R_1 =

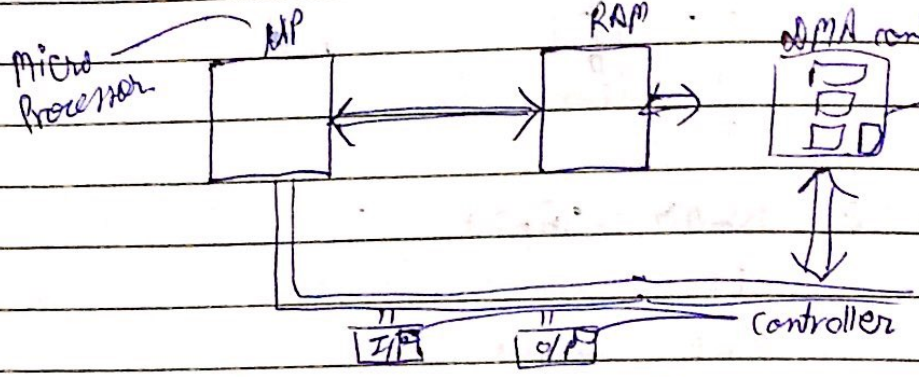
create_Playlist input, R_1
Add input1, input2

input (1) → Playlist no
input (2) → Song

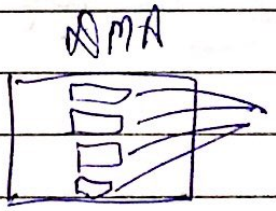
6/oct/2017

Output decide will send an interrupt to the processor that it is ready to receieve the data

Interrupt driven Program
Giving charge of input and output to another device called DMA (Direct Memory Acess)



- No input/output can acess the memory directly
- There is a controller with every input/output device.
- Procesor checking input/output device and wasting its time so now we have DMA controller.

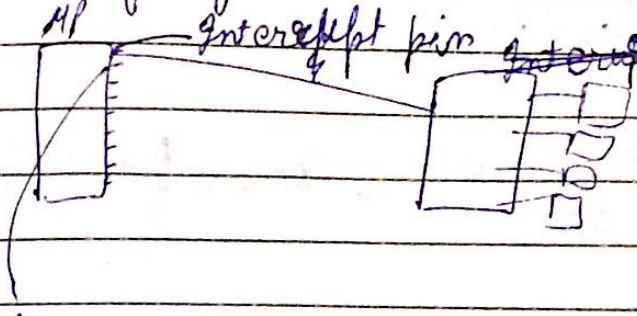


Initialise DMA controll register.
Processor send information to DMA controller.

DMA need

1. Address of input / output device.
2. Address of the Memory location.
3. Number of byte.
4. Read / Write Operation.

Input / output devices are given priority in for data bus axes (if Processor need to axes data bus) No. of cycle lost in the cycle → Cycle Stealing.



Interrupt Acknowledge

Interrupt vector table

ISR Interrupt Service Routine

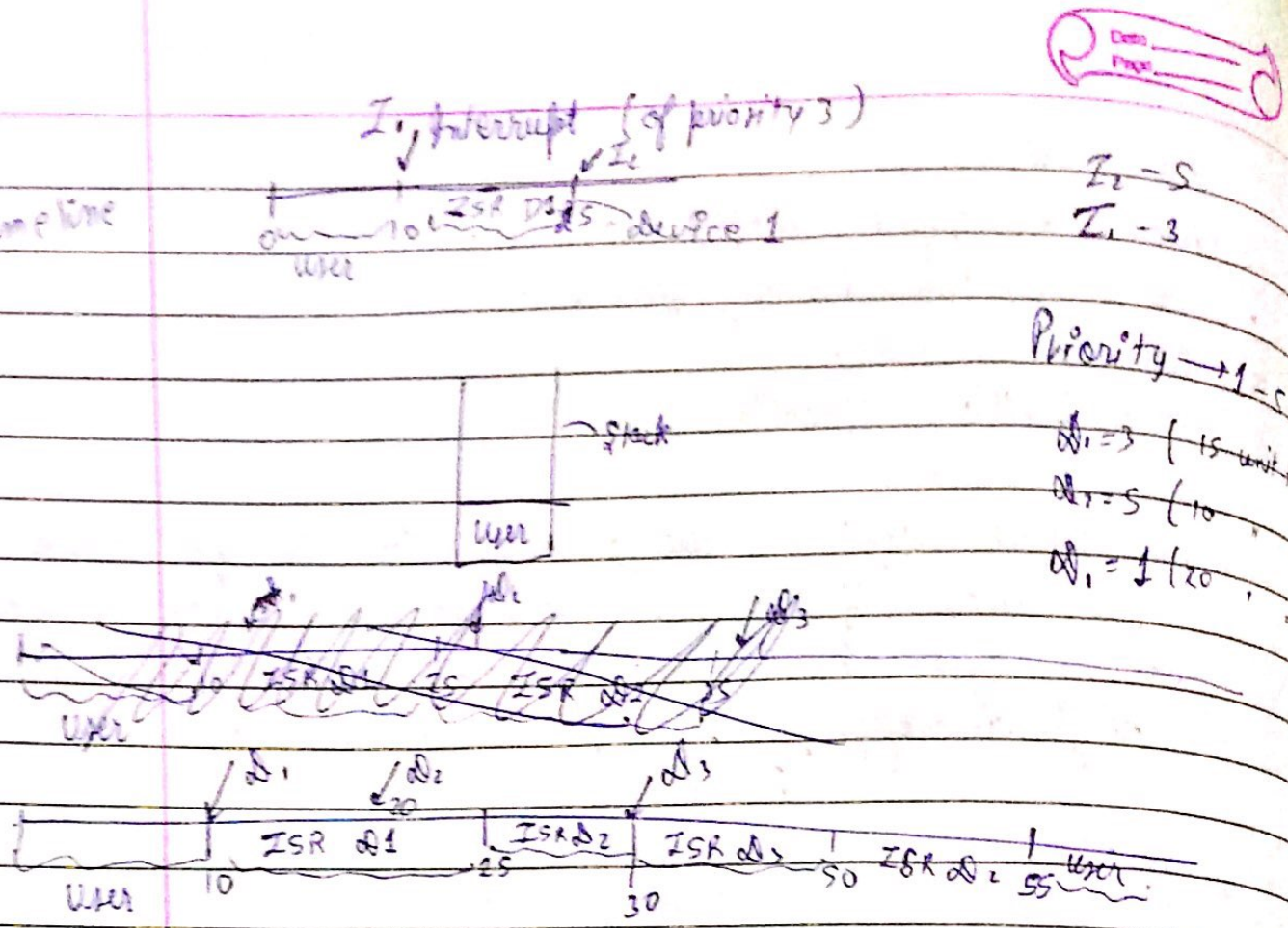
Each device will be handeled in a seperate panel. by functions

when device sends small no. on the bus. Processor check machine. Finds the address in the table. 2, 3 device $ can be shared by same ISR.

Processor running a program. An interrupt is there so it will store all the values so that it can resume its task after working with ISR. Go to ~~ER~~ ISR will process DMA controller may be initialised then come back

overheads when dealing with interrupt.

Put back all the store data in there respective register and resume there task.



Trap

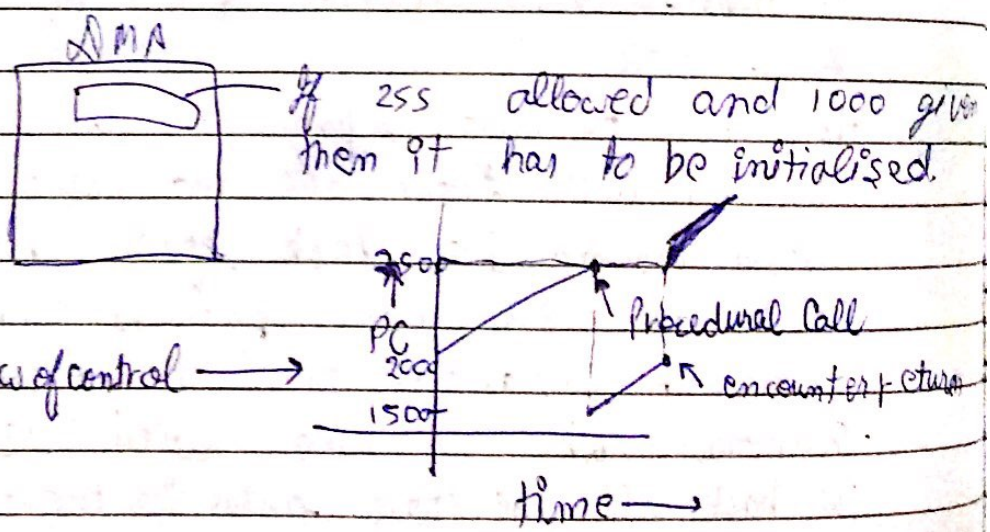
Trying to access memory address which is given to another program → SIGSEV

Trap Handling.

We have a trap handler.

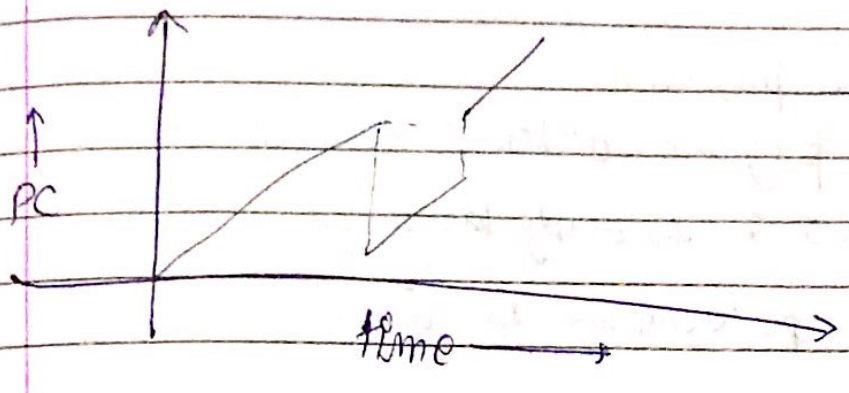
Traps are synchronous & Problem will occur at the same time

7/oct/2017



PC now changed

Interrupt → Is there because of DMA



It is talking about only PC, it is not talking about OS.

