INDEX

NAME: K. Goutham STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Computer Organization	03-	

Teacher's Sign / Remarks

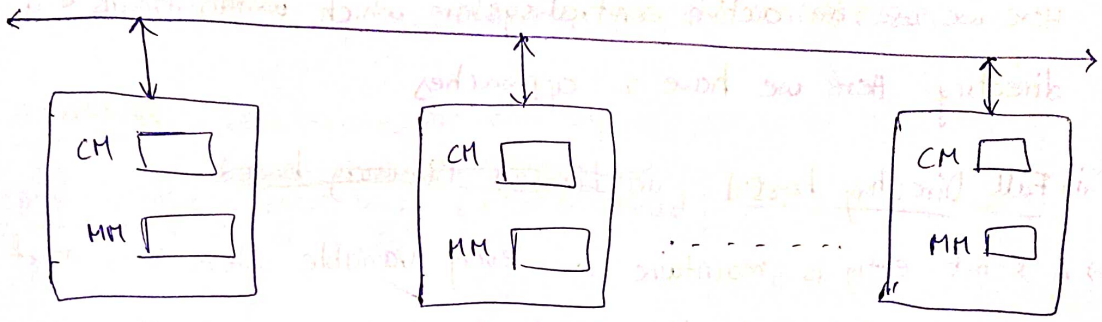
→ Here current write operation is always done in cache.

Multiprocessor Environment

* Every CPU has its own cache

* local variable : a variable within the CPU memory is local to that CPU.

global variable : a variable which is within some other CPU



* Here even using write-through also can't eliminate the cache coherance problem, cuz some other processor could be using the variable (or) data as global.

(i) Cache Snooping Problem :

(i) Update all :

~~All processors having an updated variable~~,

If a processor ~~has~~ updates a variable, the central system observes this updation and gets it reflected in all the caches.

→ Here the disadvantage is that after updation in other CPUs the CPU sometimes may not use the variable anymore and there is no use of this updation.

(ii) Selective Update :

Processors listen updation and update the shared variable only if required. It is gives better performance.

disadvantage:

→ The processor which ~~prefo~~ performs updation has to broadcast the information related to the updation the bus.

So every processor must monitor the bus activity.

(ii) Directory bayed Protocols:

Here we use an active central system which ~~to~~ maintains a directory. Here we have 2 approaches

i) Full Directory bayed ii) ~~Limited Directory based~~.

→ A k-bit entry is maintaine for every variable where k is no of processors. '1' denotes the variable is being used by the corresponding processor.

→ Also the value (recent one) is maintaind.

→ A V/I flag is also maintained.

if V/I = 1 then the value stored in directory is valid

if V/I = 0 " " " " " " invalid

→ If a processor wants to update the value, then ~~it~~ until the updation is finished V/I is set to 0 and later after updation V/I is set to 1.

→ This protocal uses array for directory implementation

limitations:

→ more space is required.

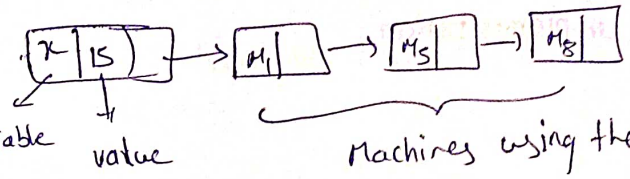
→ ~~Out of~~ for large value of k, most of the bits go waste if ~~we don't~~ only few processors use the variable.

This limitation is overcome by using ~~to~~ limited directory protocol.

(ii) Limited directory based Protocol

* Linked list is used for directory information.

→ if a variable is used by only p machines the the variable has p nodes.



Advantage: less space for directory

Disadvantage: more work (more overhead)

Block Replacement Techniques:

Optimal strategy:

→ It gives best performance. For replacement it identifies the MM block in CM which is not needed in future for longest time.

Adv: Best performance

Disadv: difficult to know the future references.

LRU:

→ Here most recently used block is protected from replacement. Use bits are used for the implementation. least recently used block is selected for replacement.

→ If there is no hit, then it is similar to FIFO

→ Queue can be extended for implementation, however on hit contents are to be reorganized

FIFO

→ The block which entered first into the cache is selected for replacement (longest time spending)

→ Here assumption is that most of the references for that block might have completed

→ Queue is suitable for implementation.

~~FIFO~~

Note:

→ FIFO & LRU can be used with associate & set associative mapping.

→ In directe* mapped cache we have only one option to place any MM block into cache. ∴ we don't use any of these.

Eg: A cache of 4-blocks, the following MM block references are made by the CPU

8, 13, 7, 8, 9, 16, 7, 13, 8, 13, 17, 7

optimal:

8 17	
13	
7	
8 16	

total ref = 12
∴ 6 misses
~~hit~~ ∴ 6 hits

∴ Hit ratio, h = 0.5

FIFO

8 16 7	
13 8	
7 13	
8 17	

~~hits~~
∴ misses = 9
⇒ hits = 3

i.e., 50% of the optimal strategy

∴ hit ratio = 3/12 = 0.25

LRU:

8 13	
13 16 17	
7	
8 8	

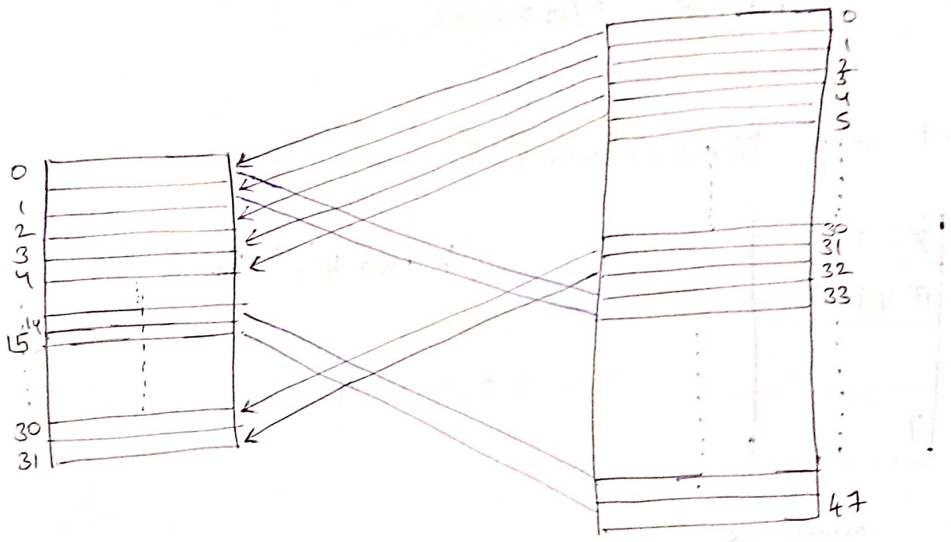
misses = 8
hits = 4
∴ h = 4/12 = 0.33

i.e., 66% of the optimal strategy

(Q31) A 2-D array occupies 48 blocks and cache is having 32 lines. Array is accessed 3-times. The last block of array is filled 10% only. How many cache lines are overwritten during the successive accesses and what is the minimum value denoting total no of miss references?

a) 16, 112 b) 8, 112 c) 16, 122 d) 8, 122

Sol:



The cache blocks 16 to 31 will be brought in first array access and will be left untouched during next access.

∴ In the next two accesses access to element of block 16 to 31 will be hits i.e., $16 \times 2 = 32$

total no of references to blocks = $48 * 3 - 32 = 112$

Also in every access the cache blocks 0 to 15 will be overwritten.

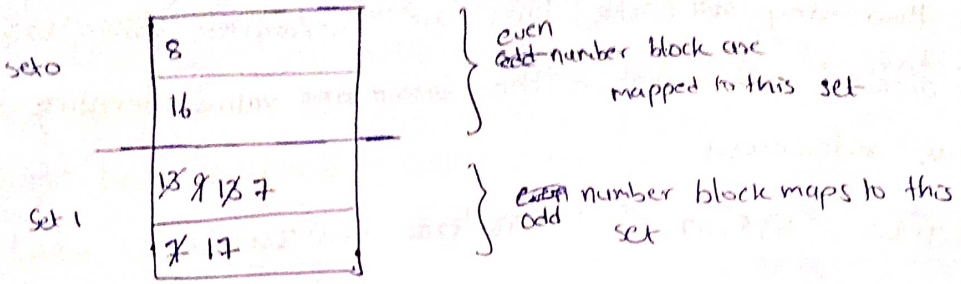
∴ 16

∴ Ans: (a)

Eg: Block replacement with 2-way set association

8, 13, 7, 8, 9, 16, 7, 13, 8, 13, 17, 7

~~optimal~~: LRU:

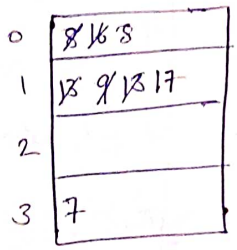


∴ no of ~~hits~~ misses = 8

⇒ hits = 4

⇒ hit ratio = 4/12 = 0.33

Eg: Block replacement with Direct mapped cache



$(k \bmod 4)^{th}$ block for k^{th} MM block

8, 13, 7, 8, 9, 16, 7, 13, 8, 13, 17, 7

- misses = 8

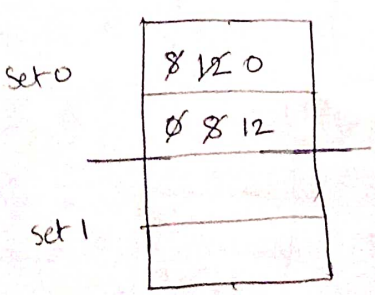
⇒ hits = 4

∴ hit ratio = 4/12 = 0.33

(Q32) A 4-block cache is organized in 2-way set associate mapping. It is empty initially. LRU is used for block replacement. For the following MM ref ~~used~~ what will minimum no of faults

~~Soln~~:

8, 0, 12, 8, 0, 12



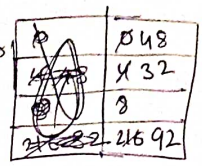
∴ no of faults = 6

Q33) 4-way set associative, 16-block cache.

MM ref: 0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155

which MM block will not be present in cache with LRU

Sol: a) 8 b) 129 c) 216 d) 3



remainder	Ref
0	0, 4, 8, 216, 8, 48, 32, 92
1	1, 133, 129, 73. (all will be present)
2	—
3	255, 3, 159, 63, 155 ⟶ 255 will not be present

∴ opt (c) no of misses = 16

Method 2:

For the reference string we can check from opposite direction & find it.

Main memory:

(i) Memory Expansion

(ii) DRAM

(iii) Memory interleaving

Memory Expansion

→ Construction of bigger capacity memory using smaller size chip

Eg: 8K x 16 memory using 1K x 4 chips

Target memory: 8K words
16 bits/words

Basic chip: 1K words
4 bits/words

no of chips req = $\frac{\text{target capacity}}{\text{Basic chip capacity}}$

$= \frac{8k \times 16}{1k \times 4}$

$= 8 \times 4 = 32$ chips

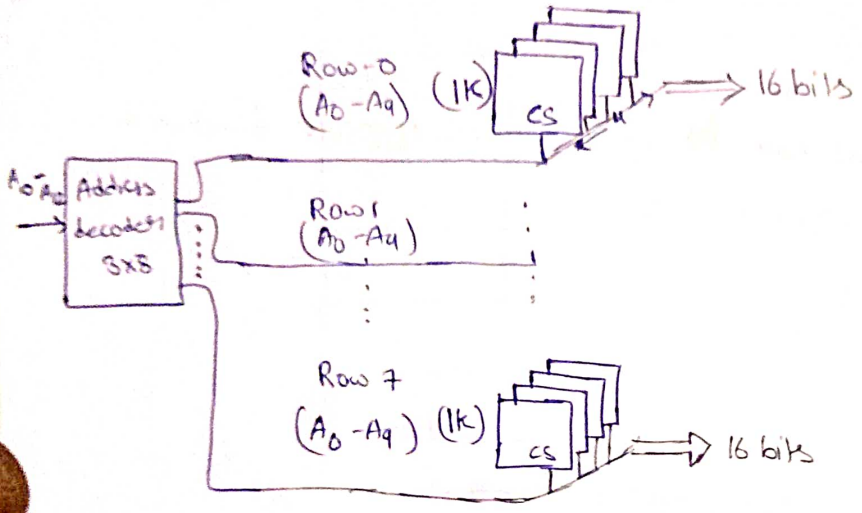
1k x 4 means ~~4k rows~~ 1k rows & 4 colums

4

these are arranged as

8 (1k x 4) rows with 4 (1k x 4) Column

i.e., 8 rows & 4 colum



chip select lines are connected in parallel. i.e., to all columns

Here each row is called a module.

