Date 13-July-2016
Page

INTEL M. PROCESSORS

Processors:

Bus: (Bus can be defined as bunch of wires)

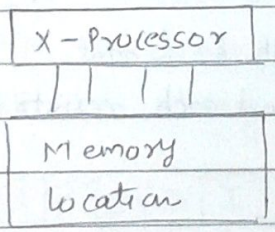
(i) **Address Bus:**

By knowing the size of address bus, we can defined the how much memory locations its provides.

- for Serial comb^n (far distance transmission)
- By Parallel comb^n (short distance transmission)

(ii) **Data Bus:**

- func of data bus is to carry the data.
- Size of data bus (n-bit) = n bit processor
- data bus size always is in 2^N (because data is in binary 0 & 1 so]



20 bit Ad. Bus $\longrightarrow 2^{20}$ Memory locations

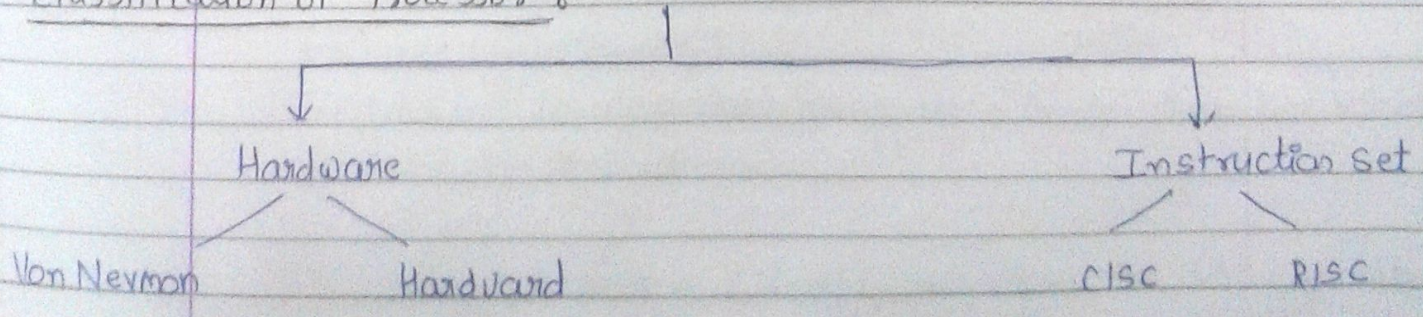
16 bit Data Bus $\longrightarrow$ 16-bit processor

(iii) **Control bus:**

its access to other Processor or memory.

as 4, 10, 16, 32 .. bit

Classification of Processor:



① Von neumann :

Memory
CPU — Data & Program — its having only one single memory

② Harvard :

Program — CPU — Data — its having two different memory for data & Program.

③ CISC :

Complex instruction set computer → CISC
- Diff^r instructions of diff length & format.
- limited general Purpose register.

④ RISC :

reduced instruction set computer → RISC
- large general purpose resistor.
- each instruction of fixed length & format.
- 64 Bit RISC → 64 resistors and each resistor having 64 bits.

8086 Processor :

↳ 16-bit data bus
↳ 20-bit Address bus
↳ 16-bit control bus

Architecture

(A) Execution Unit (EU)
⇒ ALU (addition, substraction, Mul.)
↳ Conditional [CarryFlag, Parity, OverFlag, Sign, Zero, AF]
⇒ Flag Register (16 bit)
↳ Control Flag [InterFlag, Direction F, Trap F]
↳ Unutilized

IF → One process interrupted by another
TF → line by line execution
DF → string operation

Flag :

				O	D	I	T	S	Z		A		P		C

O — Over
D — Direction
I — Inter
T — Trap
S — Sign
Z — zero
A — Auxiliary
P — Parity
C — carry