- ALP (Assembly level Program)
  - → ~~If~~ you have access to the machine, the hardware can check the flags.
  - → These are faster to execute
  - → For embedded application where memory is less program is written using ALP.
  - → Compiler is the form of a translator converting ~~one~~ high level language program to assembly language program. ~~If~~ Compiler may have a assembler to convert the program to machine language.
  - → After conversion instruction are read one by one.
  - →

Harddisk → RAM → Processor → Fetche Decoded Executed
When one instruction finishes it goes to next instruction.
This is known as interpretation.

Translater	Interpreter
→ Compiler are translater	→ Every sentence interpreted → Instruction by Instruction → Fetch Decode Execute

In 100 sec of a program.

10% of the program utilising 90% of the time

↓

Rewrite this part in assembly language

↓

known as performance tunning.

	Program Time (yrs)	Execution Time (sec)
ALP	50	30
HLL	10	100

Before tunning

Critical	10%	1 yr	90
Rest	90%	9 yr.	10

Critical 10% $1 + 50 \times \frac{1}{10} = 6$ yrs

Rest 9 yrs

Pseudoinstructions (Assembler Directives)

Pseudoinstruction

from X EQU 100 → to → Machine language

Go through the list of Pseudoinstruction

```
ADD R1, R2
CMP #0
JNE LOOP
≡
ADD R1, R3
CMP #0
JNE LOOP
≡
```

```
FN   MACRO
     ADD R1, R1
     CMP #0
     JNE LOOP
     ENDM
≡
FN   ← MACR Call
≡
FN
```

Defination

Macro Expansion: As soon as the program find this it replace FN with the whole defination of FN

- What is the difference between Macro and Procedure.
  - → Avoid procedure because of the overheads included in it.
  - → Macro is not expanded during runtime, macro call will not take place during call time. It will change as soon as the assembler converts to machine language.. Macro is has been converted at this time Expansion is taking place when assembler is converting to machine language

Macro	Procedure
→ during assembly time	→ During runtime
→ If called 5 times then 5 copies of the macro	→ only one copy in the address
→ No return statement.	→ There is a return statemen

ILC

0 ADD R1, R2 ← Assembler takes instruction find opcode and convert to Machine language.

2 JMP L1 ← Forward referencing (don't know the value of L1 Right now)

5 CALL B

L1 stored in the Symbol table.

L1: OUT R3, 0

HLT

Two pass assembler

Going to read your ALP program two times.

Every time it reads it is called a pass.

Read one time ⟶ Pass 1.

ILC (Instruction Location Counter)

I Pass

Opcode Table

Symbol Table

Macro Table

Opcode Table

Opcode	Oprand 1	Oprand 2	Hexadecimal code for this instruction (Hex code)	Length	Type

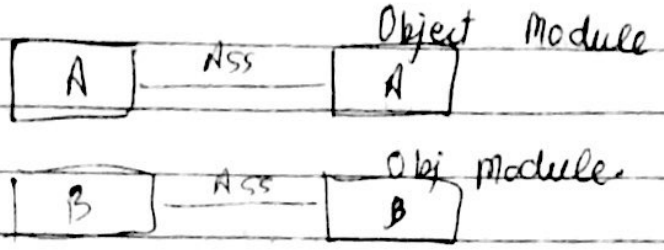
→ If a macro encountered then it is ~~expo~~ stored in macro table.

→ Clueless about B.

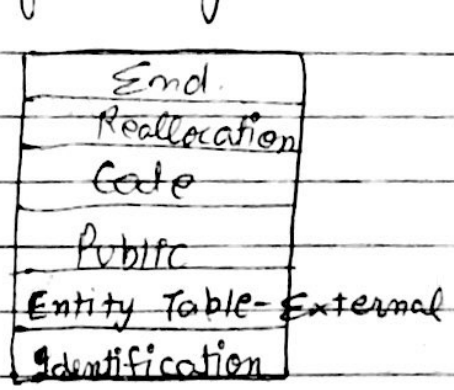
II Pass

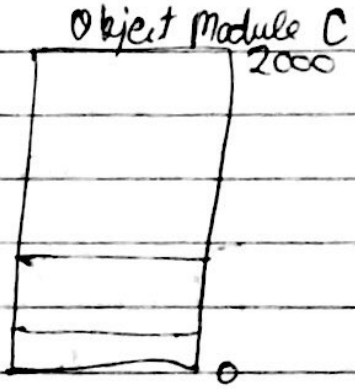
It will check the table and execute it.

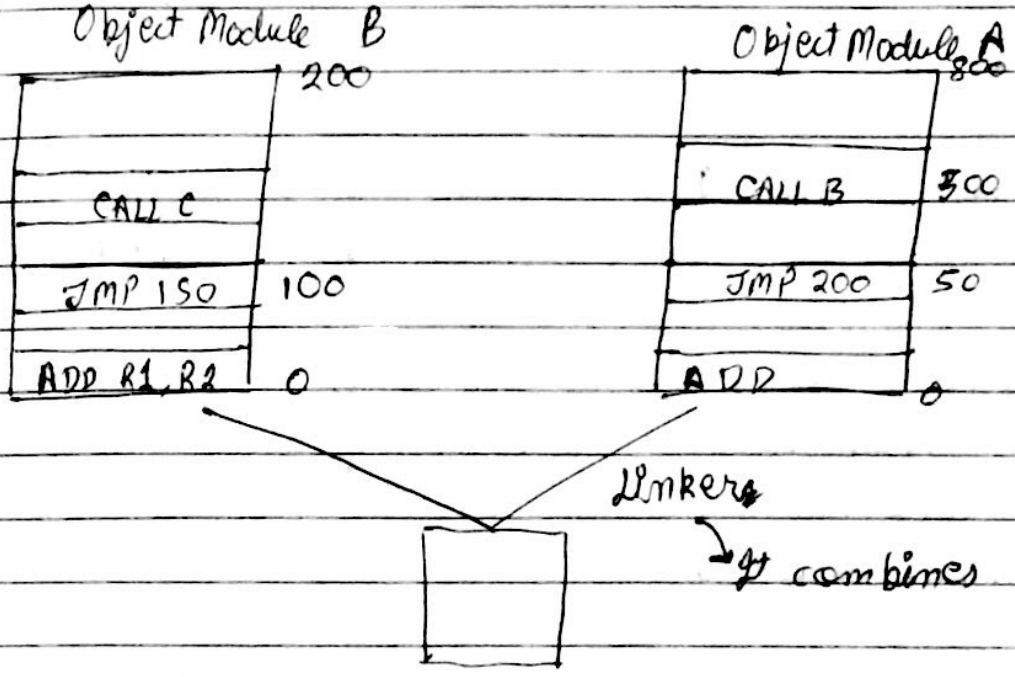
→ At the end of PASS II all the ~~of~~ thing complete and object module is ready.



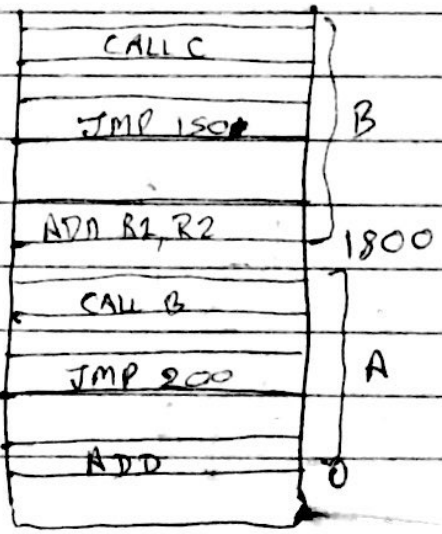
Structure of an object module







LOADER → LOADS.



LINKER LOADER
Performs both the task.

Relocation problem
External Reference problem

→ Relocation Problem : Address (base address) and jmp statement are according to the address (base addres) of their module

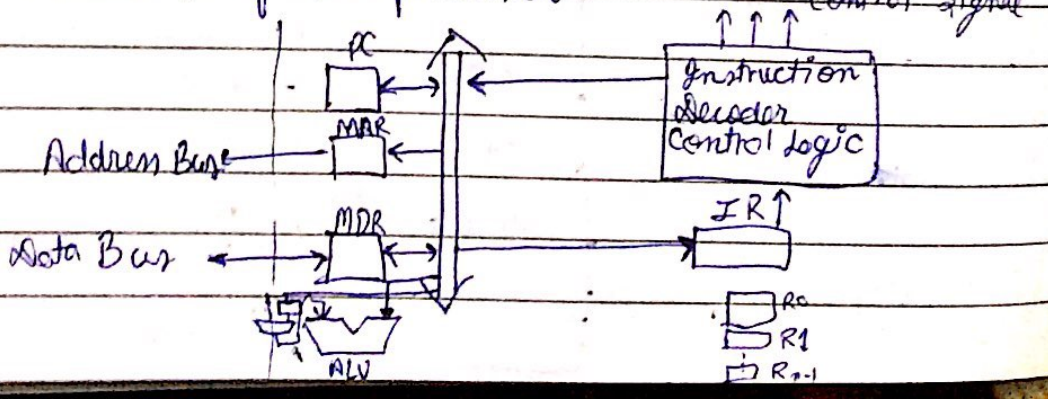
→ External Reference problem : Module A does not know the address of rest of the module and vice-versa

Module Name	Length	Starting Address
A	800	1000
B	200	1800
C	500	2000

Thus JMP B. converts to JMP 1800
JMP 150 converts to JMP 1950.