Thus we have 8 modules in which each module gives 1k x 16

(Decoder) $A_{12}A_{11}A_{10}$	(within module) $A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$		Addres
0 0 0	0 0 0 0 0 0 0 0 0 0 to 1 1 1 1 1 1 1 1 1 1	Row-0 (Module 0)	00FF to 03FF
0 0 1	0 0 0 0 0 0 0 0 0 0 to 1 1 1 1 1 1 1 1 1 1	Row-1 (Module 1)	0400 to 07FF
⋮	⋮	⋮	
1 1 1	- - - - - - - - - -	Row 7 (Module 7)	1C00 01FF

Adv

total address range 0000 to 1FFF

Eg: Target size : 64 KB

available : 4K×4 chips

Find no of chips required & size of decoder.

Sol:

$$\text{no of chips} = \frac{64K \times 8}{4K \times 4} = 16 \times 2$$

∴ 32 chips are required

arrangement : 16×2

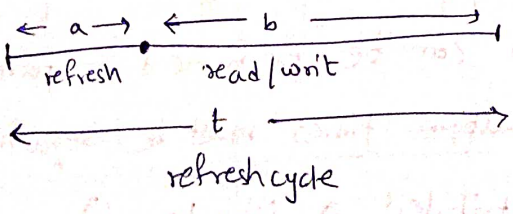
i.e., 16 rows & 2 colums

size of decoder depends on no of rows

i.e., 4×16 decoder in this case.

Dynamic RAM (DRAM)

* DRAM is used in primary memory construction
* volatile
* contains one transistor & one capacitor for each bit
* refreshing is required.
* In one refresh entire row gets refreshed
* In one refresh cycle all rows are refreshed
* To perform memory read or memory write the memory must be stable. i.e., refreshing must be done.



Here 'a' is overhead time

$t = a + b$

Refresh overhead % = $\frac{a}{t} * 100$

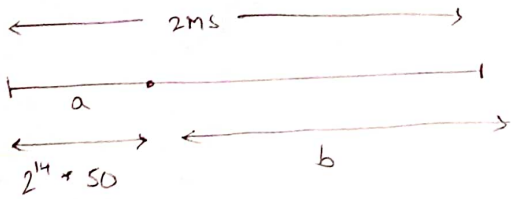
% time available for memory read/write = $\frac{b}{t} * 100$

→ Here we use counter for regular refresh cycles.

→ DRAM has more package density than SRAM, however it is comparitively slower.

(Q34) A MM 1GB is constructed by 256M × 4 bit DRAM chips. The no of rows in DRAM chip is 2^{14}. It takes 50ns for refreshing one row. The refresh period is 2ms. percentage time that is available for memory read/write is.

Sol:



$$\text{req \%} = \frac{b}{2ms} \times 100$$

$$= \frac{2ms - a}{2ms} \times 100 = \frac{2ms - 2^{14} \times 50ns}{2ms} \times 100$$

$$= 59\%$$

Memory Interleaving:

→ In order to improve the performance, MM uses modules rather than single large unit.

→ Module internal operation can be performed in parallel. However accessing same module multiple times must be sequential. Information is always distributed a (inter leaved) across the modules.

→ 2 ways of interleaving:

(i) lower order interleaving

(ii) higher order interleaving

Lower order interleaving:

uses higher address bits for word selection and lower address bits for module selection. It gives faster response. However it suffers from spatial locality.

Higher Order interleaving:

user lower address bits for word selection and higher address bits for module selection.

→ provides better spatial locality (consecutive words in same module)

~~2 MEMORY~~

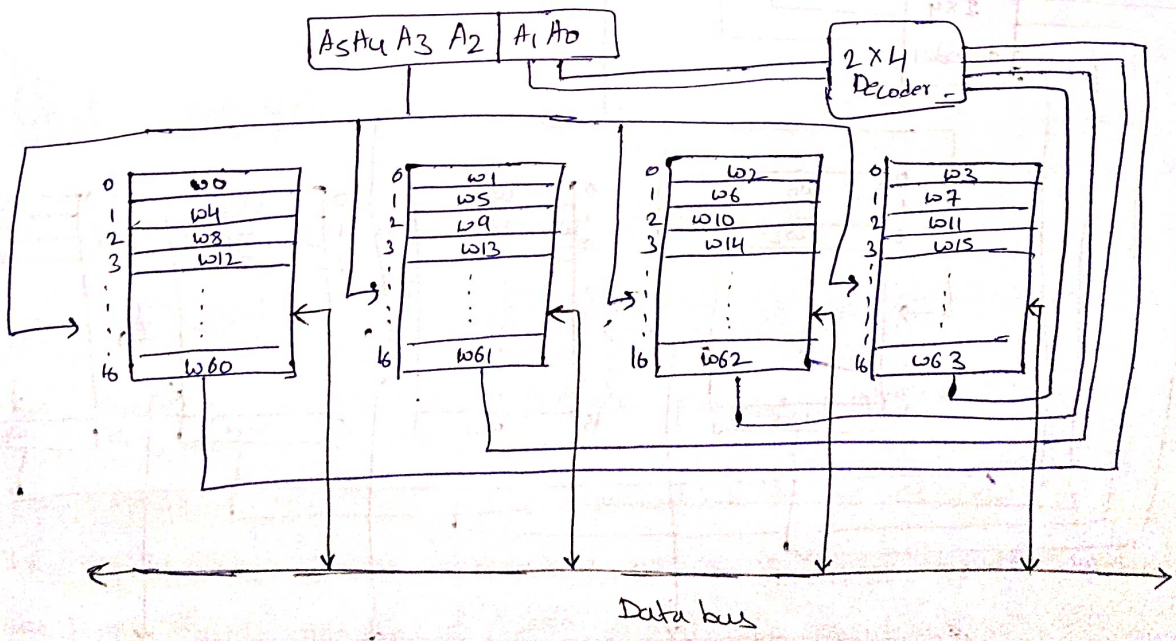
→ Memory bank uses combination of both.

→ If we use k modules we call it k-way interleaving

Eg: Consider 64 word memory

6 bit address is needed ($A_5 A_4 A_3 A_2 A_1 A_0$)

4-way interleaving (lower order)

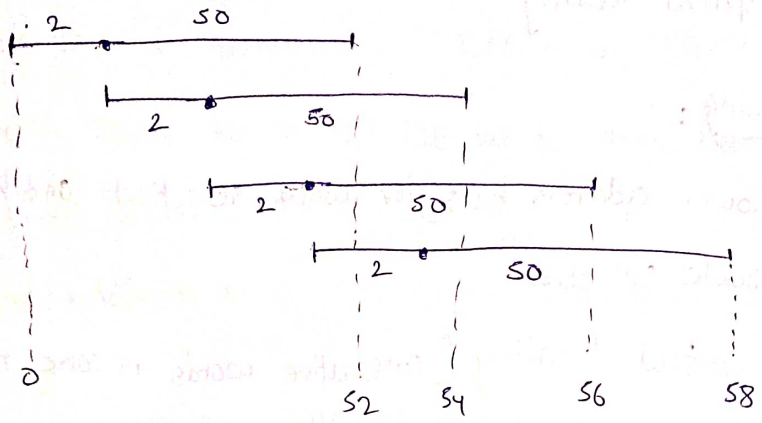


→ Here words are stored in wrap around fashion.

Assume accessing memory takes 50 ns
and decoder delay is 2 ns

→ If memory is a single large unit,
to access 4 consecutive words it would take
$4 \times 50 = 200$ ns

→ But in lower order 4-way Interleaving

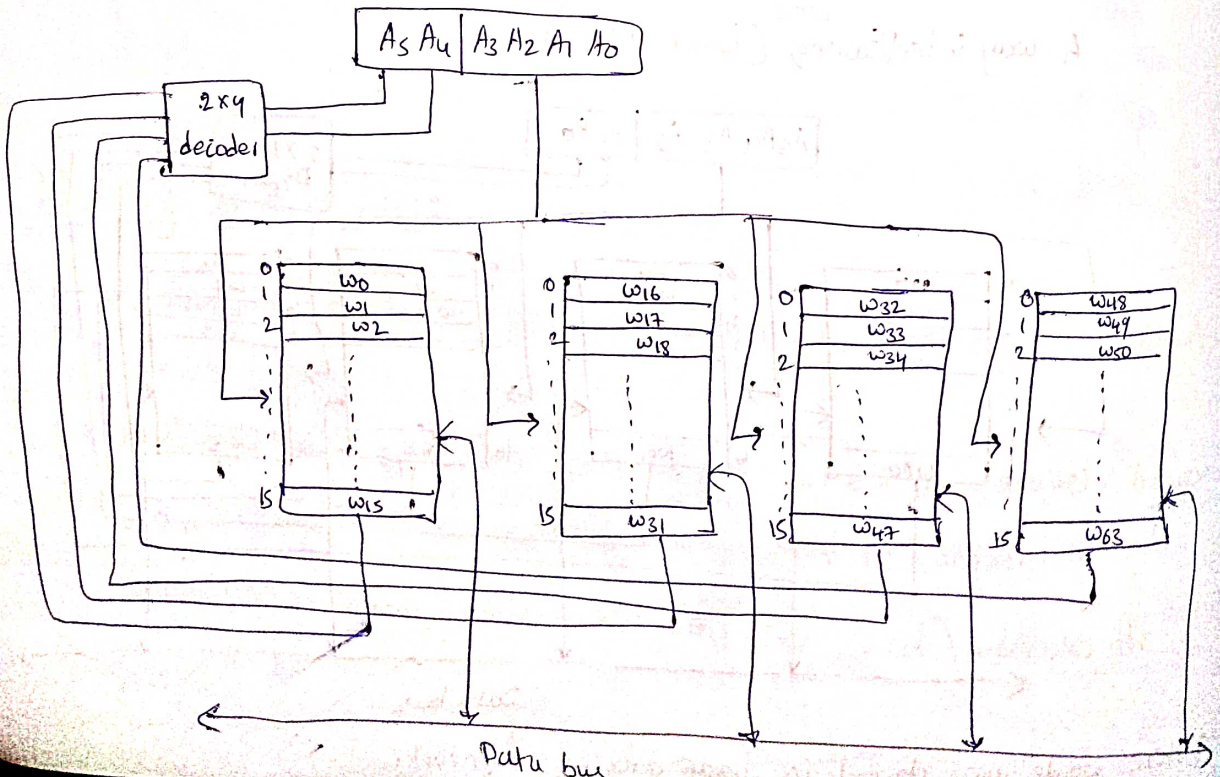


And this organization takes only 58 ns

→ Internal operations of memory are overlapped (but not addressing).