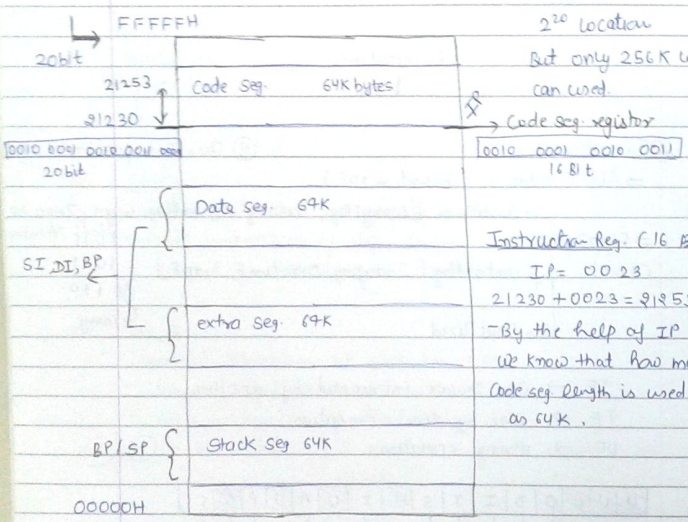
⇒ Index Register [Source Reg , Destination Reg] [SI , DI]
⇒ Pointer Register [Stack pointer , Base Pointer] [SP, BP]

(B) Bus Interface Unit : (It provides the data to execution for execution.)
⇒ Queue (Small) Memory)
FIFO (First in First out) [Six upcoming instructions are stored in Queue for execution)

⇒ Segment Registors (16-bit) only stored the starting address
- Code Reg.

- data reg.
- extra reg.
- stack reg.

⇒ Instruction pointer (IP): is referred to as an offset for the code seg.



2^{20} location
But only 256K we can used.
IP → Code seg. register
0010 0001 0010 0011
16 Bit

Instruction Reg. (16 Bit)
IP = 0023
21230 + 0023 = 21253
- By the help of IP we know that how much code seg length is used as 64K.

- If data seg. is full only then extra seg. can be used to stored
- stack seg. is a memory store aside to store a data (Backup) (Temporary Backup)
- 16 bit ⟶ 20 bit
  Code seg. Register → code segment
  By added 0000 → code seg. adress
  only starting code segment address is ended with 0000

```
CS'   3 4 8 A 0   ← hardware zero
IP  +     4 2 1 4
    3 8 A B 4     ← new address of code segment
```

- A stack is a section of memory set aside to store address and data while a subprogram executes.
- BIU handles all transfers of data and addresses on the buses for the execution unit.

Register organizations

Accumulator	AH	AL	AX
Base	BH	BL	BX
counter	CH	CL	CX
Data	DH	DH	DX

code segment	CS
Data	DS
extra stack	SS
extra	ES

Instruction	IP
Source	SI
Destination	DI
Base Pointer	BP
stack Pointer	SP

- max. code segment memory = 64 Kbyte
- minimum code segment memory = 1 byte = 8 bit
- By the help of instruction point we can access the code segment memory.
- 16-bit processor ⟶ 16-bit data word
  db → data byte OS (
  dw → data word (16bit)
- mov ax, data
  mov ds, ax
  } supply correct data to the code

Accumulator registor : all the data is stored here

Counter registor Cx : gives counter values

Data registor Dx : used in multiplication & division.

Date 21-July
Page

Pin Diagram of 8086 :

In Digital ic's we supply only upto 5V. & In analog ic's we supply upto 12-15V.

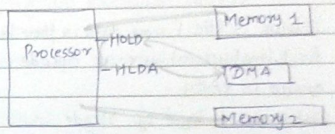
- 40 Pin Ic's [Digital Ic]
- Clock (CLK) : 33% ON or OFF time
- Vcc : power supply 5V ±10%
- RESET : registers, segment & flag
  CS : FFFFH ; IP : 0000H
  when Reset = 1 ; all pins (all values) are reset.
- AD1-AD15 : Address & data Bus
  Address (20-bit)
  data (16-bit)
  which is used, this decided by the ALE (address latch enable)
- ALE : adress latch enable
  When ALE is high that it provides to used as address.
- A16 to A19 : address status Bar
  By S_4(A17) & S_3(A16) we can decide which segment is used.

S_4 A(17)	S_3 A(16)	
0	0	→ extra
0	1	→ stack
1	0	→ code
1	1	→ data

  S_6 (A19) : Logic zero
  S_5 (A18) : Indicate condition of (interrupt) IF flag bits.
- 2 ground Pins - interrupt (8086 has 2 interrupt Pins NMI, INTR
- NMI : non maskable interuppt (can not be disabled Unless the instruction is completed.
- INTR : interuppt request (other processor connected to 8086 wants to use its

Date
Page

- INTA : interrupt acknowledgement
- Direct Memory Acess :
  HOLD :
  HOLD Acknowledge (HLDA) : if memory 2 data is to sent to memory 1 then if Processor is involved then ALU & other operations gets stopped.