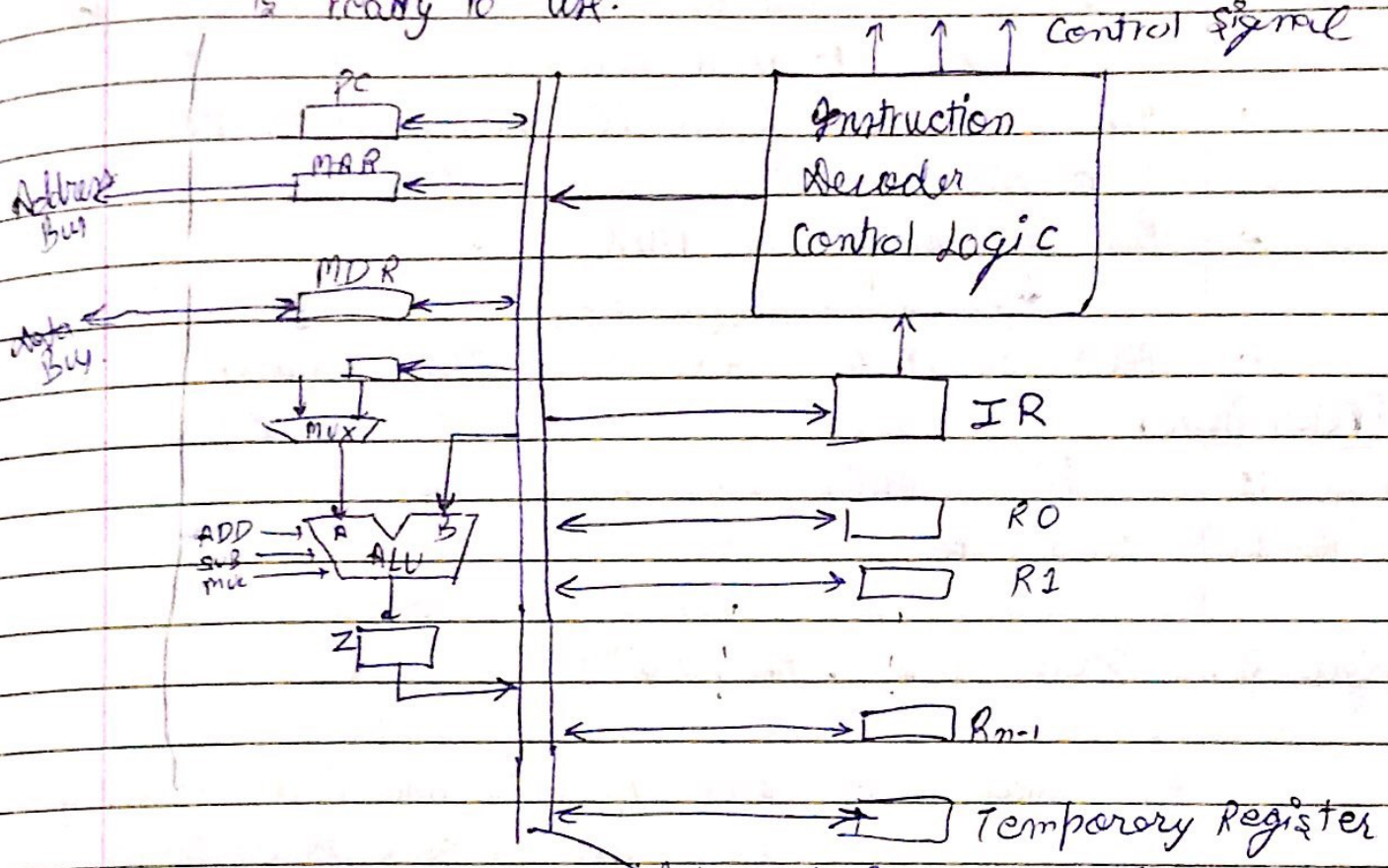
14/Oct/2017 From memory to processor → Fetch.
Fetch - Decode - Execute cycle
Processor Unit.
Micro Architecture.

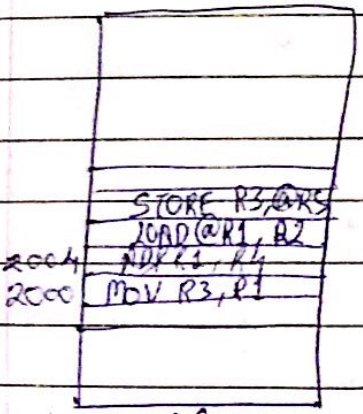
PC → MAR → Address Bus → location accessed → MDR
Internal bus of the processor



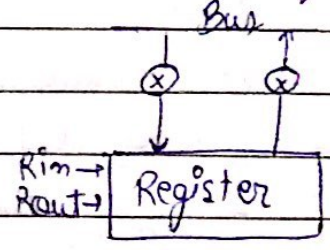
Only one bus inside processor so only one can access the bus.

- If ALU is working on two operands
  Bring R0 store it in the Temporary register in ALU then bring R1, add R0 and R1 store it in the Temporary register till the bus is ready to use.





Each Register is having an in control signal and an out control signal



Rin: If this is high whatever value in bus will be stored in the register.

Rout: If this is high whatever value in register will go into the bus.

| PC out | → control signal for the Program counter which is allowing it to transfer data to the bus

16 bit bus → 16 wires.

Control unit decides in or out.

→ MAR in activated then Read signal needs to be generated. Read operation is for the memory. It will tell to read memory.

→ Information ~~to~~ from PC has gone to MAR as well as ALU

→ Pass 4 through the MUX.
↓ size of one instruction.

→ MUX select 4 add it, Zin enabled.

Fetch Phase

Clock Cycle [1] PCout, MARin, Read, Select 4, Add, Zin

Data bus 2000 ~~for~~

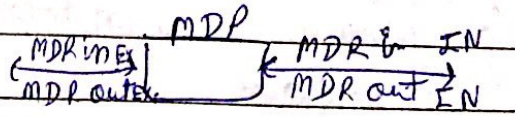
MDRin-E

Clock Cycle [2] Zout, PCin, Yin, / WMFC

* First check cache then check in main memory, if not in cache then program send to cache since accessing the main memory takes time but cache can be accessed easily.

Wait for Memory function Complete.
↓ Waiting for data in MDR from memory.

MDR → Bidirection communication with Internal as well as external bus



Clock Cycle 3 MDRout , IRIN

Now Fetch ~~for~~ phase is completed.

MOV R3 R1 → This data has been converted to binary by decoder, control signals have been generated for this instruction.

Decode Phase

Clock Cycle 4. ~~PCout , ALUIN , MARIN, MDRIN~~
Execution. R3out , R1in , END

Next Fetch Cycle will be executed

This was the micro program or the micro routine for the instruction.

Fetch Phase

⋮ (Same)

Now next instruction has been decoded by decoder, control signals have been generated for this instruction

ADD R1, R4

Decode Phase

Clock Cycle R1 out , Yin
Clock Cycle R4 out , Select Y, Add , Zin
Clock Cycle Zout , R1 in , END

Fetch Phase

⋮ (Same)

LOAD @R1, R2

[IR] ← [[PC]]
[R2] ← [[R1]]

Decode Phase

Clock Cycle — R1 out, ~~MDR in~~, MAR in, Read Signal Generate
Clock Cycle — ~~MDR~~ MDR in, WMFC
Clock Cycle — MDR out, R2 in, END

Fetch Phase

⋮ (Same)

STORE R3, @R5 — [[R5]] ← [R3]

~~Clock Cycle~~ — ~~R3 out, MAR in, MDR in~~
~~R5 out, MAR in~~

Decode Phase

~~Clock Cycle~~ — ~~R3 out~~, MDR$_{IN}$ — Process of accessing the memory location done first
Clock Cycle — R5 out, ~~MAR~~ IN

Clock Cycle — R5 out, MAR in, Write
Clock Cycle — R3 out, MDR in, MDR out-EX, WMFC, END

RISC — CISC Instruction
↖ ADD @R7, R8 → HW

16/oct/2017 — ~~R7 out, MAR in, Read Signal, WMFC~~
~~MAR out~~

Decode Phase

~~Clock Cycle~~ — ~~R7 out~~, MAR$_{IN}$, ~~READ~~
~~MDR out~~ MDR$_{IN}$ ~~EXT~~, MMF

Clock Cycle R7out, MARin, READ
Clock Cycle MDRin-Ext, WMFC R8out, Yin
Clock Cycle ~~MDRout, Yin~~ MDRout, select Y, Add, Zin
Clock Cycle ~~R8 out, SELECT~~ Zout, R8 in, END

ADD R1, R2
=
JMP L0 If L0 at 46 and JMP L0 at 20 then offset is 26
≡
L0: ≡

- OS will load the program wherever it finds space, it may vary everytime.
- offset remain same

JMP 55
~~PCout~~
Decode Phase

~~PCout, Yin~~
Clock Cycle Offset field of IRout, select Y, Add, Zin
Clock Cycle Zout, PCin, Yin, END.

In Yin we already have PC value thus
~~PCout, Yin~~
Decoder is connected to ALU.
If L0 above JMP statement then offset will be negative.

