge signal
- $\overline{INTA}$
- MN/$\overline{MX}$ = 1 ⇒ minimum mode
  = 0 ⇒ maximum mode
- $\overline{TEST}$: (during WAIT)
  i.e μP will WAIT until $\overline{TEST}$ = 0
- READY
- RESET

& exclusive MAXIMUM MODE signals

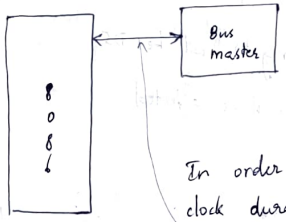
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Function
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Code access
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Not used

- QS = Queue status

QS_1	QS_0	operation
∅	∅	Reads 1st byte (OPcode fetch from queue)
0	1	
1	0	Empty the queue
∅	↓	
0	0	Not used
1	1	Read subsequent bytes from queue

- $\overline{RT}/GT_0$; $\overline{RT}/GT_1$

Request by grant 0 & request by grant 1



Whenever external bus master wants to use system bus (address + data) it will request through one of these signals ($\overline{RT}/GT_0$ or $\overline{RT}/GT_1$)

In order to request, it will send one clock durat" pulse on this line

($\overline{RT}/GT_0$ has higher priority)

Then μP → first completes ~~execution of~~ current instruct" / operat"
→ then it informs bus master through same signal, (1 clk pulse), that 8086 will relinquish system bus to bus master.

After data transfer, bus master sends active low signal (not 1 clock durat" (very short)) ($\overline{RT}/GT_0$ or $\overline{RT}/GT_1$). Then 8086 regains control of system bus.

i) $\overline{LOCK}$: A low on $\overline{LOCK}$ signal indicates that request of other bus master is not granted. (until service of first bus master)

once first bus master returns control of system bus back to µP, $\overline{LOCK}$ → High

INSTRUCTION SET

i) DATA transfer group

a) MOV instructions:

MOV register 2, reg/memory 1 [reg 2] ← [reg1/mem]

REG - REG
MEM - REG
REG - MEM

* Data transfer from mem2 to mem1 is not allowed.

eg: MOV CL, BL CL [35] BL [99]

After
CL [99] BL [99]

* Size of both registers if used, must be same
* By default segment register will be DS
* None of the flags will be affected

eg: [DS] = 5000H [BX] = 2000H 52000 [2C] 52001 [32]

MOV AL, [BX]

PA = 5000 × 10H + 2000H = 52000H ⇒ AL = [2C]

MOV AX, [BX] ⇒ AX [32|2C]

this can be BX, BP, SI, DI

can be BX, BP (Base addressing mode)

eg 2: MOV AL, START [BX] DS = 5000 START = 0040
or MOV AL, [START + BX] BX = 3900

PA = [DS] × 10 + START + [BX]

= 50000 + 0040 + 3900 = 53940

MOV AL, START [DI] ⇒ Index addressing mode

can be SI / DI

MOV AL, START [BX][DI] ⇒ Base Indexed addressing mode

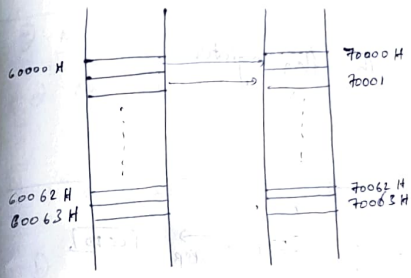
PA = [DS] × 10H + START + BX + DI

= 53940 + 9000 = 5C940

b) MOVSB : Move string byte

Let length of string $(100)_D$ = 64 H

Source Destination



* Before MOVSB instruct^n, load [0064H] to CX
* set or reset Direction flg (CLD or STD)

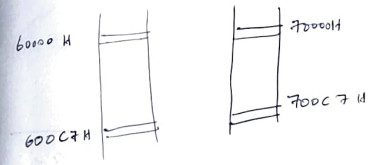
ie MOV CX, 0064
CLD
MOVSB

c) MOVSW : Move string word (2 bytes)