31/08/20

Higher Order Interleaving:



(Q35) In a system design MM employs 8 modules with 8-way interleaved design. The words are stored in the modules in wrap around fashion. Internal operations of modules can be done in parallel, however accessing the modules multiple times has to serialized. Let 50ns required for module internal operation and 100ns for bus activation. Max no of memory operations allowed to be initiated in 1ms for this main memory.

a) 1000 b) 10000 c) 100000 d) 1000000

Sol:

for every initiation require 100ns

$$\therefore \frac{1ms}{100ns} = \frac{10^{-3}}{100 \times 10^{-9}} = \frac{10^6}{100} = 10^4 = 10000$$

(Q36) A DRAM chip is having 2^{10} rows & each row contains 2^{10} cells. Time required for one refresh operation is 100ns. How much time is req for refreshing the DRAM.

Sol:

one refresh means for entire row

$$\therefore 2^{10} * 100\,ns$$

Control Unit Design:

* CU is responsible for implementing instructions
* Each processor will have its own instruction set architecture (ISA)
* CU has processor register for specific purpose / general purpose.

PC: maintain address of nxt instruction to be executed

IR: holds the current instruction which is to be implemented.

Stack Pointer register(SPR): Maintains address of top of the stack.

general purpose registers

* These are also known as scratch pad registers. They are used for storing information temporarily.

Accumulator: maintains one of the operand and also it is the destination where result is stored.

* It is central register to interact with I/O devices
* It supports implied addressing mode.

ISA (Instruction Set Architecture):

→ ISA influences the design of processor

→ It indicates behaviour of instruction

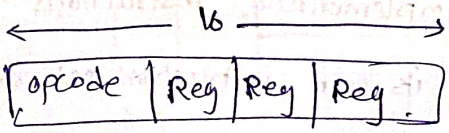
Eg: ∅

→ ~~Sample Instruction format~~

→ In general processor do not allow direct data transfer b/w two memory location. generally it is done via registers.

Eg: A processor uses 16 bit instruction and each instruction will have opcode, destination registor, source operand one register, and source operand register. If ISA is having 100 instructions what will be the no of general purpose registors registors

Sol:



100 instruction ⟶ 7 bit opcodes

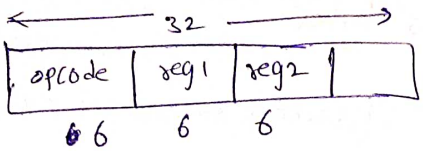
⇒ 9 bits ⟶ 3 reg

⇒ 1 reg ⟶ 3 bits

∴ $2^3 = 8$ general purpose registors.

Eg: A Machine has 32-bit architecture with 1 word long instruction. It has 64 GPRS (general purpose registers). It needs to support 45 instructions. Each instruction will have opcode, 2 general purpose registers. what is the maximum decimal value of unsigned operand supported by immediate ~~field~~ field

Sol:

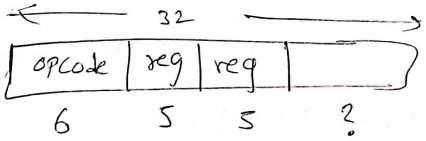


∴ bits for operand = 32 - 18 = 14

∴ Max value = $2^{14} - 1$ = 16383

(Q37) A processor is having 40 distinct instructions and 24 GPRs. A 32-bit instructions has an opcode, 2 register operands and one immediate operand. The no of bits available for immediate operand is _____

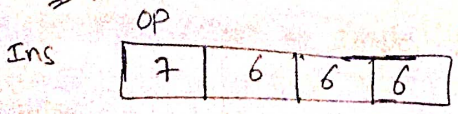
Sol:



∴ 32 - 16 = 16 bits

(Q38) A processor's ISA has 80 instructions and has 64 GPRs. Each instruction contains opcode with 3 registers (1-Dest & 2 source operand). A program is having 100 instructions how many ~~of code memory is needed~~. bytes of code memory is req?

Sol:



i.e., 25 bits

size of code memory = 100 * 25 bits

= 2500 bits

= 2500 / 8 = 312.5 bytes

PC points to byte but it cannot point to a bit

∴ we consider size of instructions as 4 bytes

∴ 100 * 4 = 400 bytes

Addressing Modes:

→ Addressing Mode denotes how operand is refered in the instruction

→ Addressing modes give flexibility in program development.

→ The address of the operand is termed as effect address (EA)

→ In broad addressing modes are

1) Non-computable addressing mode (calculation is not needed)

2) Computable addressing mode (calculation is needed to get the address.)

Non-Computable addressing modes:

i) Immediate:

→ operand itself is present in the instruction.

→ There is no EA.

→ Fastest addressing mode and suitable for refering program constants

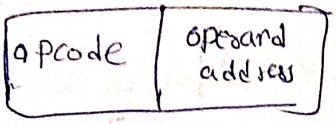
format:

opcode	operand

Limitation: operand range is small.

(ii) Direct Addressing Mode:

Here operand address is given instead of operand



adv: overcomes limited operand range

but it requires one extra memory read.

EA: operand address

Operand: M[operand address]

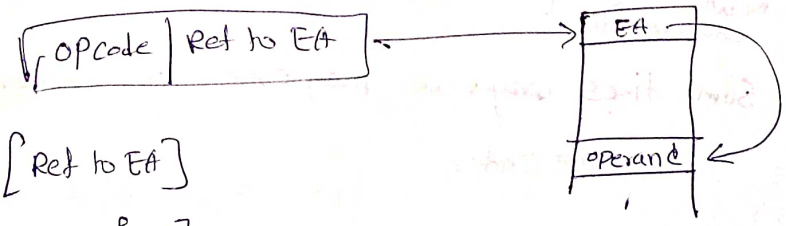
→ It is suitable for accessing static variables

limitation:

Limited address range.

(iii) Indirect Addressing Mode:

Here address of the effective address is given



EA: M[Ref to EA]

operand: M[EA]

= M[M[Ref to EA]]

→ two memory ref are needed

∴ slowest among non-computable addressing mode

→ It overcomes the problem of limited operand range & limited address range

→ It is used to support pointers, parameters passing.

→ This is also known as single indirection.

double indirection:

opcode	Ref of Ref of EA

EA: M[M[Ref of Ref of EA]]

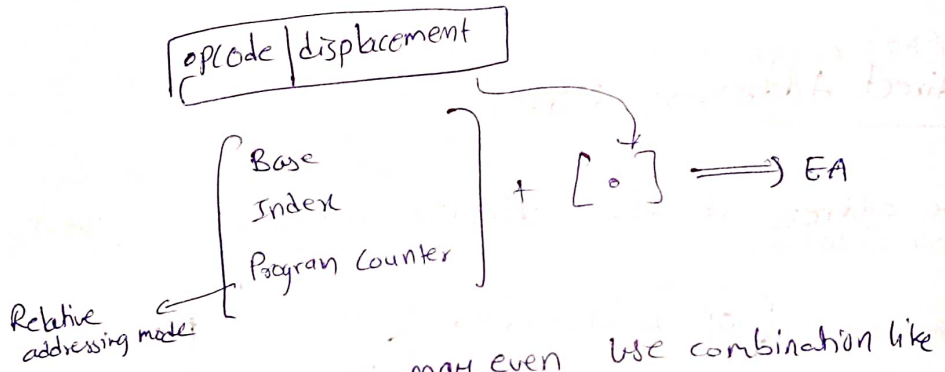
Operand: M[M[M[Ref of Ref of EA]]]

Here we need 3 memory references.

Silly for n-indirection, we need "n+1" memory references.

Computable Addressing Modes:

→ Here EA is computed using portion of instruction (displacement) using one or more processor registers



Some times we may even use combination like

Base Index

Applications:

Base Addressing Mode:

* used for relocatable codes

Index addressing mode:

* used for accessing array elements

Relative

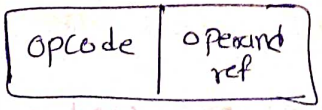
Program Counter Relative Addressing Mode:

* used for Inter segment (long jump) branching & Intra segment (short jump) branching

Instruction Design

→ A instruction is having 2 basic components.

i.e., opcode & operand reference

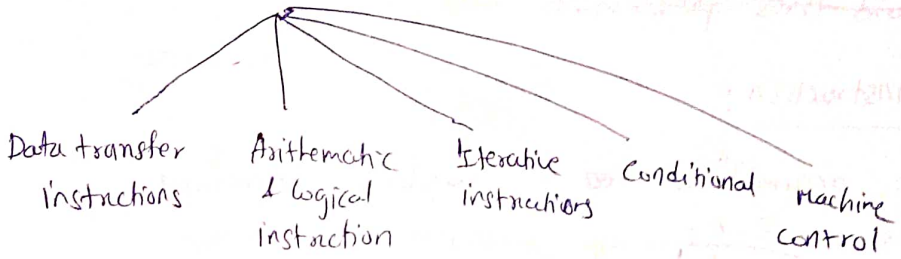


we can classify the instruction based on. ~~opcode~~

i) opecode

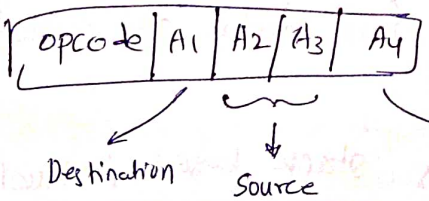
ii) operand references.

Opcode based classification (functional classification)



Classification based on operand references

4-address instruction:



Adv:

→ Here instructions need not to be store contiguously ~~beca~~ because A4 creates a linked list ~~of~~ ~~inst~~ instruction.

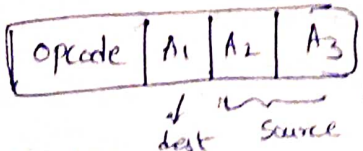
→ design is simple.

disadv:

→ instruction length is very high

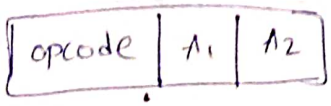
3-address instruction:

* PC is added & A_4 is removed.



2-address instruction

* one of the source operand will be used as destination too.



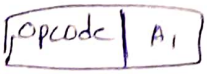
Disadv:

* overwriting (on memory)

~~To avoid this problem of~~ over

1-address instruction:

* Here accumulator ~~reg~~ is implied operand.



Here the problem of memory overwriting is overcome

Zero Address Instruction:

Implicit stack is used for data and operations are performed on stack contents.

Hence it is also known as stack based instruction.

Here we add stack pointer register (SP)

Disadv:

Now here we need PC, accumulator & SP

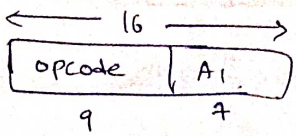
∴ Complexity is more.

11/09/20

Eg: Let us consider a machine that supports 1-address and 2-address instructions. Instruction length is 16-bit. Program is stored in 128 word memory. If there exist 2- two address instruction, what will be the no of 1-address instructions in the ISA

128 word memory ⇒ 7 bits for address

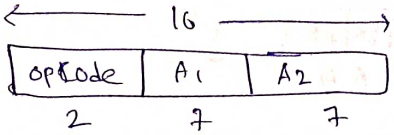
1-Address instruction



If we have only 1 address instructions then we can have

$2^9 = 512$ instructions

2-address instruction



∴ for each two address instruction 2^7 1-address instructions are to be sacrificed

Now we have 2 such two address instruction

∴ we need to accomodate $2*2^7$ 1-address instructions

∴ no of 1-address instructions

$= 512 - 256 = 256$

Q39) A processor is having 64 floating point registers and 16 integer registers. The ISA supports 4 types of instructions each having 16 bit length

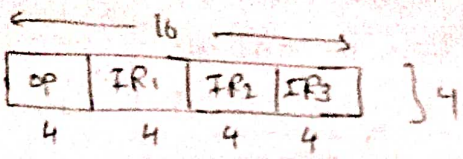
Type	Format	no of instructions
1	OP, IR1, IR2, IR3	4
2	OP, FR1, FR2	8
3	OP, IR, FR	14
4	OP, FR	N

find N

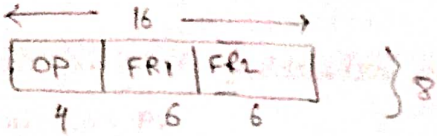
Sol:

IR — 4 bits FR — 6 bits

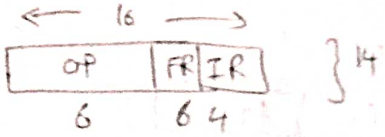
Type 1 :



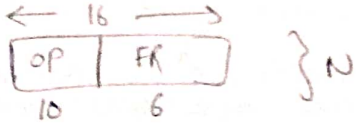
Type 2:



Type 3 :



Type 4 :



if only type 4 are present, we will have 2^{10}

adding type 3 takes 2^4 type 4 instruction

" type 2 takes 2^6 type 4 instruction

" type 1 " 2^6 " " "

$\therefore N = 2^{10} - 14 * 2^4 - 8 * 2^6 - 4 * 2^6$

$= 2^{10} - 224 - 512 - 256$

$= 32$

Instruction Pipeline :

→ Instruction pipeline overlaps instruction phases and thus reduces avg implementation time.