(Branch < 0) 26

Decode Phase

Clock Cycle Offset field of IRout, Select Y, Add, Zin, If N=0 END
Clock Cycle Zout, PCin, Yin, END

JZ 26

Decode Phase

Clock Cycle Offset Field of IRout, Select Y, Add, Zin, If Z=0, END
Clock Cycle Zout, PCin, YIN, END

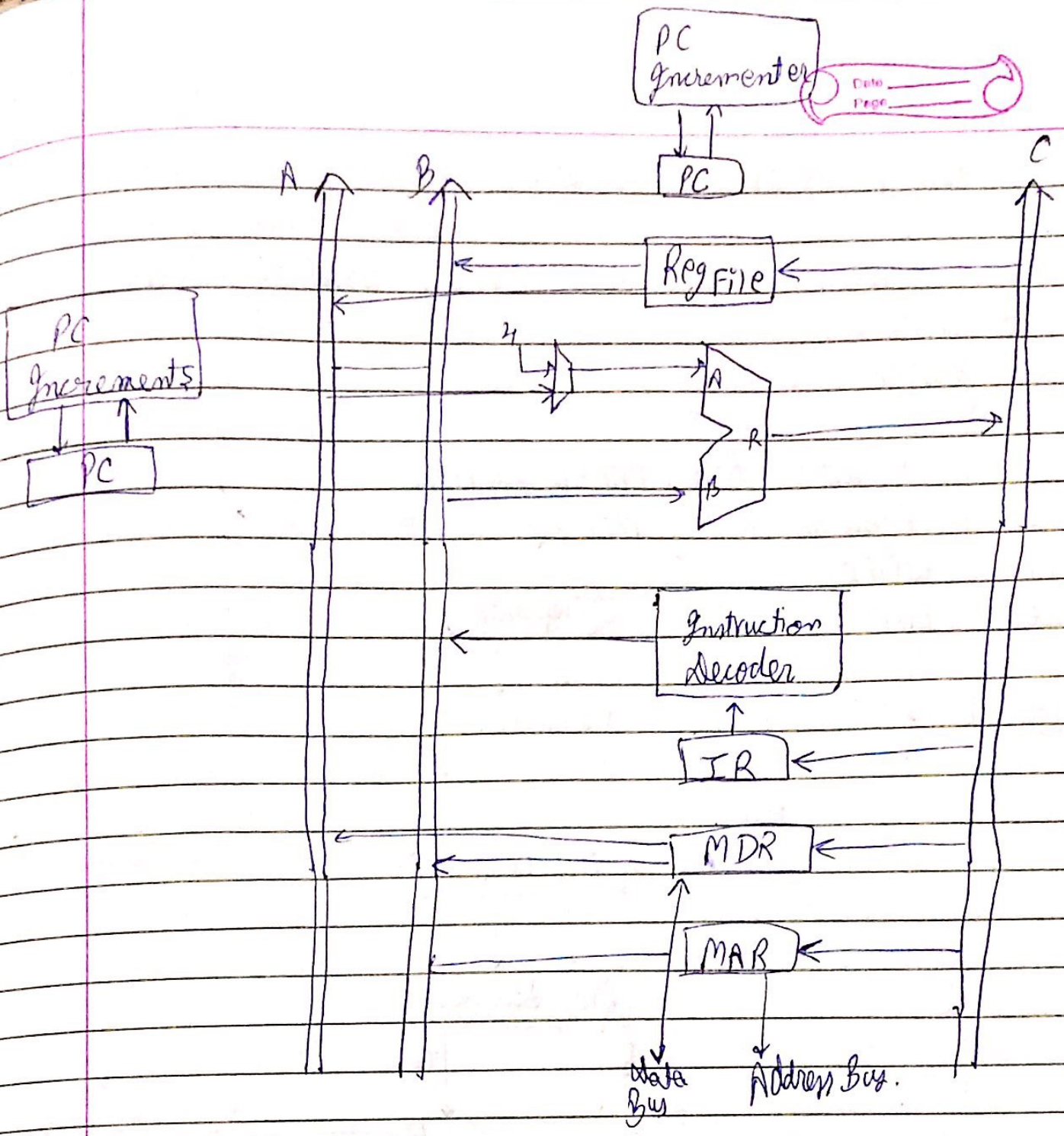
END always starts a new fetch cycle.

Tables
↓
Files (collection of records)
└ fixed

23/Oct/2017 Three Bus Architecture

Fetch Phase

1. PCoutB, R=B, MAR_{IN}, Read, IncPC
   ↓ input given in multiplexer.
2. WMFC
3. MDRoutB, R=B, IR_{IN}



Decode Phase

Add R3, R4

1. R3outB, R4outA, SelectA, Add, R4IN, END.

Decode Phase

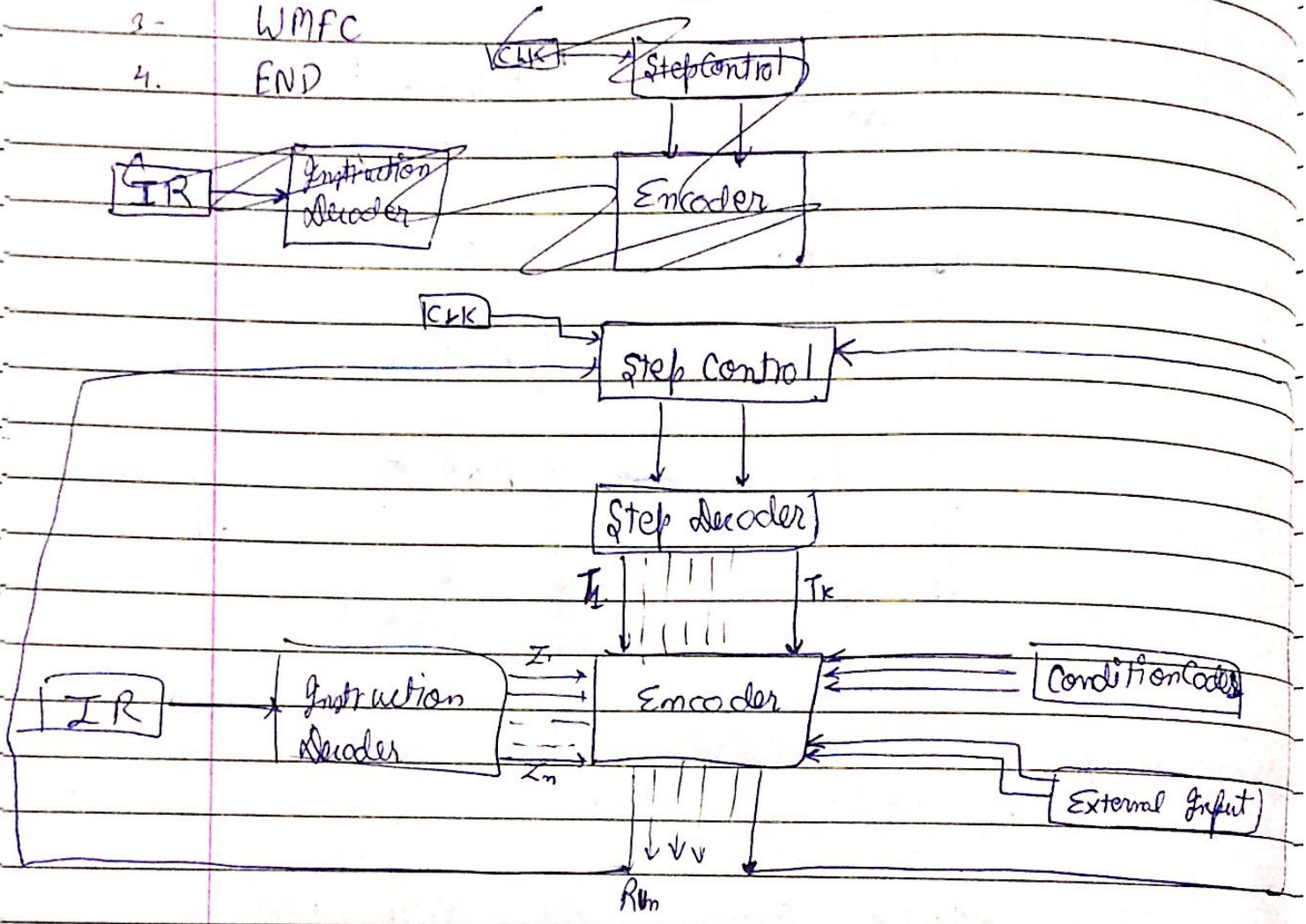
LOAD @R3, R4

1. R3outB, SelectA (R=A), MARIN, Read
2. WMFC
3. MDRoutB, R=B R4in, SelectA, Add, END

Decode Phase Store R7, @R3

1. ~~R7 out, Select A (B=A), MAR IN, Read~~
2. ~~WMFC~~
3. ~~MDR.~~

1. R3 out B, R=B, MAR IN, ~~Write~~
2. R7 out B, R=B, MDR IN, Write
3. WMFC
4. END



Interrupt Service Routine
Check External Input
Check WMFC (this is also an external signal)
At a given point only one high either $T_1, T_2 \ldots T_n$

Run signal Generates that is it enable different timing signal may be T1 becomes T2
WMFC signal Run = 0 Halt If in T3 will remain in T3.
After all the instruction end generated Result took place.

Let 1 → 7 steps.
2 → 10 steps.

Size of decoder: 4 by 16 decoder.

25/oct/2017

ADD Reg, Reg.

1. Zin
2
3
4
5 Zin, End

ADDI Reg, #N

1 Zin
2
3
4
5
6 End.

LOAD

1 Zin
2
3
4
5 End

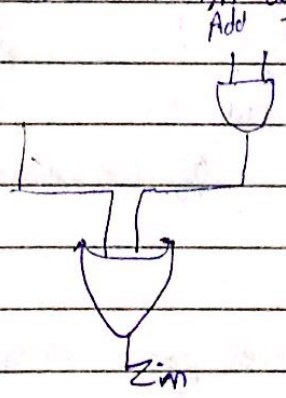
Store

1 Zin.
2
3
4
5
6
7 End

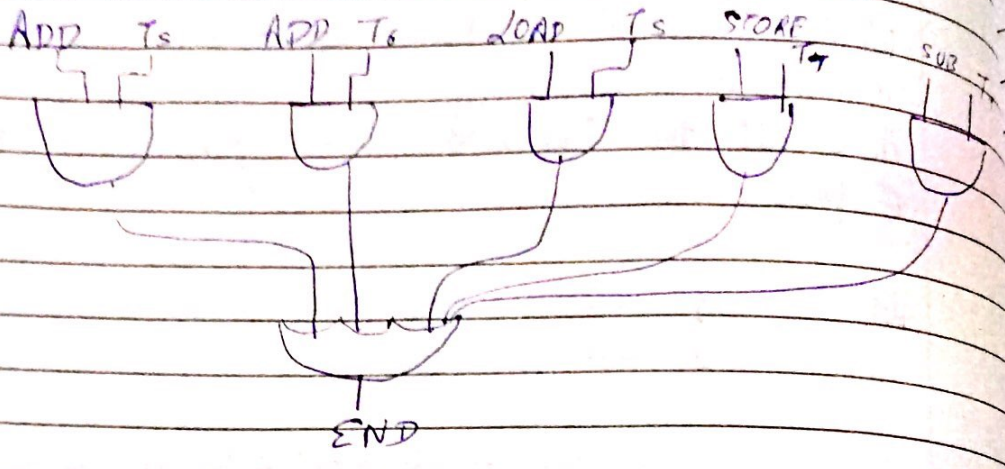
SUB

1 Zin
2
3
4 End

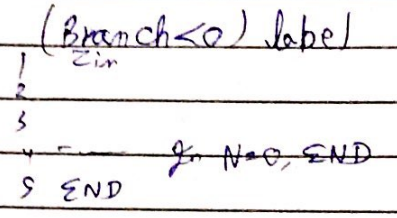
Zin activated in T_1 in all instruction.