For transferring same no. of bytes as above ie (~~64 H~~ C8 H), 100 ~~bytes~~ words = 200 bytes

[SI] = 6000, [DI] = 7000

$(200)_D$ = C8 H



(mem)

d) MOV segment, mem/reg

CS, DS, SS, ES

we have to use data segment by default to compute the PA.

i) DS: START : Direct
ii) [BX], [BP], [SI], [DI] : Reg indirect
iii) [BX/BP + START] : Based
iv) [DI/SI + START] : Index
v) [BX/BP + DI/SI + START] : Based Index

LOAD Instructions

1. LEA reg, reg/mem

2. LDS → load data segment

LDS reg, mem

[reg] ← [mem]

[DS] ← [mem+2]

eg: LDS BX, [SI]

DS [5000] SI [2000]

PA = 52000H

BX [BE AA]

DS [DD CC]

52000 [AA]
52001 [BB]
52002 [CC]
52003 [DD]

3. LES reg, mem

4. LAHF : Load AH register with flag register

[AH] ← [FRL]

(flag register lower)

PUSH and POP instructions,

1) PUSH reg — suppose BX [CCDD] push → [DD CC] pop → [CC DD]
2) PUSH mem
3) PUSH segment reg → cannot be CS
4) PUSH flag register ⇒ PUSH F

[[SP]-1] ← FRH

[[SP]-2] ← FRL

SP ← SP-2

EXCHANGE Instructions

1) XCHG reg2, reg1

[reg2] ↔ [reg1]

2) XLAT = MOV AL, [AL][BX]

[AL] ← [AL]+[BX]

used to convert ASCII code to EBCDIC

Extended Binary coded decimal interchange code

Arithmetic Instructions

1) ADD instruct^n

a) ADD reg/mem2, reg/mem1
* Both cannot be memory

b) ADD reg, data

c) ADD mem, data

d) ADC reg/mem2, reg/mem1

[reg/mem2] ← [reg/mem2] + [reg/mem1] + [CF]

e) ADC reg, data

f) ADC mem, data

III/y sub instructions

SBB: ---- -[CF]

2) MUL instructions

a) MUL reg/mem (unsigned mult^n)

For 8-bit × 8-bit mul^n : [AX] ← [AL] × [reg8/mem]

For 16 bit × 16 bit : [DX:AX] ← [AX] × [reg16/mem]

b) IMUL for signed multi^n)

3) DIVISION

a) DIV reg/mem (for unsigned)

* For 16-bit by 8-bit : [AX] / [reg/mem] → remainder → [AH], quotient → [AL]

* For 32-bit by 16-bit : [DX:AX] / [reg/mem] → remainder → [DX], quotient → [AX]

b) III/y for signed division,
IDIV reg/mem

4) INC reg : increments [reg] ← [reg] + 1

5) DEC [reg] ← [reg] - 1

7) DAA : Decimal adjust AL after addition
(implicit addressing mode)

* Used in BCD addition

eg:
```
MOV BL, 09H
MOV AL, 05H
ADD AL,BL        [AL] ← [AL] + [BL]      AL 0E
DAA                                      AL 14
```

Internal operat^n:
- If lower nibble of AL is > 9 or if AF is set
  it adds 6 (00110) to lower nibble.
- If higher nibble of AL is > 9 or CF is set
  add 0110 to upper nibble

eg:
```
09H :  0000 1001
05H :  0000 0101
       ---------
       0000 1110
       0000 0110
       ---------
       0001 0100  → 14
```

8) DAS : Decimal adjust after subtraction

if AF set, - 6 (0110) from AL lower nibble
CF set, - 6 from AL upper nibble.