→ The phases of an instruction are:

i) fetch : getting the instruction from memory to IR

ii) decode : Instruction is decode (obtain operation from opcode) and req operands are fetched from memory to GPR

iii) Execute : perform operation in ALU

iv) Store : Store the result at destination.

→ Each phase is done by seperate hardware. So when one phase is being done, other phases' hardware is idle.

→ To remove this idleness & improve performance we use instruction pipelining.

→ Instructions must be independent for giving optimal performance.

(Q40) Match the following

P: Indirect addressing
Q: Indexed addressing
R: Base Registers Addressing

1. Array Implementation
2. writing reloadable code
3. Passing array as parameter
4. accessing constants

Ans:

P-3 Q-1 R-2

(Q41) In a hypothetical processor an instruction format is

opcode	X(R1)

where XX is displacement and R1 is base register. The displacement is denoted as 2'complement signed number. If X = 1111 1111 1000 and R1 is decimal value is 300. The effective address is in decimal is

Sol:

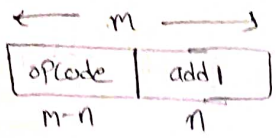
1111 1111 1000 ≅ 1000 = $(-8)_{10}$

∴ 300 - 8 = 292

Q42) A processor is having m-bit length instructions and support one address and two address instructions. The memory is having 2^n words. If there are 8-two address instructions what will be no of one address instructions.

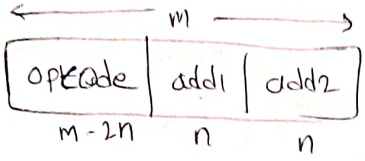
Sol.:

1-address



$\therefore 2^{m-n}$ are possible

2-address



$\therefore$ for each 2-address instruction 2^n one address instruction are to be sacrificed.

$\therefore$ no of 1-address instructions

$$= 2^{m-n} - 8 * 2^n$$

$$= 2^{m-n} - 2^{n+3}$$

→ performance of instruction pipeline is given by speed up factor.

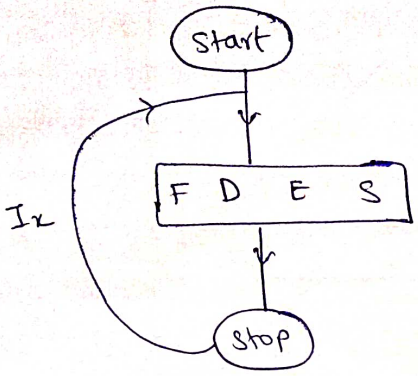
$$S = \frac{\text{time without pipeline}}{\text{time with pipeline}}$$

$\therefore S > 1$

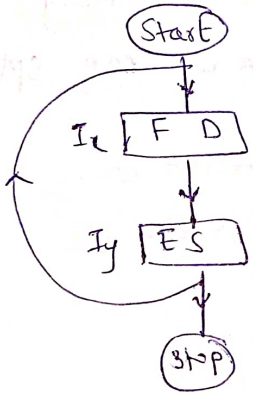
Evolution of instruction pipeline:

1. Non-pipeline system / Single stage Instruction pipeline:

Fetch - Decode - Execute - Store



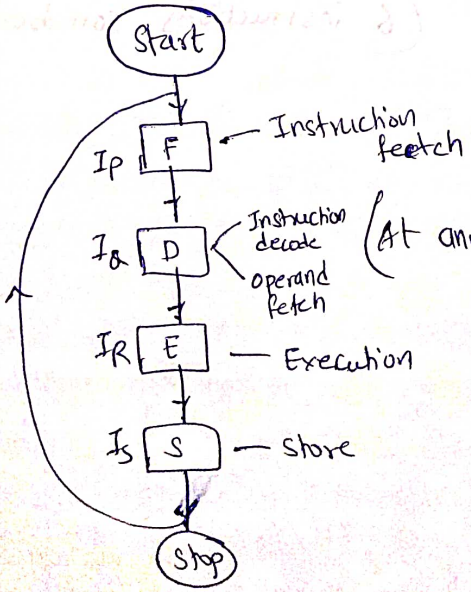
2-Stage Instruction pipeline



At any point of time 2 instructions can be active

Ideal performance = 2 times of the non-pipelined system

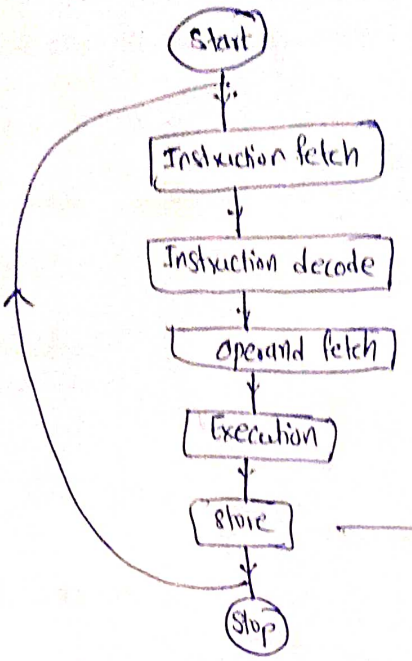
4-stage Instruction pipeline:



(At any time 4 instructions are active.)

ideal speed = 4

5-Stage instruction pipe pipline :

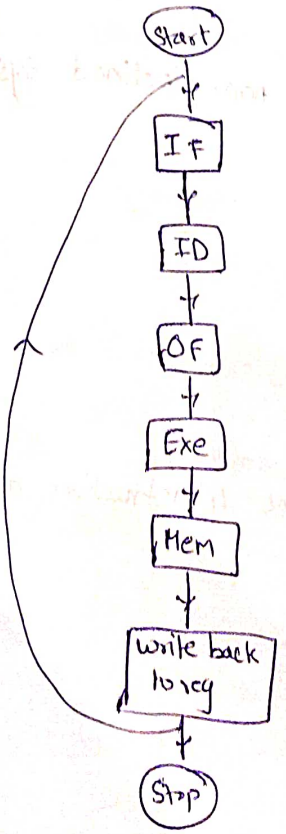


→ Here store may be done in registers or in memory

So we can split the stage

ideal speed up = 5

6-stage instruction pipeline

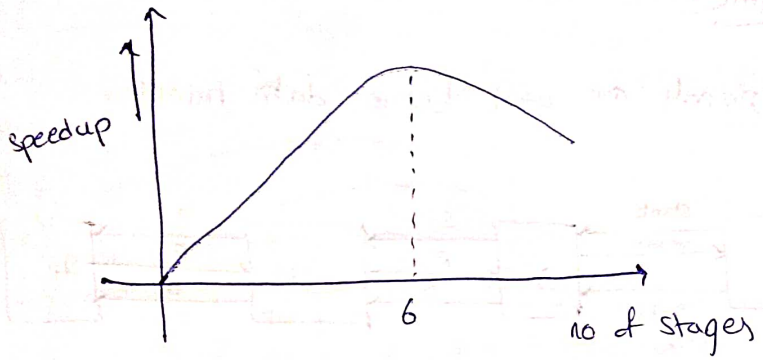


ideal speedup = 6

(6 instructions can be active if needed)

→ There is limit on how many stages we can have. we can't have arbitrarily large no of stages. Cuz overhead increases and performance drops.

So 6 is the max no of stages possible with optimal performance.



Types of instruction pipelines

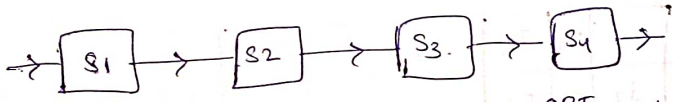
i) linear : Instructions move only forward (feed forward)

ii) Non-linear : Forward as well as backward (feed back) (i.e., back to previous stage)

iii) Synchronous : All stages move data at the same time.

iv) Asynchronous : Data movement blw the stages is not dependent on other stage's status.

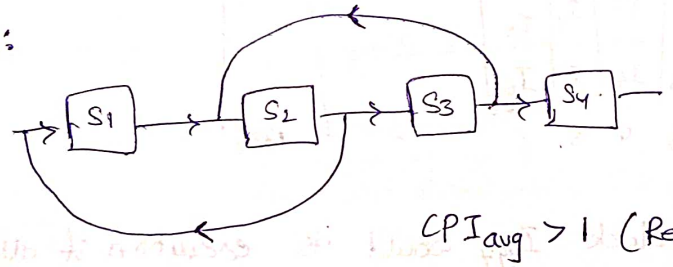
linear :



$CPI_{avg} = 1$

Instruction will not revisit any stage

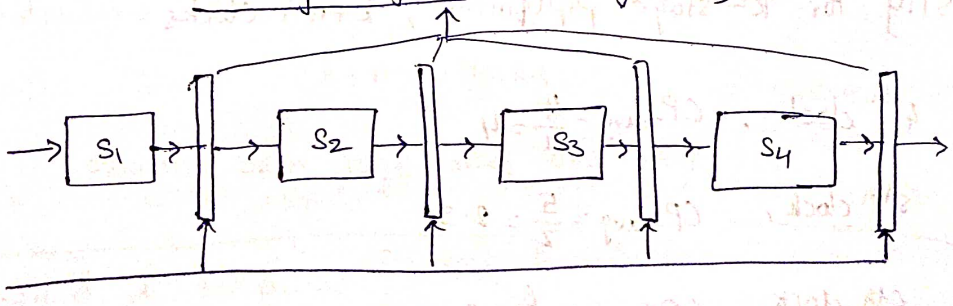
non-linear :



$CPI_{avg} > 1$ (Reservation tables)

Synchronous :

Here we have interstage registers (buffer registers)

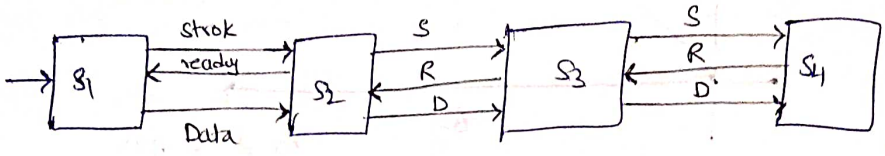


→ These registers are implement with flipflops operation with same clock.

Disadv:
→ Some stages may req more time. So other stages has to wait. Thus slowest stage decides performance.