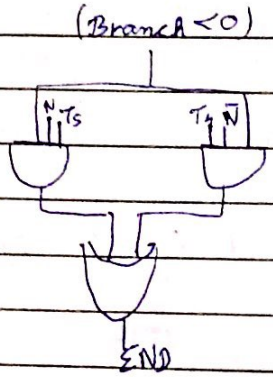


for end



If there are two end like





Maximum No of 7

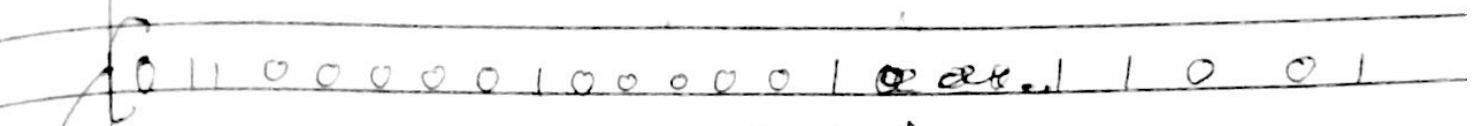
Decoder 3 by 8

Hardwired logic : If we want speed we go for Hardwired logic.

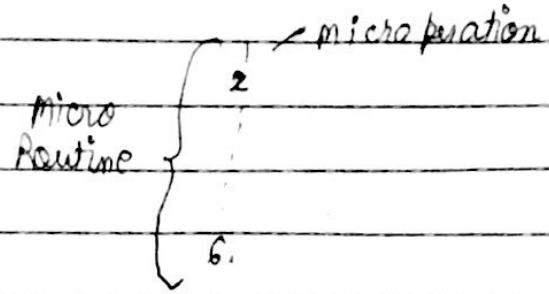
Alternate for Hardwired logic is microprogrammed control.

PC_{IN}, PC_{out}, MAR_{IN}, MAR_{out}, MDR_{IN}, MDR_{out}, Y_{IN}, Y_{out}, Z_{in}, Z_{out}, $R0_{in}$, $R0_{out}$, ..., R_{in}, R_{out}, ADD, SUB, Read, Write, Select 4, WMFC, END

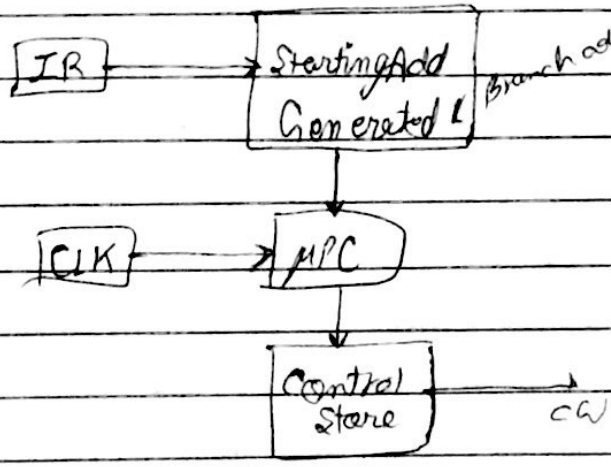
Say 1 step of Add

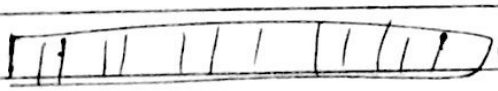


Control Word



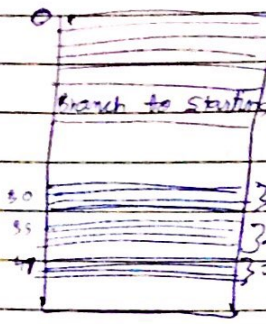
All the instruction in MicroRoutine will be stored in control Store.



 → All instruction

Whenever 1 ate microRoutine corresponding its instruction will be activated.

Fetch cycle is same for all.



0 PCout, MAR_{IN}, Z_{IN}, ADD, select 4

1

2. MDR out, R_{in}

3. Branch to starting add of microroutine

!

20 $R3_{out}$, Y_{in}

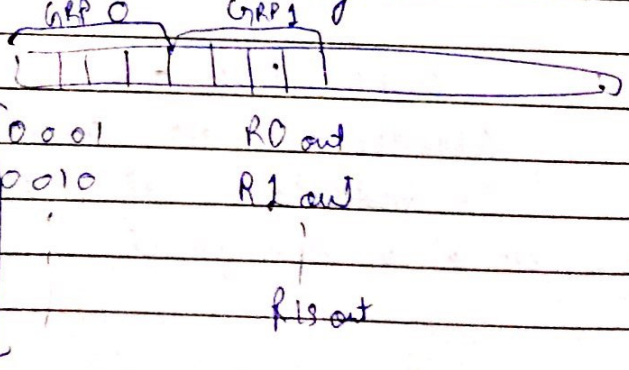
21 $R4_{out}$, Select Y, Add, Z_{in},

22 Z_{out}, $R4_{in}$, JMP to microroutine at 0.

Groups will be formed such that signals become mutually exclusive ~~fi~~ like Read write only one will be high at a time

If 12 signals then 4 bits needed

We can group all the out signals together from only so one will be high at a time.



In horizontal format 16 bits needed if 16 operation are there
In Encoded format 4 bits needed.

1 Pout
2 Zin
3 MARin

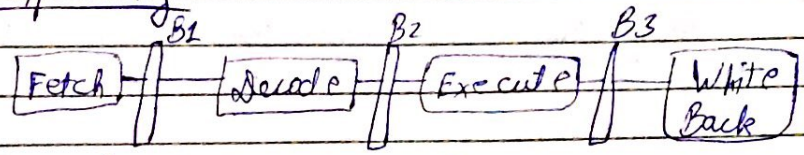
16

0000
0001
0010
.
.
.
1111

Micro Program Control.

27/Oct/2017

Pipelining



Interstage Buffers → B1, B2, B3.

5ns 2ns 3ns 2ns

Clock Cycle 3 : Decoding 3rd Instruction
Executing 2nd
Storing of 1st.

During clock cycle 3 whatever in clock cycle 2 will be there in buffer that is B2 buffer has decoded Instruction 1.
B1 have fetched instruction of no 2.

CLK → 1 2 3 4 5 6 7 8

1.) I_1 has been Fetched
2.) I_1 will be decoded and I_2 will be fetched.
3.) I_3 has bee fetched

CLK	1	2	3	4	5	6	7	8
I_1	F	D	E	W				
I_2		F	D	E	W			
I_3			F	D	E	W		
I_4				F	D	E	W	
I_5					F	D	E	W

T = 5ns

If no pipelining then more time taken.

Time taken — Latency

Pipeline

After every 5ns instruction comes up.
Total time taken is still 20 ns.
1 instruction every 5ns.

$\frac{1}{5 \times 10^{-9}}$ in 1 sec.

200×10^6 instruction/sec.

~~Throughput~~

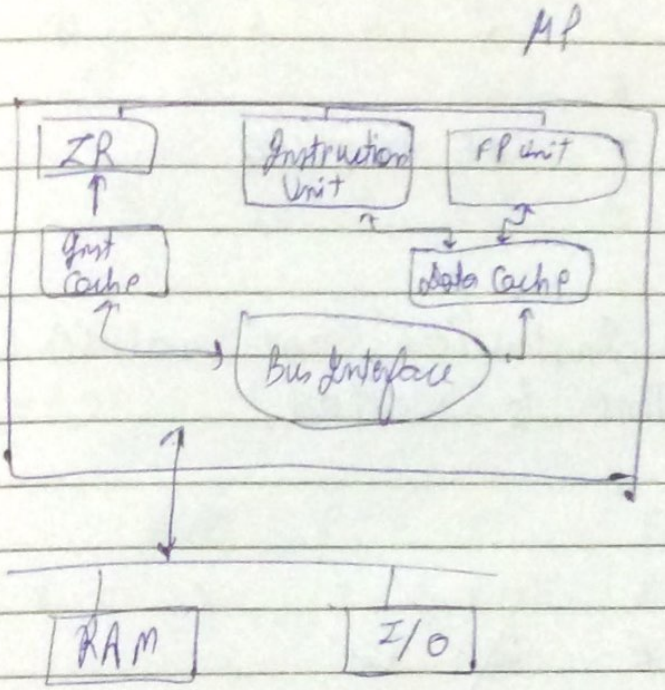
CLK	1	2	3	4	5	stall 6	7	8
I_1	F	D	E	W				
I_2		F	D	E	F	E	W	
I_3			F	D	—	—	E	W
I_4				F	—	—	D	E W

Pipeline has been stalled

Want to access memory at ~~same time in same~~ same cycle then these are called Hazards and for this pipeline is stalled

→ Cache is existing so that memory access takes only one clock cycle.