9) AAA : ASCII Adjust AL after addition

0 → 30 H
1 → 31 H
⋮
9 → 39 H

To perform ascii addition without masking 3

[AL]: 39 H
[BL] = 35 H

```
  39
  35
 6E H
```

But you need (14) CF

if AAA used AL 04
CF 1

10) AAS : ASCII adjust for subtraction

11) AAM : BCD adjust after multiplicat^n

eg: 05 D x 09 D = 45 D

```
MOV AL, 05 H
MOV BL, 09 H
MOV B
MUL BL
```

[Ax] ← [AL] * [BL]

Ax 002E But we need 45 D

→ AAM

AH	AL
04	05

unpacked BCD (ie 1 BCD per byte digit)

12) AAD : BCD to Binary convert before division

↳ converts unpacked BCD digits in Ax to its equivalent binary value in AL

67 D
9 D

07 + r = 04

67 D = 43 H

Ax

06	07

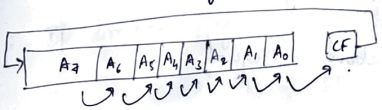
```
AAD
AL 43
MOV CH,09H
DIV CH
[AL] = 07H
[AH] = 04H
```

LOGICAL, Rotate, Shift & Compare :

1) NOT reg/mem (1's complement)
2) NEG reg/mem (2's complement) Conditional flags will be affected.
3) OR reg/mem2, reg/mem1
4) AND reg/mem2, reg/mem1
5) XOR reg2/mem2, reg1/mem1

Shift & Rotate instruct^ns

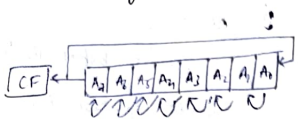
1) RCR reg/mem, 1
(Rotate through right with carry)



2) RCL reg/mem, CL (or) by value stored here

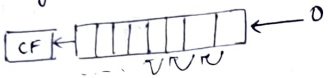
CL ← default reg, cannot use other reg.

3) ROL reg/mem, 1 or , CL , 4) ROR reg/mem



SHIFT operation:

1) SAL reg/mem, 1 or , CL



2) SAR

COMPARE instructions:

1) CMP reg/mem2, reg/mem1

[reg/mem2] − [reg/mem1] → Result is discarded
Based on result flags are set or reset.

Branch instructions
↳ Conditional : JZ/JE addr
↳ Unconditional : JMP reg/mem
[IP] ← [reg/mem]

(17) Conditional JMP instructions

1) JZ/JE addr / / set
2) JNZ/JNE addr if zero flag reset
3) JS addr : Jump if -ve / signed
4) JNS addr : Jump if no sign or +ve
5) JP/JPE addr: Jump if parity even
Even
6) JNP/JPO addr: Jump if parity odd or no parity
odd
7) JC/JB/JNAE addr if Not above or equal
carry if Below
8) JNC / JNB / JAE addr
no carry JA Above JNBE if not below or equal
9) JO addr
if overflow set
10) JNO
no overflow.
11) JG / JNLE
Greater Not lower or equal
12) JL / JNGE
13) JGE / JNL
Greater or equal Not less
14) JLE / JNG
15) JCXZ addr : if CX=0
16) JBE / JNA
Below or equal
17) JAE / JNB

CALL instructions: associated with subroutines or procedures
(Stack concept)
Lec 10:

STRING Instructions:

1) LOAD string byte : LDSB
[AL] ← [[SI]]
[CS] = 5000H
[SI] = 2000H

52000H
FE

LDSB
AL FE

If direction flag = 0, contents of SI will be incremented by 1
DF = 1 ⇒ SI content − 1

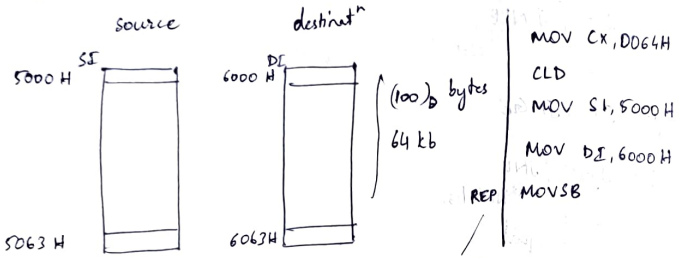
ii) LDSW DF=0 ⇒ increment by 2
DF=1 ⇒ decrement by 2