Asynchronous Pipeline:

→ Handshake signals are used before data transfer

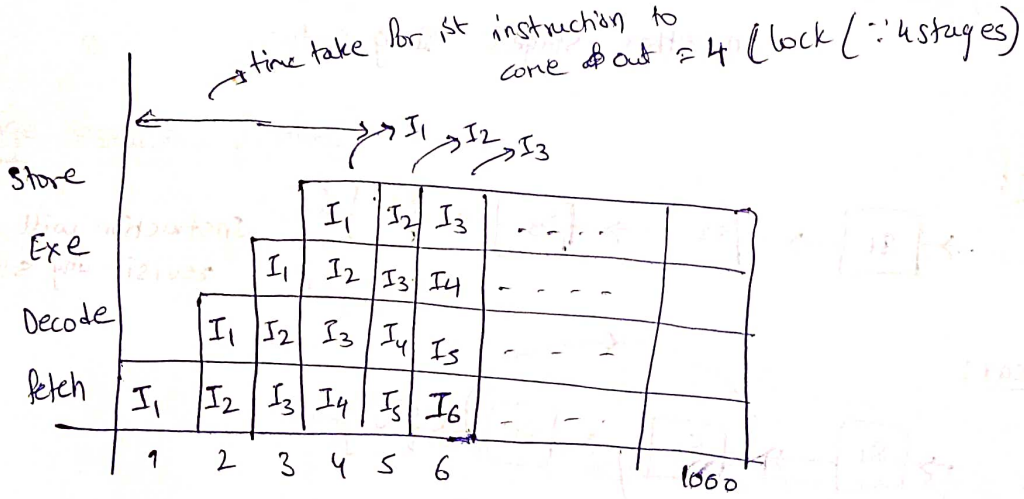


→ Here data transfer blw S_1 & S_2 is not dependent on data transfer blw S_3 & S_4.

→ Here we don't need any buffer registers.

Now assume each stage takes 1 clock

4-stage



Silly in 1000th clock I_{997} would its execution if all the instructions are independent.

∴ n instruction would take 4+n-1 clocks.

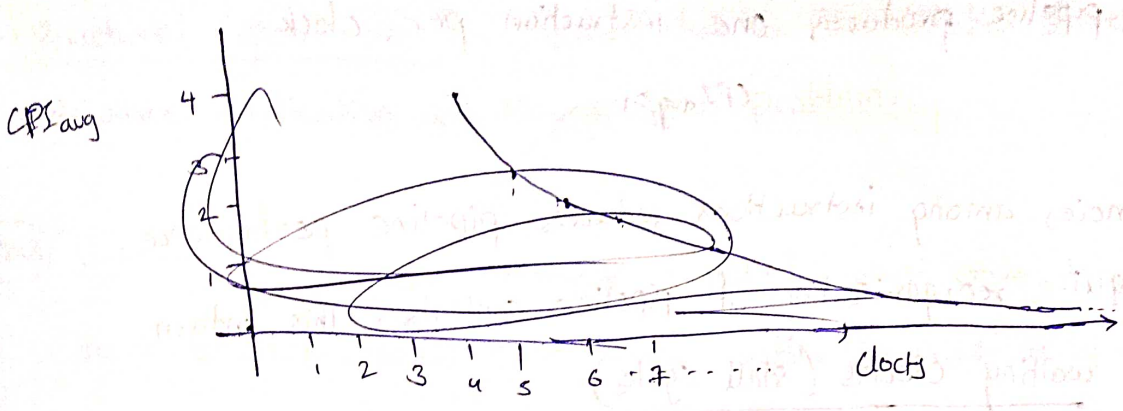
Silly for k-stage pipelining, k+n-1 clocks.

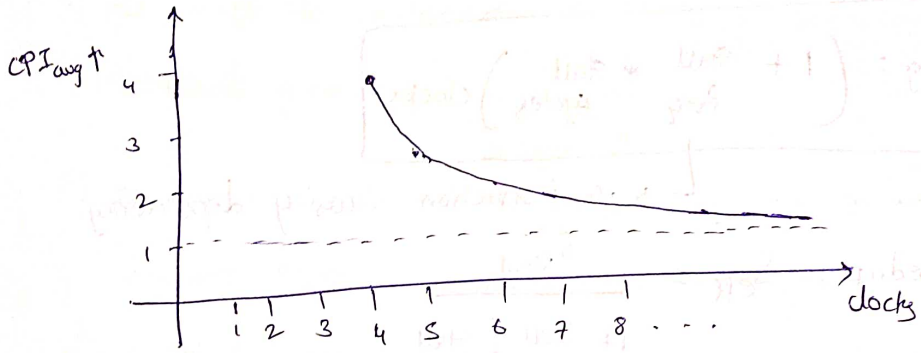
→ At 4th clock, $CPI_{avg} = \frac{4}{1} = 4$

At 5th clock, $CPI_{avg} = \frac{5}{2} = 2.5$

At 6th clock, $CPI_{avg} = \frac{6}{3} = 2$

At 1000th clock $CPI_{avg} = \frac{1000}{997} \approx 1$





∴ for large no of instruction $CPI_{avg} \simeq 1$

Performance of pipeline:

→ performance of pipeline is given speed up factor.

$$S = \frac{\text{time without pipeline}}{\text{time with pipeline}}$$

ideal value of $S = k$

$k \rightarrow$ no of stages

→ For n-instructions, k-stage instruction pipeline takes $(k+n-1)$ clocks

assuming each stage takes one clock.

Clock period of instruction pipeline:

= ~~Max (buff~~ = Max(stage delay) + Buffer overhead

→ On avg pipeline produces one instruction per clock

$CPI_{avg} = 1$

→ Dependencies among instructions reduces pipeline performance.

→ They require reorganization of pipeline contents. This inturn require waiting clocks (stall cycles)

$$\therefore CPI_{avg} = \left(1 + \underset{\text{freq}}{\text{stall}} * \underset{\text{cycles}}{\text{stall}}\right) \text{clocks}$$

↳ % instruction causing dependency.

→ Effective speedup, $S_{eff} = \dfrac{S_{ideal}}{1 + \underset{\text{freq}}{\text{stall}} * \underset{\text{cycles}}{\text{stall}}}$

$$S_{eff} = \frac{k}{1 + \underset{\text{freq}}{\text{stall}} * \underset{\text{cycles}}{\text{stall}}}$$

k → no of stages

→ Data dependencies may occur due to arithematic instructions.

→ Techniques to deal data dependency are

~~i) Multiple~~

(i) Instruction rescheduling

(ii) Operand forwarding

(iii) Introduction of stall cycles.

(iv) Register Renaming (~~for WAR~~) (for WAR, WAW)

→ Control dependencies may result due to branch instructions.

techniques to deal control dependencies are:

(i) Multiple pipelines.

(ii) Delayed load.

(iii) Prediction techniques.

(iv) Delayed branching

→ Structural dependencies are resulted due to the limitation of resources.

• Resource duplication will resolve this problem.

Ques Consider a 4-stage instruction pipeline with stage delays 20 ns, 15ns, 18ns, 10ns. Inter stage buffer overhead is 5 ns

(i) How much time is needed on an avg per instruction in non-pipelined system.

(ii) How much time is needed on an avg per instruction in a pipelined system.

(iii) Calculate speed up.

Sol:

(i) time = 20 + 15 + 18 + 10

= 63 ns (No buffer registers)

(ii) $CPI_{avg} = 1$ clock

$T_{clock\ period}$ = Max (20, 15, 18, 10) + Buffer overhead

= 20 + 5 = 25 ns

∴ 25 ns

(iii) Speed up = $\frac{63}{25} = 2.52$

Ques In the previous problem, if 100 instructions program is given

(i) time of execution in non-pipelined system

(ii) " " " " pipelined system

(iii) speed up

Sol :

(i) $100 * (20 + 15 + 18 + 10) = 6300$ ns

(ii) Ideal $CPI_{avg} = 1$ clock

Clock period = Max (20, 15, 18, 10) + 5

= 25 ns

for 100 instruction is 4-stage pipeline

no of clocks req = $4 * 1 + 99$

= 103

∴ time req = $103 * 25$

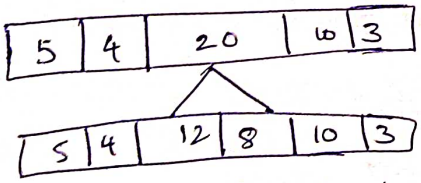
= 2575 ns

(iii) speed up = $\frac{6300}{2575} = 2.446$

Split Pipelines :

→ Identify max delay pipeline and split it into further stages and thus we can decrease clock period. Thus performance increases.

Eg :



5-stage

6-stage

Assum buffer overhead = 2 ns

Clock period of 5-stage = 20 + 2 = 22 (i.e., avg time for execution of instruction)

Clock period of 6-stage = 12 + 2 = 14

∴ Speed up = $\frac{22}{14} = \frac{11}{7} = 1.57$

Qus A non-pipelined system is operating with 2.5GHz takes 5 clocks on an avg per instruction. ~~5~~ 5-stage instruction pipeline due to stage management operates with 2GHz clock. A program containg 30% memory operations, 60% ALU operations and rest branch operations. The cache miss result 5% of memory operations 50 clock cycle stalls, 50% of branch operations result 2 clock stalls. What is speed up of pipeline over ~~non-pipe~~ non-pipeline system.

Sol:

Avg execution time per instruction in non-pipelined system $= 5 * \frac{1}{2.5 \times 10^9}$

$= 2 \times 10^{-9}$

$= 2ns$

In pipeline System

$CPI_{avg} = 1$

clock period $\phi = \frac{1}{2 \times 10^9} = 0.5ns$

no of stall cycles $= 0.3(0.05)(50) + 0.1(0.5)(2)$

(0.3 → Memory, 0.1 → Branch)

$= 15(0.05) + 2(0.05)$

$= 17(0.05) = 0.85$ cycle per instruction

∴ avg execution time per instruction ~~$= 0.5 + 0.85$~~

~~$= 1.35ns$~~

$= 1 + 0.85$

$= 1.85$ cycles

clock $= 1.85 \times 0.5ns$

∴ speed up ~~1.35~~

Speed up $= \frac{2ns}{1.85 \times 0.5ns} = \frac{4}{1.85} \cong 2.16$

(Qu6) A pipeline system operates with 2 GHz clock and result in 2-stall cycles for 30% of instructions. Non-pipeline system operates with 2.5 GHz clock and takes 5 clock on avg. Find speed up

Sol:

for non-pipeline

$$t_{avg} = 5 * \frac{1}{2.5 \times 10^9} = 2\,ns$$

for pipeline

$$\text{clock period} = \frac{1}{2 \times 10^9} = 0.5\,ns$$

avg no of clocks per instruction = 1 + stall cycles

$$= 1 + 0.3(2)$$
$$= 1 + 0.6$$
$$= 1.6 \text{ cycles}$$

∴ avg time per instruction = $1.6 \times 0.5 = 0.8$

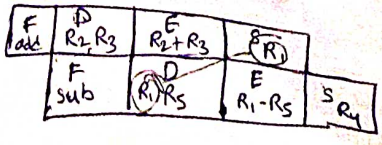
∴ Speed up = $\frac{2}{0.8} = 2.5$

Data Dependencies:

→ Data dependencies result when one instruction in pipeline is waiting for the result of earlier instruction which is not yet computed.