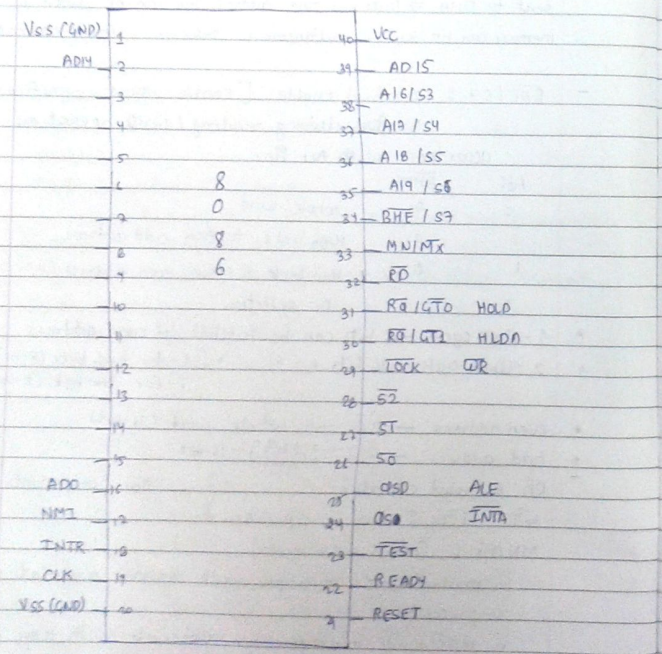
- So, to avoid this DMA will sent HOLD signal, then HLDA signal is sent to DMA ⇒ Data bus and address bus can be used by DMA & memory can be sent simultaneously, processor can do its internal operations
- $\overline{BHE}$ / S7 : Bus high enable [enable most significant data bits D8-D15 during reading / write operations. work with accordance with A0 Pin.

A0	$\overline{BHE}$	
0	0	whole word
0	1	High byte to/from odd address
1	0	low byte to/from even address
1	1	no selection

- 1-Clock cycle - 16 bits can be fetched if even address
- 2-Clock cycle - 16 bits " " " [1-for high byte (8-bit) 1-for low byte (8-bit)]
- even address — 00 → whole word (16-bit)
- odd address — 01 → 8 bit, 10 → 8 bit] — 16-bit
- $\overline{RD}$: read operation } — for peripherials
- $\overline{WR}$: / $\overline{LOCK}$: write operation
- MN/$\overline{MX}$: [Min & max mode]
  in minimum mode microprocessor do not associate with any co-processor
  in max mode microprocessor associate with any co-processor
- $\overline{S2}$: status signal, $\overline{S1}$, $\overline{S0}$

- $\overline{LOCK}$: used to lock the peripherals devices [analogous to sleep mode] internal processing is possible, Peripheral can't use the processor
- $\overline{RQ}$ / $\overline{GT0}$ /HOLD : request] -DMA
- $\overline{RQ}$ /$\overline{GT1}$: Grant]
- Queque status

QS0	QS1	
0	0	Queque is idle [some data is there but we're using not]
0	1	first (instruction) byte of opcode
1	0	queque is empty
1	1	subsequent byte of opcode



Date 26-July
Page

(3.7) Instruction set of 8086 :

Classification :

1. Data transfer instruction
2. Bit manipulation instruction
3. Arithmetic
4. program execution transfer
5. string
6. Processor control

Avoid Mistakes By :

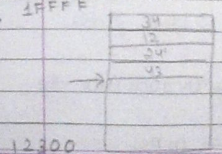
(i) Source and destination operands must be of same size as example move al, 08H
(ii) Can't transfer data between to memory location with the help of single instruction.
(iii) Destination must be register or memory location.

① Data transfer instruction :

Eg. MOV BX, [0301H]
↑ memory location address → offset address

base address + (0301) this value = gives the new location and at new location value, which is stored in Bx (which is 16-bit.

- PUSH Operand : / POP Operand (it pops the operand from top of stack to pop)
  it push the operand into top of stack. Last in first out
  E.g. PUSH BX
  SS = 1230



In stack (LIFO)
it decrement the stack pointer by 2
PUSH BX = 1234
PUSH DX = 4324
POP DX (increment by 2 SP)
POP BX

- stack pointer moves down and then data is stored.