AX ← SI & SI+1

iii) MOVSB / MOVSW

[[DI]] ← [[SI]] if DF=0 ⇒ SI & DI incremented by 1

eg: to move



iv) CMPSB / CMPSW

[[SI]] − [[DI]]

For Above eg.,

MOV SI, 5000H
MOV DI, 6000 H
MOV CX, 0064 H
CLD
REP CMPSB ⇒ operatn continuees until CX=0000H (or) if [[SI]] = [[DI]] ie ZF is set

↳ REPE / RPEZ

v) SCASB / SCASW → [AX] − [[DI]]

[AL] − [[DI]]

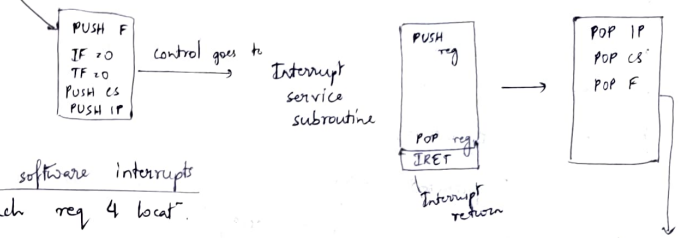
MOV AL, 00H
MOV CX, 0064 H
MOV DI, 6000 H
CLD
REP SCASB

INTERRUPT

INT nn nn = 00 to 255

Main program

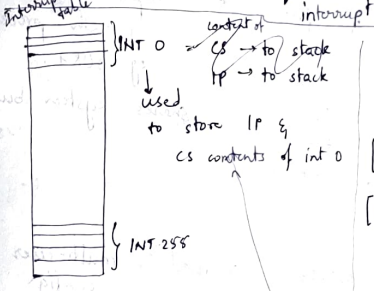
int



256 software interrupts

each req 4 locatn

⇒ 256 x4 = 1024 = 1 kb req. to store interrupt addresses



INT nn

4 x nn

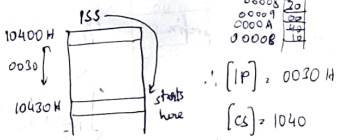
[IP] ← [(4 x nn)]

[CS] ← [(4xnn +2)]

eg INT 2

[IP] ← [[0008H]]

[es] ← [[000AH]]



∴ [IP] = 0030 H

[CS] = 1040

1. INT 0 : Divide by zero
INT 1 : single-stepping
INT 2 : Non-maskable interrupt (NMI)
INT 3 : Break-pointing
INT 4 : Interrupt on overflow

} Predefined interrupts

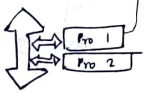
2. Interrupts for future purpose
INT 5 – INT 31

3. Maskable interrupts
INT 6 – INT 255

4. IRET

Status & Instructions / Process Control Instruct

1) CLC : Clear carry flag
2) CLD : clear direction flag
3) CLI : Clear interrupt flag (all except NMI)
4) STC : Set Carry
5) STD : Set Direction
6) STI : Set Interrupt
7) CMC : Complement Carry
8) NOP : No operations (creates delay)
9) HLT : Halt
10) ESC : escape ⇒ control transferred to co-processor or (math processor)
    eg: 8087
    shares system bus.
11) WAIT : After execut^n, μP enters idle state until low signal available on TEST pin.
12) LOCK prefix : eg: LOCK MOVSB ⇒ used in multi-user config

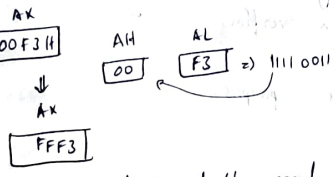


if Pro. 1 uses this, pro 2 cannot request access to BUS

used for emergency instruct^s.

13) CBW : Convert Byte to Word (operates only on AX)
    copies sign bit of AL on to AH

eg:



14) CWD : Convert word to double word.
    copies sign bit of AX on to DX.