Eg: Consider I_1: ADD R_1, R_2, R_5 --- $R_1 = R_2 + R_3$

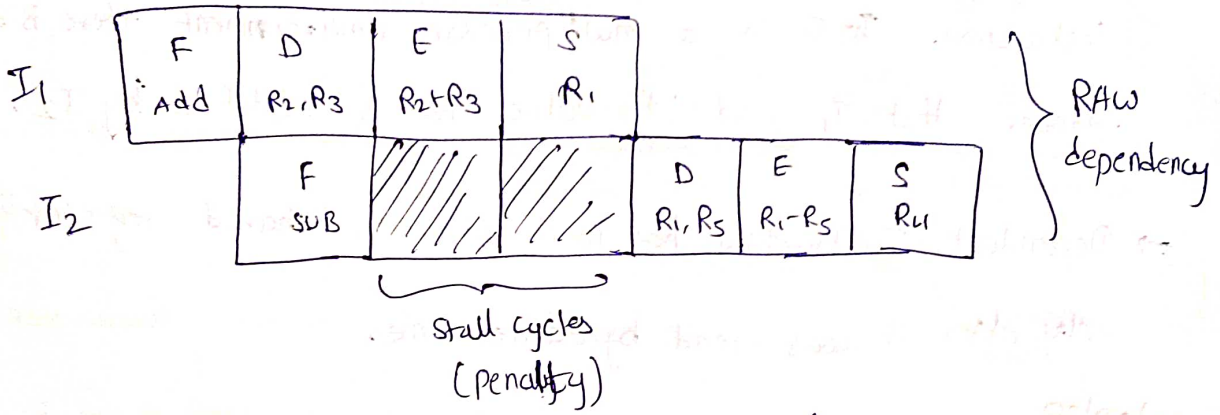
I_2: SUB R_4, R_1, R_5 --- $R_4 = R_1 - R_5$



• Here I_2 is using old value of R_1. Actually it should use the new value of R_1 modified by I_1.

∴ we say I_2 is having data dependency over I_1.

So the correct way of implementation is



Now with correct implementation no of clocks increases to 7 from 5.

Thus performance is reduced.

types of data dependencies:

→ 4 types are possible based on how shared register is used b/w the instructions.

(i) RAW (I_2 → I_1) (Read After Write) (True or flow dependency)

(ii) WAR (Write After Read) (Anti flow or Anti dependency)

(iii) WAW (Write After Write) (O/P dependency)

(iv) RAR (Read After Read) (I/P dependency or No dependency)

RAW (Flow dependency)

→ depedent instruction has to read shared register only after it was written by the earlier one

WAR

WAR (Antiflow):

Consider

I_1: SUB R_1, R_2, R_3 --- $R_1 = R_2 - R_3$

I_2: OR R_2, R_2, R_4 --- $R_2 = R_2 \mid R_4$

Generally Arithematic instruction takes more times than logical instruction. So in a multiprocessor environment there is a danger that I_1 read R_2 value that is modified by I_2.

→ Dependent Instruction has to write into shared register only after it was read by earlier one.

03/09/20

WAW (Output dependency)

Consider

I_1: ADD R_1, R_2, R_3 ---- $R_1 = R_2 + R_3$

I_2: ADD R_1, R_4, R_5 --- $R_1 = R_4 + R_5$

Here destination register is same for both I_1 & I_2

Second instruction has to write into R_1 only after I_1 does.

RAR (Input Dependency)

Consider

I_1: ADD R_1, R_2, R_3

I_2: ADD R_4, R_2, R_5

R_2 is source for the instructions.

(Q47) 6-stage pipeline instruction pipeline has perfectly balanced stages and assume no buffer overhead. What is the speed of this pipeline if it results 2-stalls for 25% instructions compared to non-pipeline system.

Sol:

CPI for non-pipeline = 6

CPI for pipeline = $1 + \left(\text{Stall freq}\right)\left(\text{Stall Penalty}\right)$

$= 1 + 0.25(2) = 1.5$

$\therefore$ speed up $= \frac{6}{1.5} = 4$

Q8) A 4-stage instruction pipeline delays in (picoseconds) are 800, 500, 400, 300. The first stage is replaced with two stages of delay 600, 350.

The throughput increase with this design is _____ percent compared to 4 stage instruction pipeline

Sol:

$CPI_1 = 800ps$

$CPI_2 = 600ps$

throughput is no of instructions per second

$\therefore$ throughput increase % $= \frac{\frac{1}{600} - \frac{1}{800}}{\frac{1}{800}} \times 100$

$= \frac{\frac{1}{6} - \frac{1}{8}}{\frac{1}{8}} \times 100 = \frac{48-36}{48} \times 8 \times 100$

$= \frac{8-6}{\frac{48}{8}} \times 100 = \frac{2}{48} \times 8 \times 100$

$= 33.33\%$

Detection of data dependencies:

Domain (source register) and Range (destination register) relations can be used to detect data dependencies.

Eg:

	Domain	Range
I_1: ADD R_1, R_1, R_3	$\{R_1, R_3\}$	$\{R_1\}$
I_2: SUB R_4, R_1, R_5	$\{R_1, R_5\}$	$\{R_4\}$

$D_2 \cap R_1 \neq \phi \Rightarrow$ RAW

$R_2 \cap D_1 \neq \phi \Rightarrow$ WAR

$D_2 \cap D_1 \neq \phi \Rightarrow$ ~~WAR~~ RAR

$R_1 \cap R_2 \neq \phi \Rightarrow$ ~~RAR~~ WAW

→ Since compiler knows these ~~dependencies~~ domain & Ranges at compilation, it is possible to detect data dependencies during compilation.

→ DAG is used at the time of compilation. to know which instruction are dependent on whom.

Handling Data Dependencies:

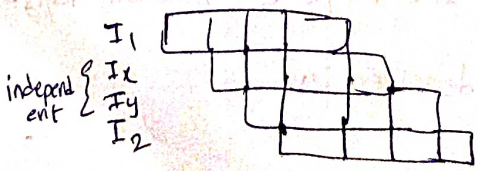
i) Instruction rescheduling:

→ Here compiler places independent instruction b/w dependent ones. The performance of this approach depends on the program & ability of compiler.

→ Based on no of stall cycles required b/w the dependent instructions, no of independent instructions to be place is decided.

Eg: If there are 2 stall cycles b/w two data dependent instructions, then we need to place 2 independent instruction b/w them.

If I_1 & I_2 are dependent



→ Thus there will be no performance loss with instruction rescheduling.

→ However, here, the main problem is identifying the independent instructions (I_x & I_y) in the program.

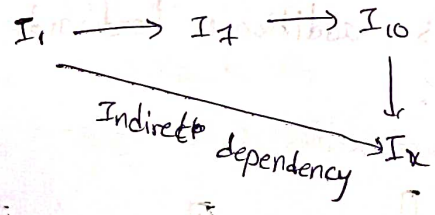
It is because

* I_x & I_y should be independent blw I_x & I_y also blw I_1 & I_2. Along with that they should also be independent from successors of I_1, I_x, I_y & I_2.

This requirement makes identifying such instructions difficult.

→ Detecting direct dependencies is easy, but detecting indirect dependencies is hard. It is NP-hard problem.

Eg: If I_7 is dependent on I_1, I_{10} is dependent on I_7 and I_x is dependent on I_{10} then we say I_x is indirectly dependent on I_1.



(ii) Operand Forwarding:

* used to resolve data dependency.
* Here operand value (before it is stored) is forwarded to the required stage using inter-stage buffer registers.
* only one dependent operand is allowed.

Data hazards can be also be handled at software level by inserting which the compiler inserts NOP blw dependent instruction. Now compiler may rearrange instruction if need. no of NOP inserted = no of stalls

* forwarding can ~~be done~~ not be done to previous clocks.

* operand forwarding can't resolve the data dependencies, if execution stage is computing effective address of the operand (memory word) and result stall cycles.

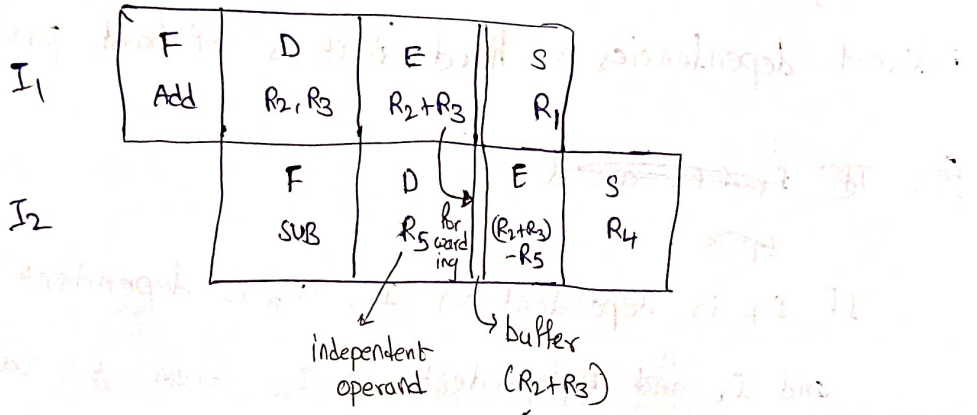
Consider

I_1: ADD R_1 R_2 R_3 - - - $R_1 = R_2 + R_3$

I_2: SUB R_4 R_1 R_5 - - - $R_4 = R_1 - R_5$

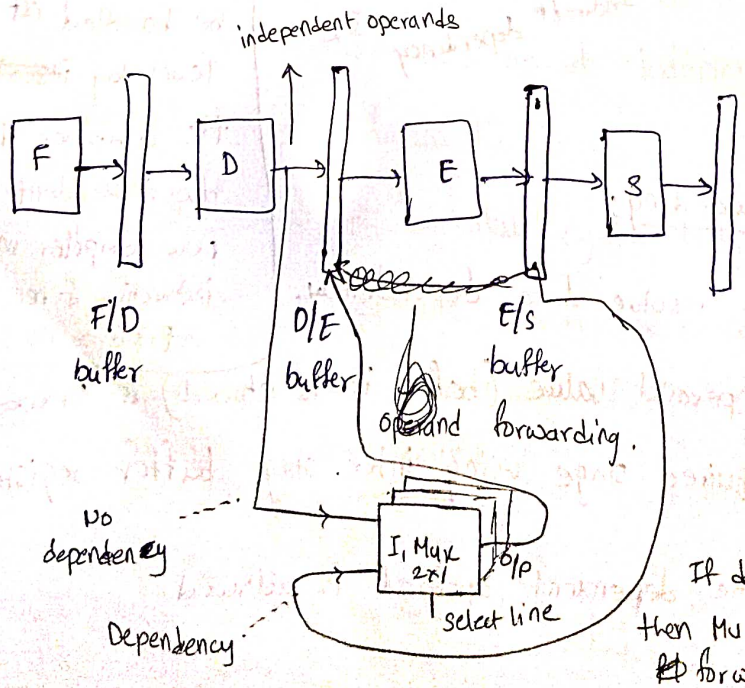
(i.e., $(R_2+R_3) - R_5$)

So here we forward R_1 value before it is stored



(dependent operand is forwarded)

→ This approach requires additional hardware & cost of implementation is more.



If there is dependency then Mux does operator forwarding.

→ Selection on Mux is done by control unit using the information from compiler.

→ Mux can send only one bit of data, thus if size of data bus is 16 bits, then we need 16 - 2×1 Muxes. This is the additional hardware required for implementation of operand forwarding.

→ Since executional unit can product only one o/p atmost, so only one operand can be forwarded.

(Q49) Consider 5-stage pipelined processor with IF, ID, OF, PO, WO stages. Except PO stage all other stages take one clock each for any instruction. PO stage takes 1 clock for ADD and SUB instruction, 3 clocks for MUL instruction and 6 clocks for DIV instruction respectively. Operand forwarding is used in the pipeline. what will be no of clocks required to execute the following sequence of instructions.