Cache miss →

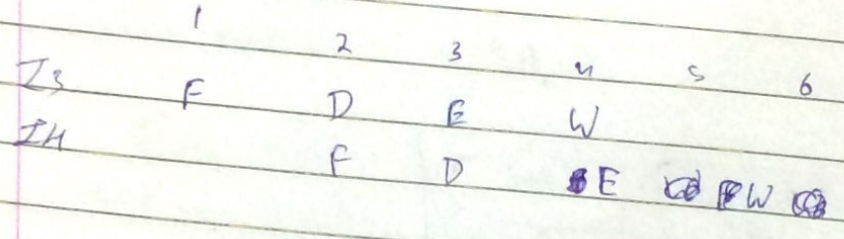


30/Oct/2017

Non-availability of data causes stall

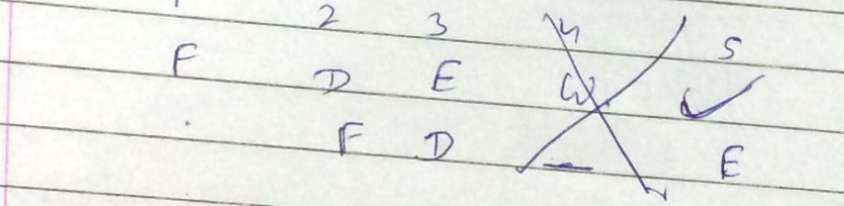
Hands
1. Data Hazard
2. Control
3. Structural

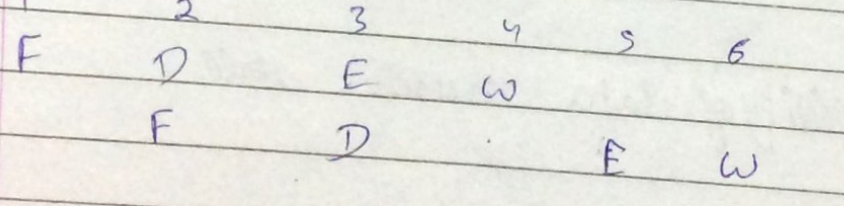
```
main ( )
{
    int x, y, z = 10;
    x = 2 * z  ⟶ MUL R1, R2, R3
    y = 3 + z  ⟶ ADD R2, R4, R5
    .
    .
}
```



Cache miss → Instruction not available in cache so pipeline is stalled.

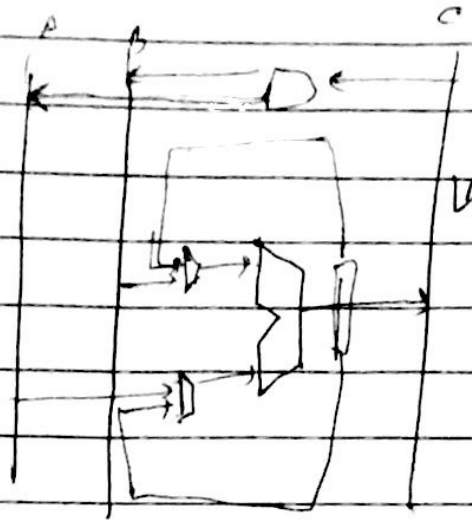
```
x = 2 + 3
y = 3 + x
```





RAW kind Hazard ⟶ Read-after-write Hazard.
WAW
WAR

after 3 bus architecture.



Operand forwarding so that stalling is not needed.

F D E W

F D E W

From output Buffer to input buffer

LOAD R2, 10(R3)
ADD R1, R3, R2.

F D E W

F D E W

Write after Write Hazard.

F D E M W

F D E W → Delay Writing Stage
Put a Stall over here.

MUL R1, R2, R3 NOP → No Operation.
ADD R3, R4, R5 NOP
SUB R7, R8, R9

We can reorder the instruction

For RAW

→ Stalling
→ Decoder checking hazard and hardware exists (Operand Forwarding)
→ No Operation when compiler detect error but no hardware. Then it can also go for reordering
→ Renaming of registers also helps.

1/November/2017

MAR_{IN}, PC_{OUT}, C_3, C_4, C_5, C_6, C_7.

C_3 010
C_4 001
C_5 100
C_7 110

2 1 1 1

ADD R1, R2, R3
SUB R1, R4, R3

Instruction / Control Hazard.

Split Cache < Data / Instruction

If there is a cache miss then instruction taken through memory which took time.

```
I1   F   D   E   W
I2       F   F   F   D   E   W
                 F   D   E   W.
```

Checking main memory
If ~~fetch~~ (cache Miss found)

Branch Instructions.

```
loop   ADD  R1, R2          ADD R1, R2
       DCR  R3              JMP label
       CMP  R3, #0
       JNE  loop.           label.
```

We don't know the target address.

Uncondition Jump

Changes the flow of the program these are called. Branch Instruction.

```
        1   2   3   4   5   6   7
I1      F   D   E   W
I2          F   D   E   W
I3              F   D   E   W
I4                  F   D   E   W   X
I5                      F   D   E   W   X
                            F   D
```

Remove wasted from the pipeline.

For branch instruction address is available at the end of Execution.

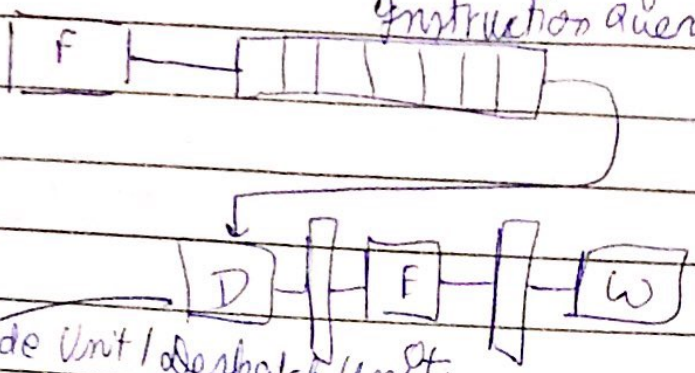
Branch Penalty : Task of removing execution due to Branch Instruction.
Branch Delay Slots

```
F   D   E   W

F   D   E   W
```

Instruction / Control Hazard.

Instruction Queue and Prefetching.



Data Cache Miss

Here in Fetch unit it is checked if the instruction brought is a branch instruction or not

	1	2	3	4	5
I_1	F	D	E	W	
I_2		F			D
I_3			F	D	E
				F	
					F

Branch Folding : Branch instruction concurrently executed with other instruction.

Results of these instruction will be put in correct order only.

If there is other instruction then only branch folding possible.

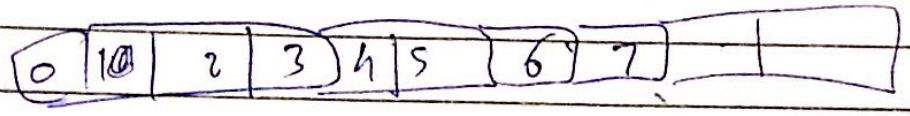
ADD R1, R2, R3
SUB
AND
OR

ADD → 0000
SUB → 0001
AND → 0010
OR → 0011

R1 R2 R3 R4 R5 R7 R8 R9

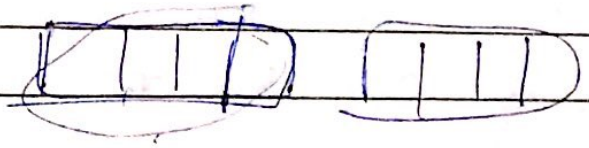
0000
0001
0010
0011
0

$2^4 = 16$.



6 7 10 11 14 15

ADD R1, R2, R3, 0



3/Nov/2017

Conditional Branch
- Delayed Branch
- Branch Prediction
  - Static
  - Dynamic

I_1	loop:	Shl R1
I_2		DCR R2
I_3		JNZ loop
I_4		ADD R3, R4

I_1	F	D	E	W		→ Target Instruction
I_2		F	D	E	W	
I_3			F	D	E	W
I_4				F	D	E → Branch Instruction
I_5					F	D
I_1						F

To solve this problem we can do this ⇩

I_1	loop:	DCR	R2
I_2		JNZ	loop
I_3		SLL	R1
I_4		ADD	R3, R4

~~for~~ Delayed Branch → Reordering instruction in such a way that one of the instruction is brought up to reduce delay.

Assume that Branch will never be taken and if branch is taken remove the un wanted prediction instruction. 50% time logic will be correct.

If there is condition checking at the end of the loop then all probability branch will be taken.

If condition checked at the starting of the loop then in all probability branch will not be taken.

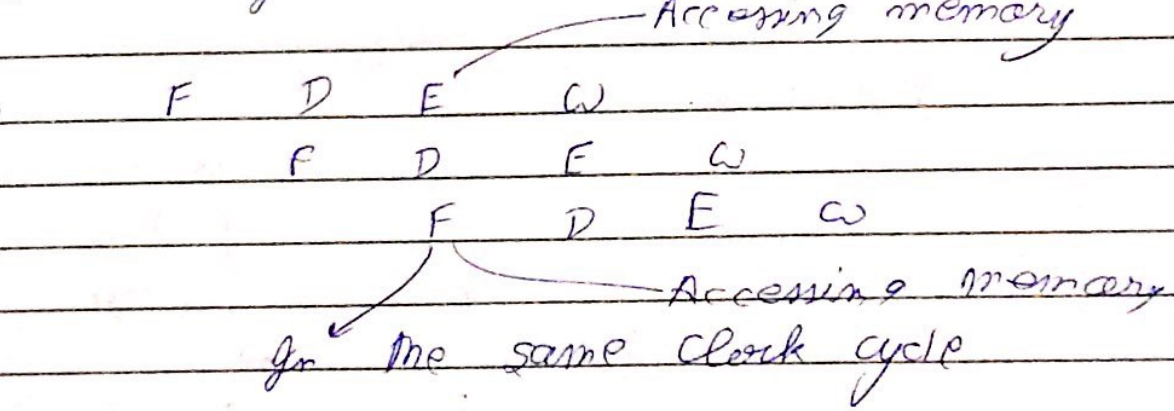
~~This~~ These are Static ~~condition~~ prediction

In dynamic ~~co~~ Prediction it check the previous history and predict not taken if not taken previously and vice versa

Speculative Execution
Given.

If wrong prediction then have to remove the instruction.

Structural Hazard



Split Cache is the way to overcome this problem of Structural Hazard.

How can addressing modes can cause hazards in pipeline or ~~dot~~ cause delay in the pipeline?

POP PUSH operation may change the stack pointer
Complex instruction always degrade the performance of pipeline.