I/O Instructions

* each I/o device + port has some addr.
* only AL & AX registers used
* 2 instruct^s : IN & OUT

↳ IN AL/AX, port addr.
[AL/AX] ← (port addr)

DX can be used to store port address
eg: IN AL, DX
IN AX, DX

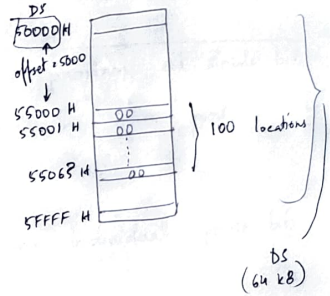
Fixed
OUT port addr, AL/AX
(or)
OUT DX, AL/AX

8086 : (In total 117 Instructions)

Programming :

ex: 1 Write AP to clear 100 consecutive memory locations

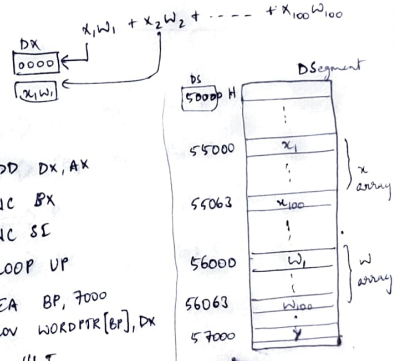
→
```
MOV CX, 0064 H
MOV AX, 5000 H
MOV DS, AX
LEA BX, 5000 H
UP : MOV BYTEPTR [BX], 00 H
     INC BX
     LOOP UP
     HLT
```



ex: 2 : $Y = \sum_{i=1}^{100} x_i w_i$

$x_1w_1 + x_2w_2 + \dots + x_{100}w_{100}$

→
```
MOV AX, 5000 H
MOV DS, AX
MOV CX, 0064 H
LEA BX, 5000 H
LEA SI, 6000 H
MOV DX, 0000 H
UP : MOV AL, [BX]
     IMUL BYTEPTR [SI]
     ADD DX, AX
     INC BX
     INC SI
     LOOP UP
     LEA BP, 7000
     MOV WORDPTR [BP], DX
     HLT
```



In order to interface I/O devices we need a programmable peripheral chip interface (PPI) 8255

8255 - 40 pin IC

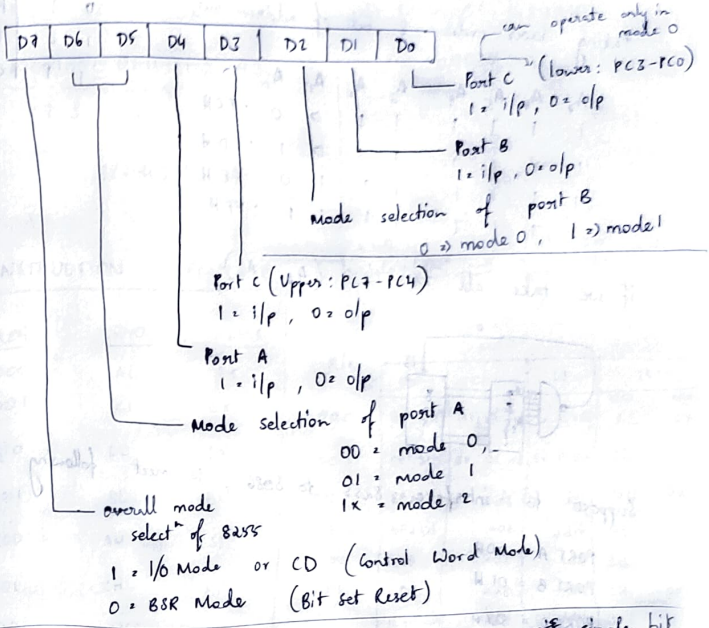
with 3 ports → Port A, Port B, Port C

can operate in 3 modes
- mode 0
- mode 1
- mode 2

each port can be programmed as i/p or o/p port.