- Des can be a general purpose register, segment register (except CS) or memory location.

↳ XCHG Des., Src:
- This instruction exchanges source with destination.

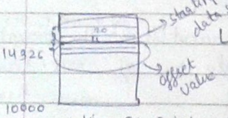
Eg.: XCHG DX, AX ⟵ Value of AX and DX are exchange with each other.

↳ IN ACCUMULATOR, PORT ADDRESS:
it transfer the operand from specified port to accumulator register.
E.g.: IN AX, 0028 H
↳ Location

↳ OUT PORT address accumulator: transfer operand from Ax to Port
e.g. OUT AX

↳ LEA Register: loads a 16-bit register with offset address (load effective address) of data specified by source.
LEA BX, [DI]

↳ LDS Des, src: [Load data seg. address]



LDS BX, [4326H] (offset address)
mov [4328], Al
mov [4329], Ah

earlier DS points at 10000; after this instruction DS = 1120 (starting address of Data segment)

data is refred as starting address of ds.

- it copies a word from two memory location into the register specified in instruction.
- then copies word from next 2 location in DS.
- useful for pointing SI & DS at start of string

↳ LAHF [load AH flag register]
it copies the lower byte of flag register to AH.

↳ SAHF
it copies the contents of AH to lower byte of Flag Register

↳ PUSHF
pushes flag register to top of stack

↳ POPF
pops the stack top to flag register

Flag register
| 0 0 0 0 1 1 1 1 | LAHF → AH = 00000000
11 000000 ← SAHF

② **Bit Manipulation Instructions:**

- testing a zero bit / set or reset a bit
- shift bits across registers

> AND > OR > XOR > NOT > shifting

- AND AL, BL
- OR AL, BL
- XOR AL, BL
- NOT AL

AL = 0100 0111
BL = 0000 1010

Shift: (i) - SHL AL, 1 AL = 0000 0100 (4) *
0 0000 1000 (8) $2^1 \times 4 = 8$
C ← MSB ← LSB ← 0

⇒ mov CL, 04
SHL AL, CL

(ii) SHR
AL, 1 AL = 0000 0100 = (4) *
0 0 0 0 0 0 1 0 = $4/(2)^1 = 2$
0 → MSB → LSB → carry

⇒ mov CL, 04
SHR, AL, CL

Date 27-July Page

(iii) SAL is same as SHL

(iv) SAR

MSB → LSB → carry

SAR AL, 1 [Arithmetic] [in this MSB bit represent the sign]

AL = 1 0 0 0 0 0 1 0

1 1 0 0 0 0 0 1 carry

1 is copied

AL = 0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 1 Carry

* { shift left by n ≈ multiply by 2^N
  { shift right by n ≈ divide by 2^n

Rotate instruction:

ROL rotate over left
ROR rotate over right
RCL rotate over left with carry flag
RCR rotate over right with carry flag

- ROL des, count

ROL AL, 1 C ← MSB ← LSB

AL = 1 0 0 0 1 1 1 0

carry 0 0 0 1 1 1 0 1

again rotate by 1 then 0 0 1 1 1 0 1 0

- ROR MSB → LSB → C

- RCL C ← MSB ← LSB

CF = 0

RCL AL, 1 AL = 1 0 0 0 1 1 1 0

CF=1 0 0 0 1 1 1 0 0

CF=0 (Previous)

- RCR MSB → LSB → C

③ Arithmatic instructions:

Add	MUL
Adc	DIV
Sub	INC
Subb	DEC
CBW	CWD

- ADD des, src

11
+90
101
+30
31

carry is not considered

- Adc des, src

11
+90
100
30
131

carry is considered

- Subb. → considers borrow

- MUL, DIV → multiplication & division always performed with AL.