I_0 : MUL R_2, R_0, R_1

I_2 : DIV R_5 R_3 R_4

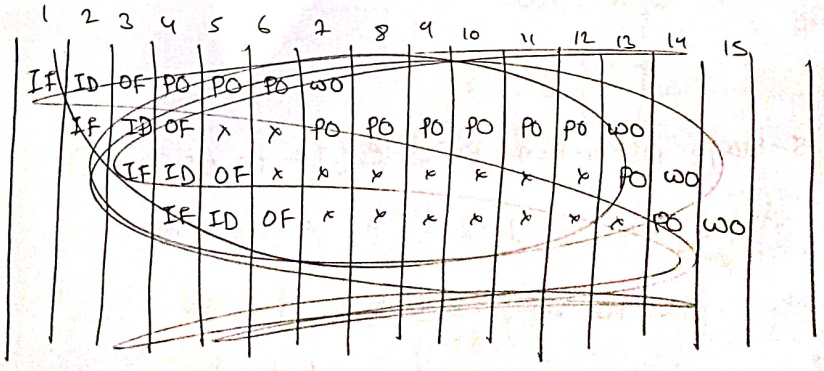
I_3 : ADD R_2 R_5 R_2

I_4 : SUB R_5 R_2 R_6

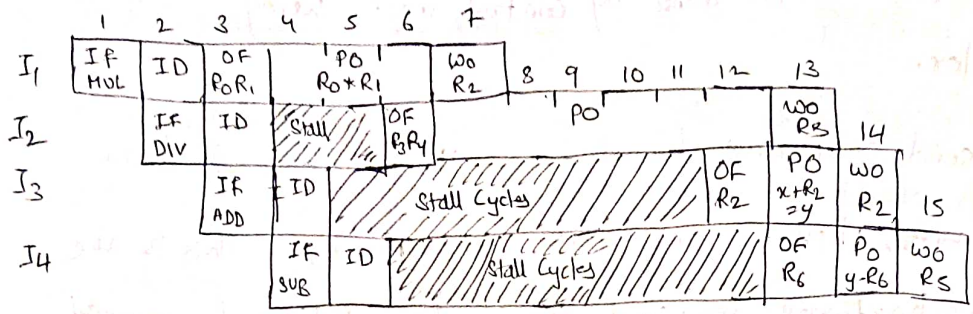
PO : perform Operation

WO : write operand.

Sol :



∴ 15 clocks.



∴ 15 clocks.

★ Here the reason for not performing OF of I_2 at 4th clock is due to design constraints.

★ i.e., Data should not be exposed.

★ i.e., Immediately after fetching we perform PO so that data won't be exposed much.

★ Method 2: (short cut)

Except one stage all other stages take one each

MUL R_2 R_0 R_1
DIV R_5 R_3 R_4
ADD R_2 R_5 R_2
SUB R_5 R_2 R_6

no of clock req

$= (k+n-1)$ + extra clocks

$= (5+4-1) + 5 + 2$ +0 +0

(extra for DIV, extra for MUL)

$= 15$

→ This shortcut works only if the condition is that except one stage rest takes 1 clock.

(Q50) A RISC processor of 5-stage instruction pipeline (IF, ID, OF, PO, WB). Except PO, all other stages take 1 clock each. A program sequence of 100 instruction the PO takes 3 clocks for 40 instructions, 2 clocks for 35 instructions and 1 clock for remaining 25 instructions. How many clock are req to complete

program sequence. Assume there are no hazards and no buffer overhead.

Sol:

no of clocks req = (k+n-1) clocks + extra clocks

= (5+99) + 2*40 + 1*35

= 104 + 80 + 35

= 219 clocks.

Control Dependencies:

→ Control dependencies result due to change in flow of execution.

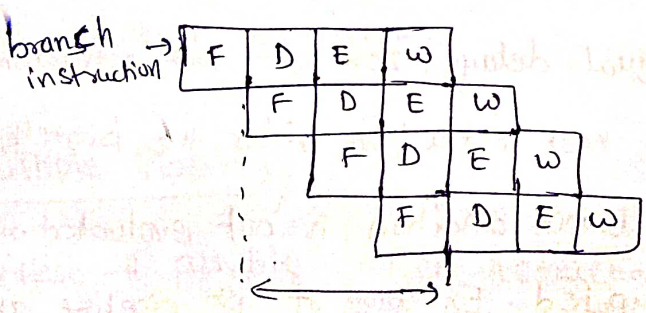
Branch instruction may result change in the flow of execution.

→ Pipeline contents are to reorganized due to control dependencies.

→ This may take stall cycles from 1 to k-1 based on intelligence of the pipeline.

→ $CPI_{avg} = \left(1 + \text{Stall freq} * \text{Stall cycles}\right)$

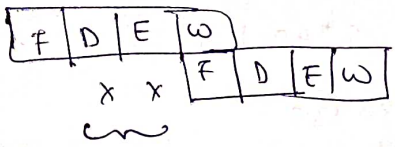
→ Effective speed up = $\dfrac{k}{1 + \text{Stall freq} * \text{Stall cycles}}$



~~Target of branch instruction will not be fetched until branch instruction is fetched~~

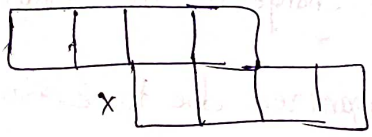
→ Thus in the ~~worst~~ worst case k-1 stall cycles are possible.

→ For the previous situation, if execution stage is capable of evaluating branch condition and compute address of ~~the~~ target instruction, the no of ~~#~~ stall cycles = 3



2 stall cycle.

→ If the branch is unconditional then decoding state itself can know the nature of instruction and computes target instruction address



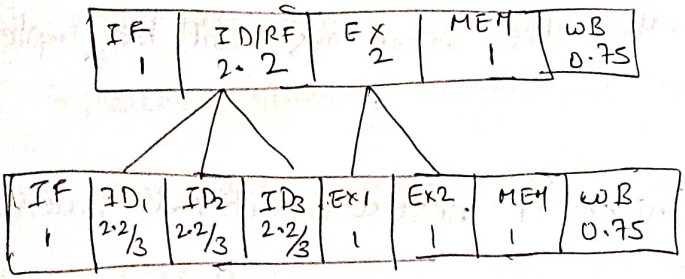
Stall cycle = 1

1 is minimum stall

∴ from this we say no of stall cycles may vary from 1 to k-1 based on the intelligence of pipeline.

(Q51) A five stage pipeline (IF, ID/RF, EX, MEM, WB) is having delay (ns) 1, 2.2, 2, 1, 0.75. In order to improve the performance the stage ID/RF is split into 3 stages of equal delays. Further Ex stage is split into 2-stages of equal delays. 20% of the instructions are branch instructions. The next instruction after the branch will not be fetched until branch condition is not evaluated and target address is not computed. Ex stage of 1st pipeline and Ex2 stage of 2nd pipeline is having this ability. what is the performance gain of 2nd pipeline over the 1st one.

Sol:

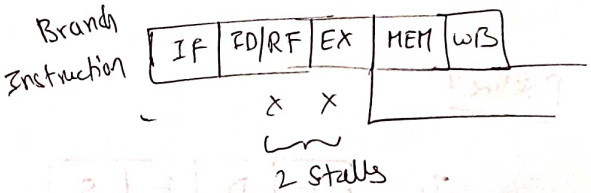


~~CPI1 = 2.2ns~~ clock period for pipeline 1 = 2.2ns

~~CPI2 = 1ns~~ clock period for pipeline 2 = 1ns

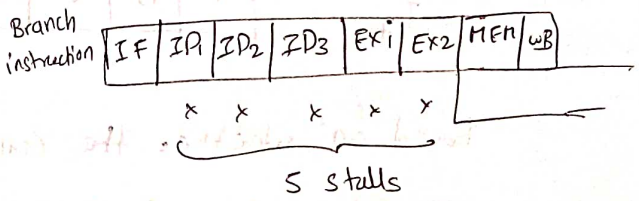
for Pipeline 1

$$CPI_{avg} = 1 + (0.2)(2)$$
$$= 1.4 \text{ clocks}$$



for pipeline 2

$$CPI_{avg} = 1 + 0.2(5)$$
$$= 2 \text{ clocks.}$$



performance gaine = time for pipline 1 / time for pipeline 2

$$= \frac{1.4 * 2.2}{2 * 1} = 1.54$$

Handling Control Dependencies:

ii) Multiple Pipelines:

→ Here all possible target instructions after the branch instruction are placed in different pipelines.

→ At the end of the branch instruction, only one of them is allowed to progress (it contains correct flow) and all others are stopped.

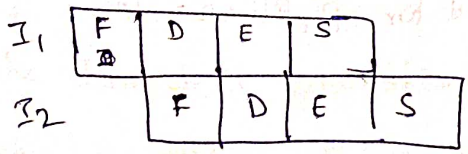
→ Here drawback is that, cost of implementation is more.

→ in general, however, if-then-else instructions will be implemented in 2-pipeline system

→ However, we cannot implement nested conditional statements with 2-pipeline system.

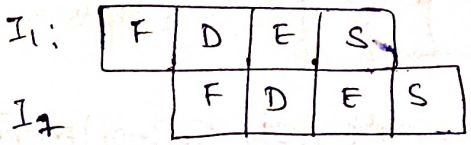
Consider below code implementation with 2-pipeline system

Pipeline1




```
I1 : if (---)
        I2;
     else
        I7;
```

Pipeline 2

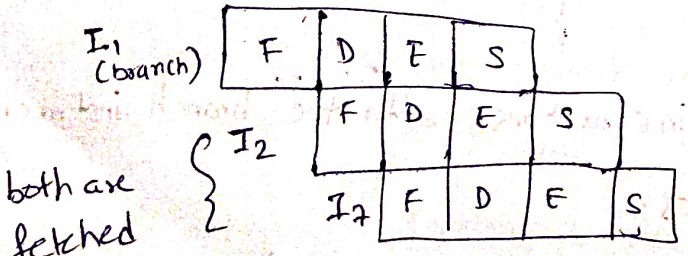


Based on whether the condition is is true or not, one of the above two pipelines proceeds with the execution.

→ If depth of nesting is 'n' then we need n-pipelines.

(ii) Delay Slot (or) Delay load

In case of branch instruction, the next sequential target instruction are fetched for subsequent clocks.



So here irrespective of condition 1-delay slot happen (i.e., 1 Stall cycle)

(Q52) A 5-stage instruction pipeline is used to implement the program that contain 20% branch instructions. The first stage is fetch and the instruction following branch will not be fetched until the branch instruction is completed. The no of stalls and avg CPI of this pipeline is

Sol:

$$CPI_{avg} = 1 + 0.2(4)$$
$$= 1.8$$

∴ 4, 1.8

(Q53) In (Q52), if branch is taken by branch instruction then only stall cycles occur otherwise no stall cycles. Assume among the branch instructions 40% are unconditional branches and 50% of conditional branch instruction, branch will not be taken. The average CPI is ——

Sol:

$$CPI_{avg} = 1 + \underbrace{(0.2)(0.4)(4)}_{\text{unconditional branch}} + \underbrace{(0.2)(0.6)(0.5)(4)}_{\text{conditional branch}}$$

$$= 1 + 0.32 + 0.24$$
$$= 1.56$$

Note:

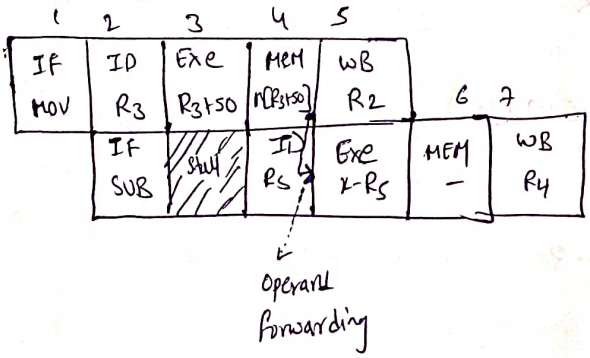
~~I: Move~~

I_1: MOV R_2, 50(R_3)

I_2: SUB R_4, R_2, R_5

~~The dep~~ I_1 & I_2 has RAW dependency.

Since we are using computable effective address, Operand forwarding technique doesn't work in this case. So even after using operator forwarding, we get one stall cycle.