8255 Block Diagram

Format of Control Word

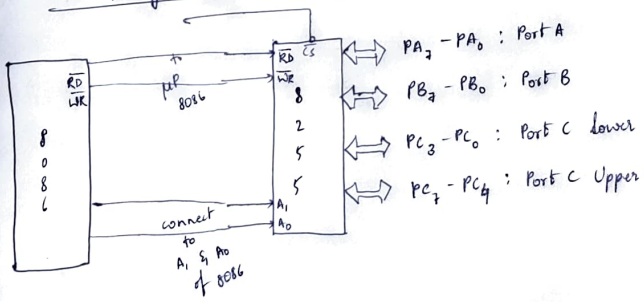


1) In CD or I/O Mode, we cannot program single bit of a port as i/p or o/p port. The entire port can be programmed as i/p or o/p.

2) In BSR mode; we can program individual port bits as i/p or o/p.

i.e $PA_7 - PA_0$, we can set PA_0 = i/p, PA_1 = o/p

Interfacing 8255 to 8086



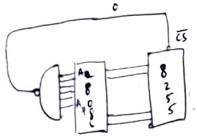
How to find chip select signal :

A_1	A_0	
0	0	Port A = FCH
0	1	Port B = FDH
1	0	Port C = FEH
1	1	CWR = FFH

taking lower order 8 bits of address bus,

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	1	1	1	1	1	0	0	: FCH
1	1	1	1	1	1	0	1	: FDH
1	1	1	1	1	1	1	0	: FEH
1	1	1	1	1	1	1	1	: FFH

if we take all as 1's $(A_7 - A_2)$:

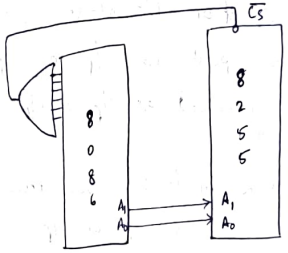


Suppose to interface 8255 to 8086, to meet following specifications

PORT A = 00H
PORT B = 01H
PORT C = 02H
PORT D = 03H

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
00H =	0	0	0	0	0	0	0	0
01H =	0	0	0	0	0	0	0	1
02H =	0	0	0	0	0	0	1	0
03H =	0	0	0	0	0	0	1	1

A_2 to A_7 = 0



① MOV 43H[SI], DH

10001000 01110100 43

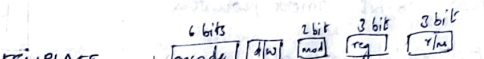
8 8 7 4

88743 H

② MOV CL, [437AH]

100010

INSTRUCTION TEMPLATE



REG	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

R/M	MOD 00	01	10	11 W=0	11 W=1
000	[BX]+[SI]	[BX]+[SI]+d8	BX+SI+d16	AL	AX
001	BX DI	BX DI d8	BX DI d16	CL	CX
010	BP+SI	+d8	+d16	DL	DX
011	BP+DI	+d8	+d16	BL	BX
100	SI	+d8	+d16	AH	SP
101	DI	+d8	+d16	CH	BP
110	16 bit addr	+d8	+d16	DH	SI
111	[BX]	+d8	+d16	BH	DI

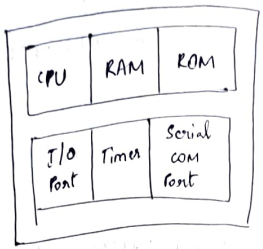
segment overwrite prefix : 001___110

00 ES
01 CS
10 SS
11 DS

8051 "on-chip computer"

Basic Components

- 4k bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0-P3)
- Two 16-bit timers / counters
- One serial interface



Microcontroller → can be considered as single chip computers.

Other 8051 features

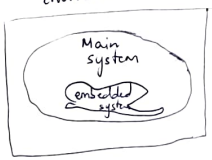
- Only 1 on-chip oscillator (external crystal)
- 6 interrupt sources (2 external, 3 internal, Reset)
- 64k external code (program) memory (only read) & PSEN
- 64k external data memory (can be read & write) by RD, WR.
- Code memory is selectable by EA (internal or external)

Embedded System (8051 Application)

→ An embedded system is closely integrated with the main system.