MOV AL, 03H

MOV BL, 03H

MUL BL

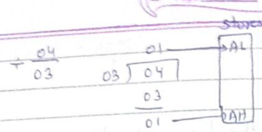
AL = 03
BL = x 03
Ax → 16 bit

If Ax = 1103 } 16-bit
Bx = 2003 }
x
→ 32 bit

then Ax will have LSB (16-bit) + Dx will have MSB (16-bit)
= 32-bit

8-bit	8-bit	16-bit
↓		↓
Dx		Ax

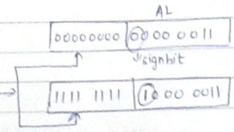
```
mov AL, 04H
mov BL, 03H
Div AL
```



- inc src → increment value by 1
  dec src → decrement Value by 1

- CBW → convert byte (8-bit) into word (16-bit)

```
mov AL, 03 H
CBW
```



performs with AL or Ax

- CWB → convert word (16-bit) to double word (32-bit)

```
mov Ax, 8003 H
CWD
```

Ax = 1000 0000 0000 0011

due to CWD instruction

Dx = 1111 1111 1111 1111

↳ fills with sign bit

④ Program execute transfer:

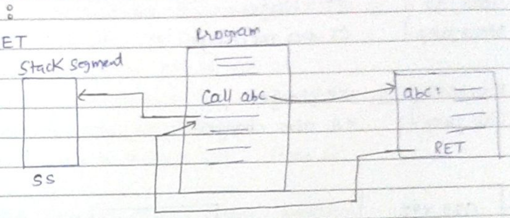
→ DAA : decimal adjust after addition
save value of addition in AL or Ax
and then write DAA ⇒ ans. will be in proper BCD form.

→ DAS : decimal adjust after subtraction
save as DAA but works after subtracting two numbers
⇒ ans. in proper BCD form.

INT → interrupt
INTO → interrupt if OF = 1
IRET → interrupt return

↳ Call label :

RET



When func is called → the add. of next instruction saves in SS and when RET (return) is done, add. from SS is fetched.

↳ Loop label :

```
mov CL, 10
LOOP abc
```

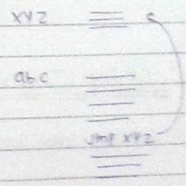
abc :

decrement of CL is done every time loop is run.

↳ Jmp label :

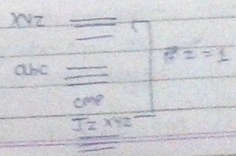
Jump to the label

- when this instruction is run, then the jump label is executed
- unconditional jump



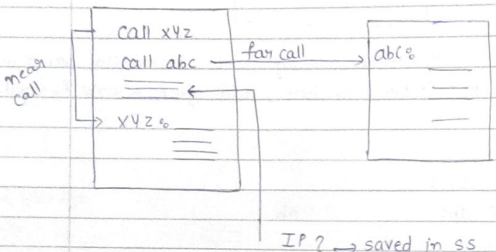
↳ CMP :

CMP des, source → compare through virtual subtraction
JZ label → jump if zero flag = 1



- Near call } → IP changes
  Near jump } → CS does not change.

- Far call } → IP changes
  Far jump } → CS also changes



IP } → saved in SS of this line when 'far call' is executed.
CS }

↳ condition jump instructions
CMP

⑤ String Instruction:

CMPS dst, src → compare

SCAS

MOVS / MOVSB / MOVSW :
- src → data seg
  dst → extra seg
- offset word is stored in

REP → repeat

⑥ Processor Control instructions:
• repeatation happens till CX is 0.

STC → sets CF = 1
CLC → clears CF ⇒ CF = 0
CMC → complement the state of the carry flag CF.
STD → sets direct flag = 1
CLD → sets DF = 1, ie., clears it

→ AAA → ascii adjust after addition.
Packed data / Unpacked data

(2.19) ADDRESSING MODES:

① Implied Addressing modes:
instruction does not have any operand.
for ex - CLC, STD, CLD, STC

② Immediate Add. modes:
One of the operand is the immediate data.
for ex: mov AL, 05H

③ Register Addressing Mode:
destination & source both are registers
for ex: MOV AX, BX

④ Direct Addressing Mode:
direct address is available as part of instruction
for ex: mov DL, [0058H]
- memory address offset value for memory addr. is given in the second instruction directly.