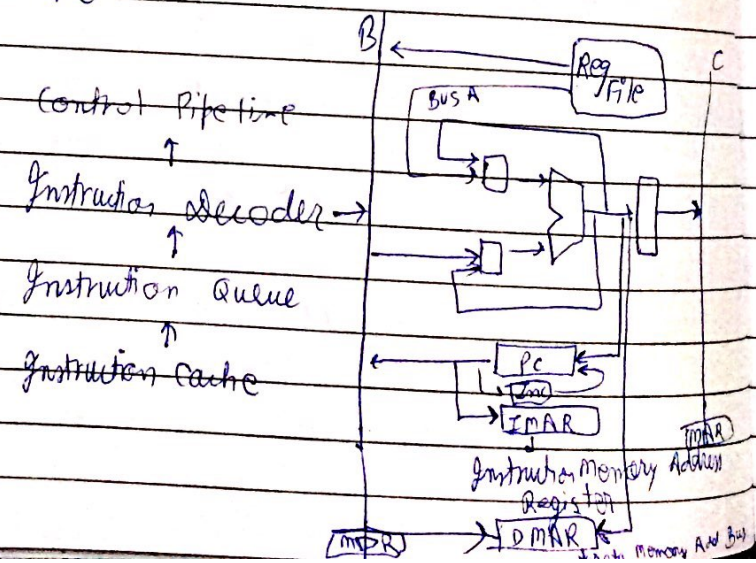
LOAD (R1), R2
LOAD 10(R1), R2 ← Needs ALU.
LOAD ((Offset + R1)), R2.

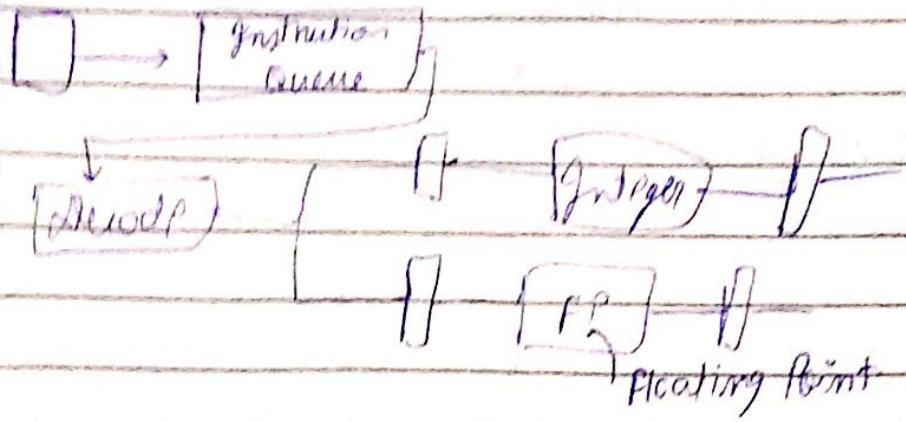
R1 = 3000
Offset = 100

1st memory access.

ADD offset, R1, R1 F D E M E M W
LOAD @R1, R3 F D
LOAD @R3, R2.

~~Instruction~~ Cache
~~Instruction~~ Cache.

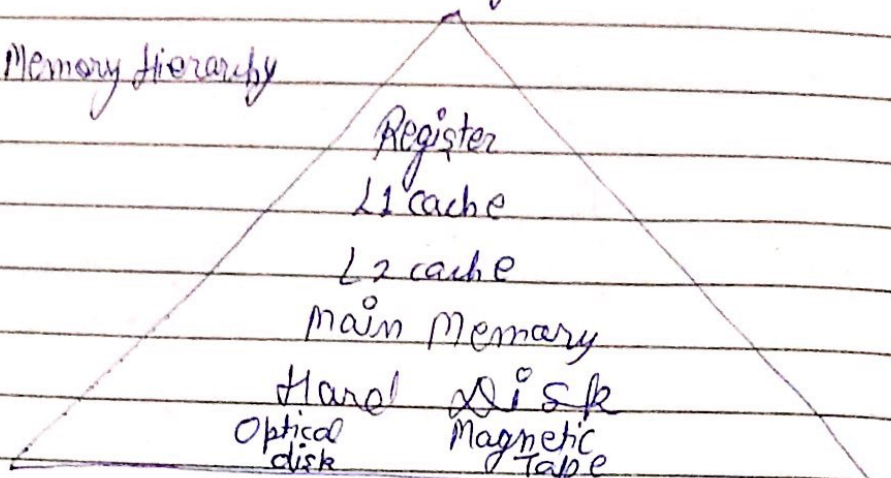


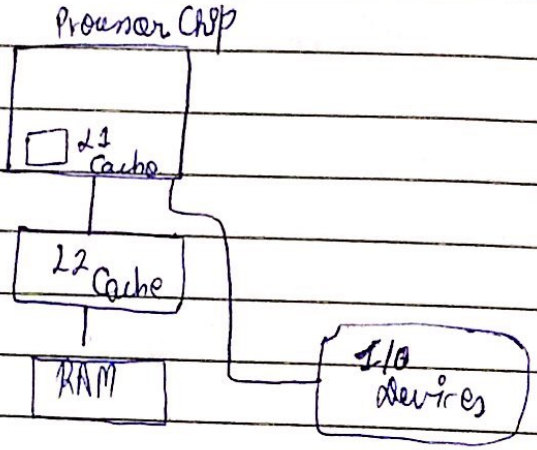

Instruction Queue
Decode
Integer
FP
Floating Point

6/Nov/2017

Memory

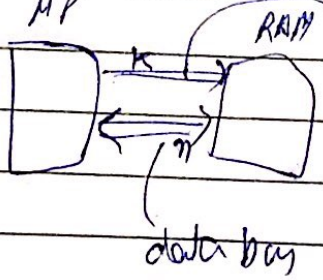
Memory Hierarchy





Cache are made up of 6 transistor

RAM read and write anywhere in same time.



K address lines
n data lines what are the maximum number of memory location that I can have?

K = 4 16 addresses.
32 bit machine it is word addressable memory. 16 words.
byte addressable 4 words.

Within a word lower two bit are going to give the byte address

32 bit address then 2^{32} → 1 GB memory

If cache miss then check main memory if not in main memory then from hard disk the program is brought.

Processor wants something checks cache there is a miss then main memory ~~then~~ if miss then through hard disk..

Load Through : Writing in cache and sending to processor at the same time.

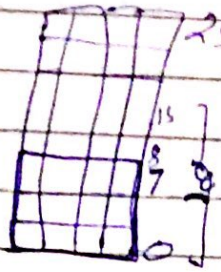
How not miss and hit checked? (later)
Cache works on the principle of locality. Locality of two type either special locality or temporal locality.
Time

If loop is running. Means after a period of time we need that instruction again known as temporal locality.

Main memory

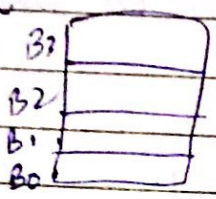
256 words

8 bit for addressing



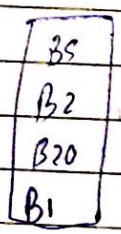
Cache

8
2
16



block size 8 words

Fully associative mapping: If we put the block (brought the block from memory) anywhere in cache.

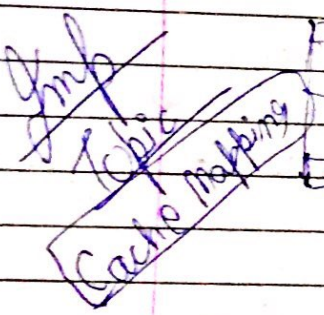


Have to follow some kind of Algorithm (FIFO may be)

7/Nov/2017

Cache

RAM



Lab Evaluation Week.

Fully Associative Algorithm

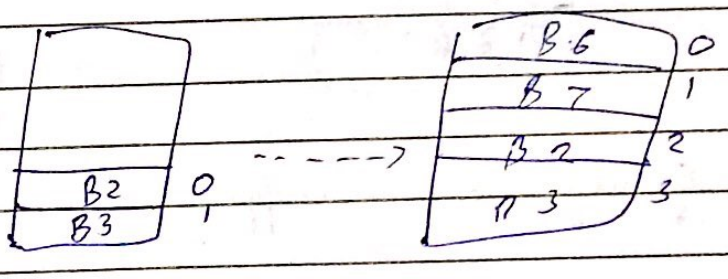
Fully Associative Memory: Cache filled in random manner and when cache is full then any algorithm is used to remove a block so that other block can be Rcpt.

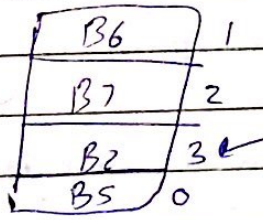
LRU: Least Recently used Algorithm is used mostly and ~~not~~ not LIFO

Less used block sent back to memory.

When processor checking block found then it is hit. We have counter for all blocks initialised with 0 and when there is a hit in the block then counter of other block are incremented by 1.

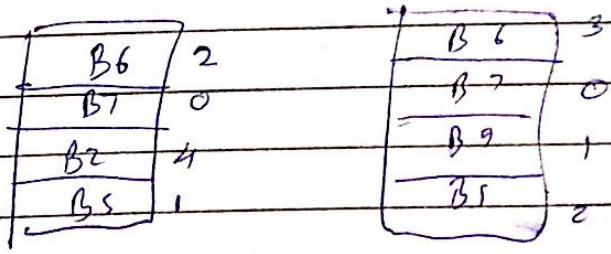
Counter also increases when new block are brought

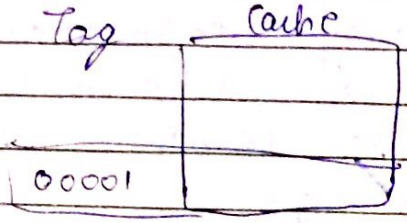




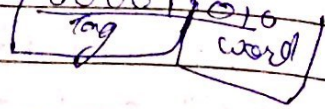
This will be removed if any other block comes in.

If B7 is used (a hit then)





Processor check the tag Bit.



Direct mapping

Direct mapping : There is a fixed place for a particular block.