→ It may not interact directly with the environment.

→ For eg: A microcomputer in a car ignition control.



Comparison of 8051 Family Members

- ROM type
  - 8031 no ROM
  - 80xx mask ROM
  - 87xx EPROM
  - 89xx Flash EEPROM
- example (AT89C51, AT89LV51, AT89S51)
  - AT: ATMEL (Manufacturer)
  - C: CMOS Technology
  - LV: low Power (3.0 V)

PIN Diagram of 8051

* One of the most useful features of 8051 → it contains 4 I/O Ports (P0-P3) [since most µcs talk to the environment, these I/O ports are imp in them]

- Port 0 (pins 32-39) : P0 (P0.0 ~ P0.7)
  - 8 bit R/W - General purpose I/O
  - Or acts as a multiplexed low byte address & data bus for external memory design.

→ we write like this because these ports are bit accessible. you can control individual bits seperately.

- Port 1 (pins 1-8) : P1 (P1.0 ~ P1.7)
  - → P1 is not multiplexed
  - So when you want to choose among the ports, P1 can be 1st option
  - → 8 bit R/W - General purpose I/O

Port 0 → lower order address & data bus

Port 2 - higher order

- Port 2 (pins 21-28) : P2 (P2.0 ~ P2.7)
  - → 8 bit R/W - General purpose I/O
  - → Or high byte of address bus for external memory design.

- Port 3 (pins 10-17) : P3 (P3.0 ~ P3.7)
  - → General purpose I/O
  - → If not using any of internal peripherals (timers) or external interrupts.

* Each port can be used as input or output (bi-directional)
* Can be controlled at bit-level.

Port 3 Alternate Func^ns

P3.0	—	RXD (serial input port)
P3.1	—	TXD (serial output port)
P3.2	—	$\overline{INT0}$ (external interrupt 0)
P3.3	—	$\overline{INT1}$ (external interrupt 1)
P3.4	—	T0 (Timer 0 external input)
P3.5	—	T1 (Timer 1 external input)

P3.6 - $\overline{WR}$ external data memory write strobe)

P3.7 - $\overline{RD}$ (external data memory read strobe)

- RST (pin 9) : reset (normally low)
  - input pin and active high
    - The high pulse must be high atleast 2 machine cycles
  - power-on reset
    - Upon applying a high pulse to RST, the μc will reset & all values in registers will be lost

Reset value of some 8051 registers:

PC → 0000	PSW → 0000	RAM → all are zero (all cleared)
ACC → 0000	SP → 0007	
B → 0000	DPTR → 0000	

- $\overline{EA}$ (pin 31) : external access, active low to access external program memory locations 0 to 4K.
  - → There is no on-chip ROM in 8031 & 8032
  - → $\overline{EA}$ pin is connected to GND to indicate the code is stored externally
  - → $\overline{PSEN}$ & ALE are used for external ROM.
  - → For 8051, $\overline{EA}$ pin is connected to VCC.
  - → "—" means active low.

- ALE (pin 30) : address latch enable to latch address o/p's at Port 0 & 2.
  - → It is an output pin & is active high
  - → 8051 port 0 provides both address & data
  - → ALE is used for de-multiplexing the address & data by connecting to the G pin of the 74LS373 latch.

$\overline{PSEN}$ (out) : Program Store Enable, the read signal for external program memory (active low)

- RXD & TXD : UART pins for serial I/O on Port 3. (Receive & Transmit data)
- XTAL1 & XTAL2 : Crystal inputs for internal oscillator.

- Vcc (pin 40) : provides supply voltage to chip. voltage source is +5V
- GND (pin 20) : ground
- XTAL1 & XTAL2 : (pins 19, 18) : - These 2 pins provide external clock.
  - way 1 : using a quartz crystal oscillator
  - way 2 : using a TTL oscillator

(~~Example 4-1~~)