⑤ Reg. indirect add.:
If offset in memory addr. is coming through a register
for ex: mov BX, 0058H
mov AL, [BX]

⑥ Base add mode:

offset value is passing through the Bx register.
for ex: mov AL, [BX] & mov AL, [BP]

⑦ Index Add. mode:
offset value is passing through the SI & DI. (→ same IP, → next instruction IP)
mov AL, [SI] & mov AL, [DI]

⑧ Base Index Add. mode:
BX = 0002 BX+SI = 0102
SI = 0100
mov AL, [BX][SI]
[0102]

⑨ Base index Add. + displacement add. mode:
mov AL, [BX][SI]+24 [0126]

0102
+ 24
0126

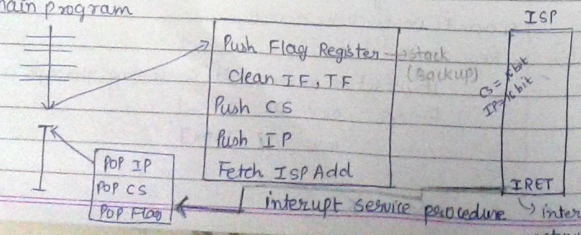
Interrupt (chapter-8)

Hardware
→ NMI, INTR

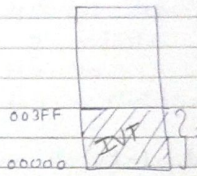
software
→ by writing program we utilize some error

Execution / Prog. errors
→ divide by 0
→ single step (execution)
→ break point
→ overflow occur
→ NMI, INTR

When interrupt will occur:
main program



- 1K byte memory is covery by the only ISP address

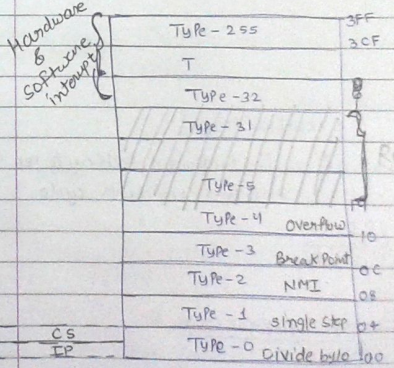


To store one ISP address we required 4 byte (location) [IP (16 bit) → 2 byte, CS (16 bit) → 2 byte]

So 8086 processor can handle 1000/4 interrupt at a time
= 256

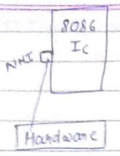
The table which is stored the ISP address, that is called as IVT (interrupt vector table)

IVT (interrupt vector table):



- Type-0 to Type-4 are predefined for 8086 processor
Type-0 : divide by 0 CS = 0002h IP = 0000
Type-1 : single step (-t) CS = IP =
Type-2 : NMI (hardware interrupt)

Date 2-Aug-20
Page



Type-3 : Break point (-t5)
Type-4 : overflow

- If 2 interrupts occurs at a same time then a priority list is there for 8086 :-
  INTO / INT / divide error
  NMI
  INTR
  single step interrupt
- the same priority list is considered by other ic's.
- reset is also an interrupt and has highest priority in advance ic's.

(Chapter-1 PARHAMI)

Performance :

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

execution time consist of = instructions x clock cycle per inst x second per clock cycle

$$= \text{Inst's} \times \frac{\text{CPI}}{\text{clock rate}}$$

To increase performance of processor :

- Instruction Set Architecture (ISA)
  no of cycles per inst'n can be decreased.
- Designer of processor does not have any control over no. of inst'n written in a code

Date
Page

- second per clock cycle depends on technology as freq x power. So if oscillator freq. is fixed power consumed also fixes
  ⇒ we have a limitation of (2-6 GHz) in this.
- To increase no of cycles per instruction, super scale is used ⇒ no of instructions in single cycle is increased
- performance = $\frac{\text{clock rate}}{\text{CPI}}$

  CPI (cycle per instruction) ≅ 1 (ideally)

  IPC → instruction per cycle

  performance = IPC x clock rate

AMDHAL'S LAW : (Performance enhancement)

$$\text{Speed up} = \frac{1}{\text{sequential part} + \frac{\text{Parallel Part}}{\text{No. of Processor/function al unit}}}$$

instruction depends on other

independent of all instructions

- as code can not be 100% parallel part ⇒ no. of cores on a particular chip is limited.

Performance = $\frac{\text{clock rate}}{\text{avg CPI}}$

L→MIPS → million instructions per second

Date Page

Example A processor chip used for 1 application in which 30% of execution time is spent on floating point addition, 25% of time on floating point multiplication, 10% of time on floating point division. For enhancement of performance design team comes up with certain suitable solution. Find out which enhancement soln is best.

(I) Re-design a floating point adder to make it twice as fast

(II) Re-design the multiplier which becomes 3 time faster

(III) Make your divider 10 time faster.