Cache Block = (main memory Block) mod (No of block in cache)

4 Block in cache.

3	7	11	15	19
2	6	10	14	18
1	5	9	13	17
0,	4,	8	16,	12

M M M M ← fully Associative

Add $\frac{2}{8}, \frac{33}{8}, \frac{4}{8}, \frac{35}{8}$

0 4 0 4 ← Block in RAM

B0 B4 B0 B4

M M M M ← Direct mapping

If fully associative mapping

At start alway a miss

Address	Main Block	Cache block	Hit/miss (Associati)	Hit/Miss (Direct)
4	0	0	M	M
33	4	0	M	M
4	0	0	H	M
35	4	0	H	M

Set Associative mapping

8/Nov/2017

1→data

0→data

0 0 0 0 | 1 0 | 1 0

Tag | Block | Word.

Tag will be different

To check if 0, 4, 8, 12 or 16 tag is checked.

Set Associative mapping

Set Associative mapping : Something b/w direct and associative

2 way set Associative means two blocks will form a set.

Cache Block = (main memory block) mod (No of sets in cache)

Fully | Tag | Word |

Direct | Tag | Block | word |

Set | Tag | Set | word |

8/Nov/2017

Cache

1 → data belongs to correct program.

Valid Tag.

0 → data belong to previous program.

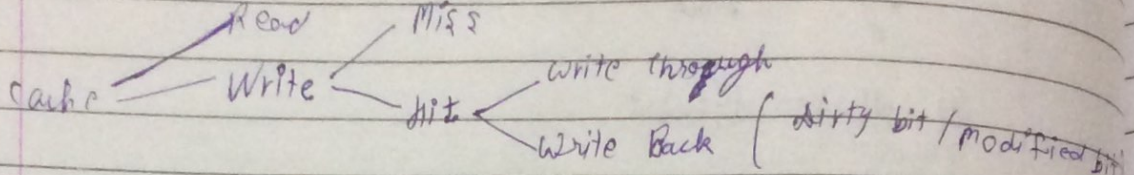
Size of the cache $256 \times 32 + 5 \times 2 + 1 \times 2$ ..

Cache — Read — Hit / Miss — Main Memory → Cache → µP

Cache — Write — Main Memory — Cache / µP } Load Through

miss penalty

If Miss then extra time to bring that data known as Miss Penalty



Cache Coherance : Whatever is there in the cache should be in the main memory (same as it is)

Write back Protocol:

Dirty block : when changes only in cache and not in main memory

Modified or dirty bit checked then at the time of replacement block will be written back otherwise block ~~is~~ removed as it is.

What will be the average acces time of the processor?

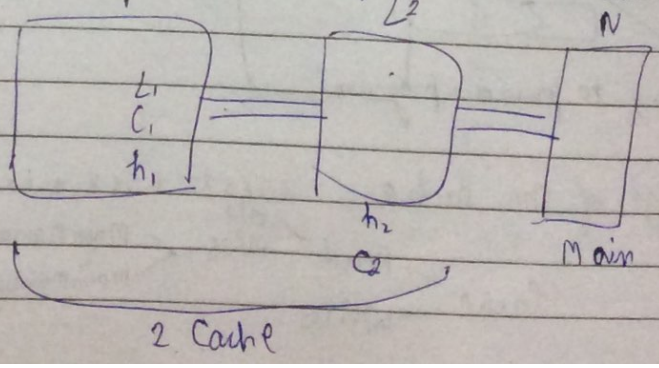
$h \longrightarrow$ hit rate

$1-h \longrightarrow$ miss rate

Miss Penalty

Avg. Access Time $= hC + (1-h)M$

C : time required to access the cache



$$Time = h_1 C_1 + (1-h_2)\left[h_2 C_2 + (1-h_2) M \right]$$

Q. 30% of the instruction in programme are memory read write operations.

Hit Rate

Instruction = 0.95
Data Cache = 0.9

Had there be no cache time required to access Main memory is 10 clock cycle.

Miss Penalty 17 clock cycle
Cache Access 1 clock cycle

Calculate Speed up — With Cache
Without Cache

Find Ratio

Solution:

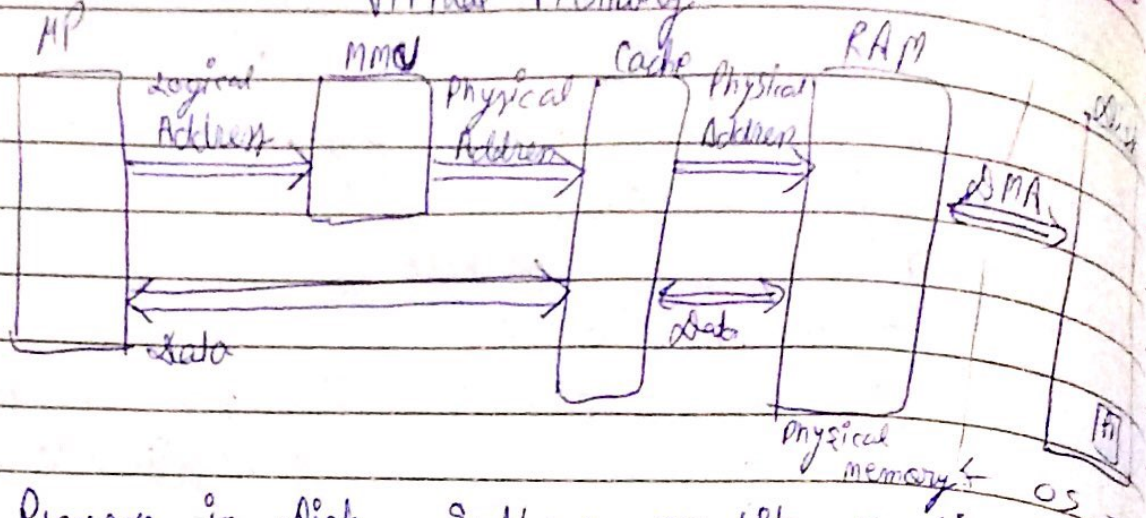
With cache $100\left[0.95 \times 1 + (1-0.95) \times 17\right] + 30\left[0.9 + 0.1 \times 17\right]$

Without Cache $(100+30) \times 10$

$$\text{Speed up} = \frac{\text{Time w/o cache}}{\text{Time with cache}}$$

10/Nov/2017

Virtual Memory



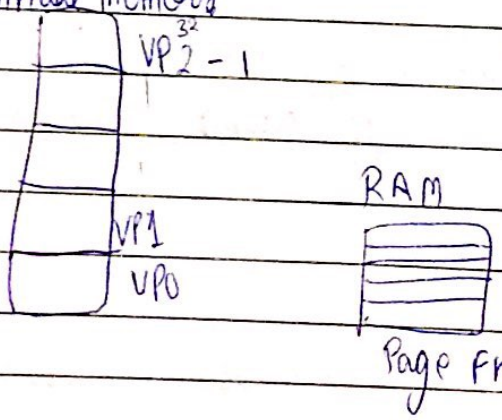
Program in Disk. Suppose 32 bit machine then $2^{32}-1$ address are possible

OS doesnot let the user know that it has less memory. It gives an impression that it has a large memory.

Direct Memory Address : DMA

OS will load a part of it to the physical address

MMU : Memory management Unit : Going to generate the actual physical address



Virtual Memory divided into virtual pages.

Page Frame and Virtual pages same size

Here also replacement same as Cache Main Memory here it is RAM and Virtual Memory.

Page fault : Virtual page 20 not available for Program 1. Page is not there in the RAM.

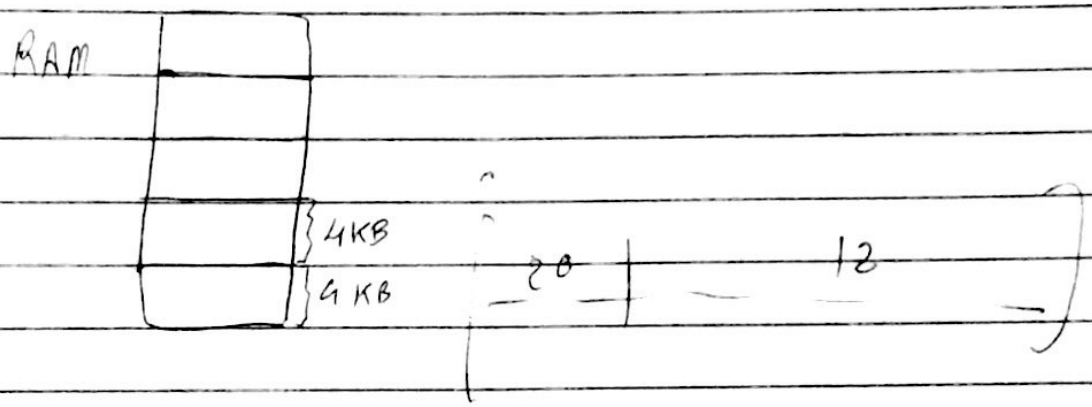
When there is a page fault replacement takes place

32 bit address → Logical Address

Physical memory depends upon the size of physical memory.

Ram - 8 4KB pages → 32 KB memory. ⇒ 2^{15} Bits

Now map 32 with 15.



3	12
Page frame	offset

Here 2^{32} → size of virtual memory.

No of virtual pages → $\frac{2^{32}}{2^{12} \times 2^{10}} = 2^{20}$

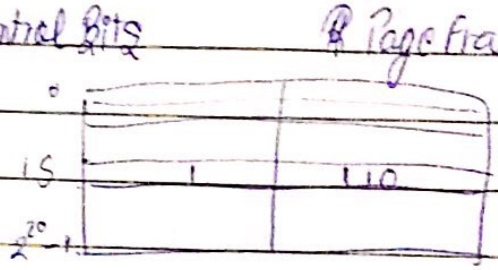
Bits for virtual Page = 30 bits.

offset 12 bits.

VP	offset
20	12

offset Bit are mapped exactly in the same manner

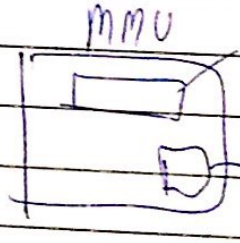
Page Frame Table



Valid but Dirty / modified bit
Read Write Permission.

In execution also No 'write' can take place

32 bit Logical Address ⟶ 15 bit physical address.



will keep a part of table in

Translation lookaside Buffer