Sols $f_A = 0.3$

$f_M = 0.25$

$f_D = 0.1$

$f_A = \frac{1}{0.7 + 0.3/2} = 1.176$

$f_M = \frac{1}{0.75 + 0.25/3} = 1.2$ (best)

$f_D = \frac{1}{0.9 + 0.1/10} = 1.098$

Date Page 3 - Aug 20

Ques:

Class	600 MHz M1 CPI	500 MHz M2 CPI
(Floating) F	5	4
I	2	3.8
(Numerical) N	2.4	2

(i) What is the peak performance for M_1 & M_2

{Peak} Performance

$= \frac{\text{Clock rate}}{\text{Avg. CPI}} = \frac{600}{5+2+2.4} = \frac{500}{3.13}$ (Performance for M_1)

$M_1 \rightarrow$ Peak Performance $= \frac{600}{2} = 300$ MIPS (million instructions per sec) (divided by lowest)

(assume all of class I)

$M_2 \rightarrow \frac{500}{2} = 250$ MIPS (all of class N)

(ii) If 50% of all instructions executed in certain applications belonging to class N and rest are divided equally amount class F and I. Find out which machine is faster by what factor?

Sols

Avg. CPI M1 $= \frac{5}{4} + \frac{2}{4} + \frac{2.4}{2} = 2.95$

Avg. CPI M2 $= \frac{4}{4} + \frac{3.8}{4} + \frac{2}{2} = 2.95$

$\frac{\text{M1 Performance}}{\text{M2 Performance}} = \frac{600/2.95}{504/2.95} = 1.2$

M1 Performance = 1.2 M2 machines

∴ M1 is faster than M2 by 1.2

ISBCE178

M1

(iii) M1 Someone want to ~~plant~~ redesigned for better Performance with assumption of Part-2, which of the following redesign greater ~~effect~~ impact on performance.

(a) faster floating point unit which will double speed.

AVG. CPI $= \frac{2.5}{4} + \frac{2}{4} + \frac{2.4}{8} = 2.325$

performance $= \frac{600}{2.325} = 258$

(b) add second ALU to reduce Class I; CPI to 1.2

AVG. CPI $= \frac{5}{4} + \frac{1.2}{4} + \frac{2.4}{2} = 2.75$

performance $= \frac{600}{2.75} = 218.$

(c) Used faster logic block that allows to increase the clock rate upto 25MHz same CPI.

Performance $= \frac{750}{2.95} = 254$

(a) modification in (a) is highly impact on machine M1.

(d) the current CPI given as included the effect of cache misses at avg. rate of 5%. each cache miss will impose penalty of 10 clock cycle. Redesigning of M1 with larger cache that can accomodate large no. of instruction and ultimatly reduce miss rate from 5% to 3%. Compare performance with any above of the modif

Sol:

5% × 10 = 0.5

3% × 10 = 0.3

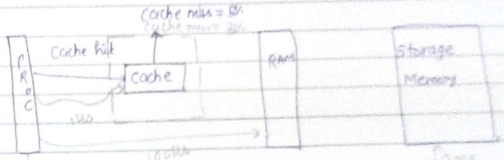
improvement = 0.2

avg. CPI $= \frac{(5-0.2)+(2-0.2)+(2.4-0.2)}{3} = 2.75$

performance $= \frac{600}{2.75} = 218$

its modification is equal to Date(II) CPI to 1.2.

CACHE:



if data is not available in cache → cache miss

- Cache size is fixed, miss rate less

- 0.2% improvement ⇒ performance is good ⇒ avg CPI is decreases ⇒ 0.2 is subtracted to CPI

Q. (e) device and application that would run faster on M1 then M2

Sol:

	M1	M2
F	5x	4x
I	2y	3.8y
N	2.4(1-x-y)	2(1-x-y)

M1 > M2

$\frac{600}{5x+2y+2.4(1-x-y)} > \frac{500}{4x+3.8y+2(1-x-y)}$

$1.2[4x+3.8y+2(1-x-y)] > 5x+2y+2.4(1-x-y)$

$4.8x + 4.56y + 2.4 - 2.4x - 2.4y > 5x + 2y + 2.4 - 2.4x - 2.4y$

$2.56y > 0.2x$

$12.8y > x$

$x < 12.8y$

(d) The current CPI given has included the effect of cache misses at average rate of 5%.