(iii) Prediction:

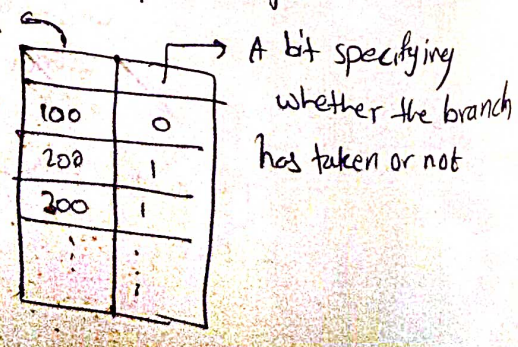
Prediction for branch instruction is

Static Predictions (Compile time)
- (i) Branch always takes (fetch: branch target)
- (ii) Branch never takes (fetch: next sequential)

Dynamic Predictions:

1) BHT (Branch history table)

* Prediction will depend on status of previous predicted results.
* Majority previous decisions will decide the prediction.



Limitation:
i) the size of the table grows with no of branch instructions.

ii) Un-related previous decisions influence current prediction

Consider

```
for (i=0; i<1000; i++)
{
    a[i] = b[i] + 5
}
if (x>y)
{
    :
}
```

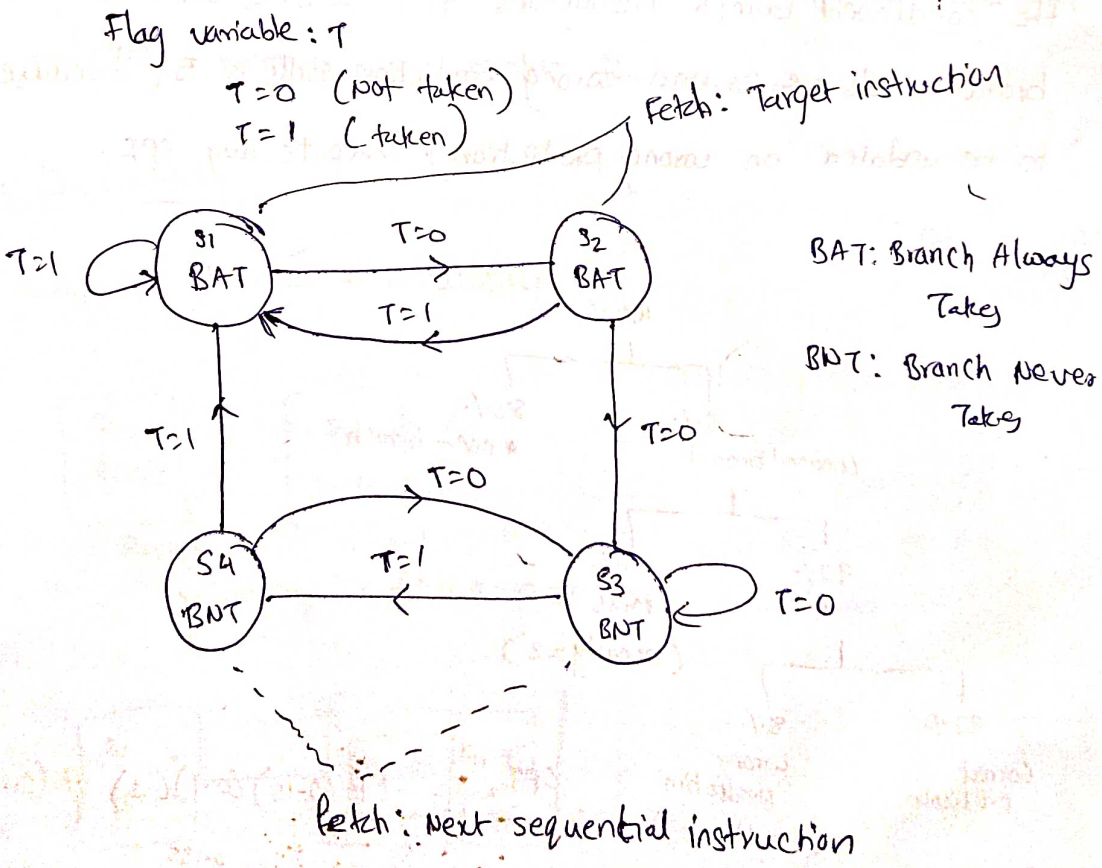
} there conditions true 1000 times and false one time

} So it is predicted that this condition could also be true even through x>y is related to i<1000 in no way

So this another limitation.

2) 2-bit dynamic prediction:

Whenever two ~~consequtiv~~ consecutive previous predictions are wrong, then it changes the prediction



→ Since we have 4 states, we need 2 bits to implement this. So we need only 2 flipflops which very less cost compared to BHT.

3) Branch Target Buffer:

→ It is a small cache which maintains the target address of the conditional branch instructions. and prediction bits.

→ If it result in a hit and prediction is correct, then no penalty.

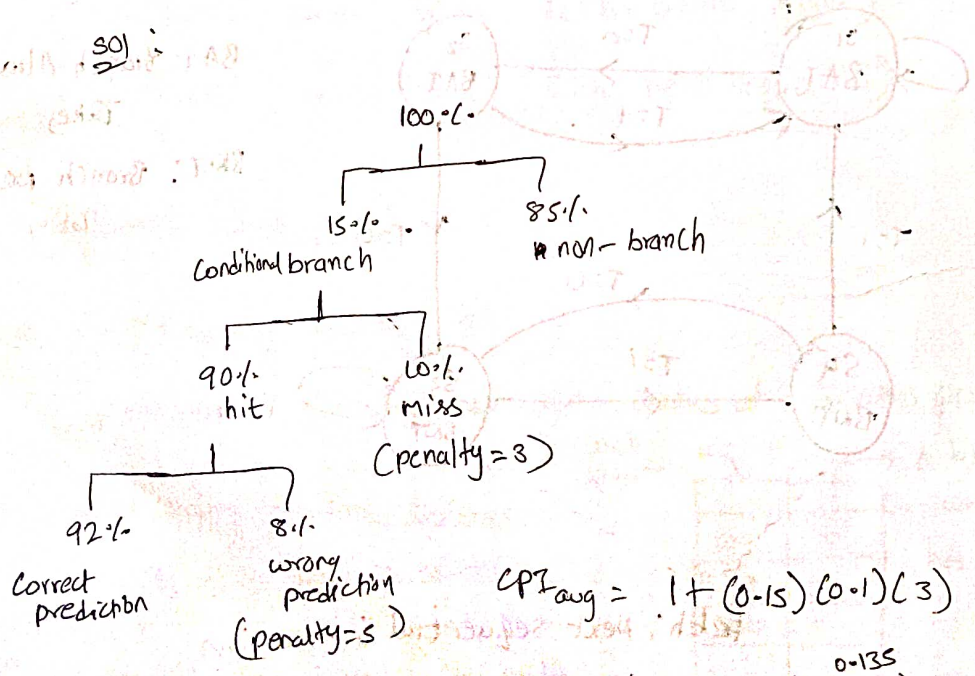
→ BTB cache miss penalty is normal miss penalty. (1 to k-1 stalls)

→ BTB hit & wrong prediction penalty is few clocks more than normal penalty as BTB has to be updated and the target is to be brought into BTB.

Miss ⟹ normal penalty
BTB hit + correct prediction ⟹ no penalty
BTB hit + wrong prediction ⟹ extra penalty

Q54) Consider a BTB is employed in the system to deal with conditional branch instructions. There are 15% conditional branch instructions and 90% hit for BTB. The prediction of BTB is branch always takes. Among the conditional branch instructions 92% satisfies condition. Let normal branch stalls are 3 and wrong prediction stalls is 5 (because BTB is to be updated on wrong prediction). Compute avg CPI.

Sol:



$CPI_{avg} = 1 + (0.15)(0.1)(3) + (0.15)(0.9)(0.08)(5)$

$= 1 + 0.045 + (0.135)(0.4)$

$= 1 + 0.045 + 0.054$

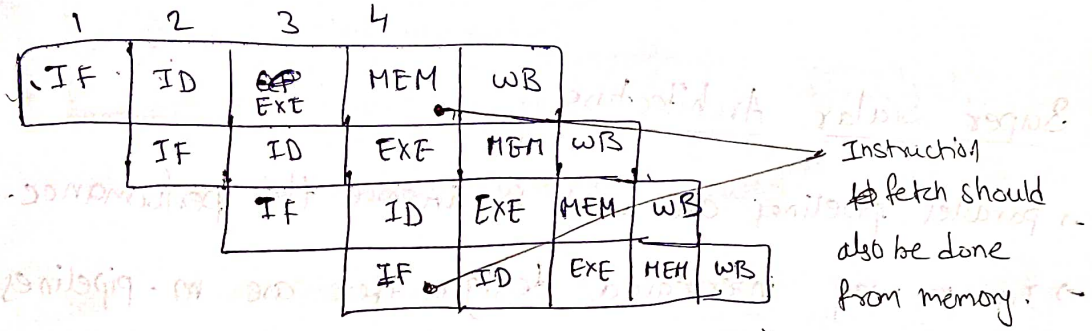
$= 1.099$

Structural Dependencies

→ It results due to non-availability of resources for different stages of pipeline.

→ ~~Result~~ Resource duplication will solve this problem.

Single port memory ~~does~~ doesn't allow two different memory operations at the same time for two different stages.

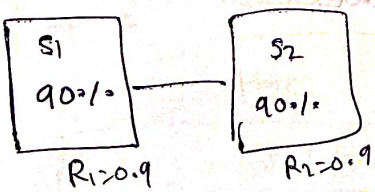


At the 4th clock we need two different memory operations.

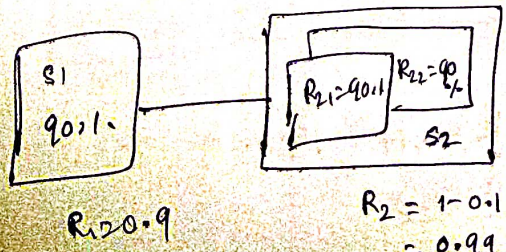
→ So instead of having single port memory, having Dual-port or multiport memory can resolve the issue.

Resource Duplication:

It is subjected to cost & reliability



reliability = 0.9 * 0.9 = 0.81 ≅ 81%.



$R_2 = 1 - 0.1 * 0.1$

$= 0.99$

∴ reliability = 0.9 * 0.99 ≅ 90%.

Eg: In a 5-stage instruction pipeline, 10% instructions result data dependencies, 20% instructions result control dependencies, 15% instructions result structural dependencies. If stall cycles are 3, 2, 1 respectively. Find CPI_{avg}

Sol:

$$CPI_{avg} = 1 + 0.1(3) + 0.2(2) + (0.15)(1)$$

$$= 1 + 0.3 + 0.4 + 0.15$$

$$CPI_{avg} = 1.85$$

Super Scalar Architecture

→ parallel pipelines are used to improve the performance.

→ In m-way interleaved design there are m-pipelines operated such that in the first clock m instructions are placed in fetch stage of the pipelines.

→ If these pipelines are having k-stages then in m-instructions are implemented in first k-clocks.

→ The time taken for n instruction in this m-way interleaved k-stage pipelines is

$$T_n = (k + (n-m)/m) \text{ clocks.}$$

→ speed up factor, $S = \dfrac{n * k}{k + \frac{n-m}{m}}$

if $n >>> 1$ then

$$S = m * k$$

08/09/20

clock 1: $I_1, I_2, I_3 \cdots I_m$ (fetch)

Clock 2: $I_{m+1}, I_{m+2}, I_{m+3}, \cdots I_{2m}$ (fetch)

Thus if it is a k-stage pipeline, after kth clock, m instructions would finish their execution.

From then onwards, m instructions would finish their execution per each clock.

If total no of instructions are n,

1st k clocks → m instruction

no n-m instructions are left for which m instructions will be executed for each clock.

∴ n-m instructions require $\frac{n-m}{m}$ clocks.

∴ time required to finish n instructions are,

$$T_n = k + \frac{n-m}{m}$$

If m=1, i.e., single pipeline

$T_n = k+n-1$ clocks

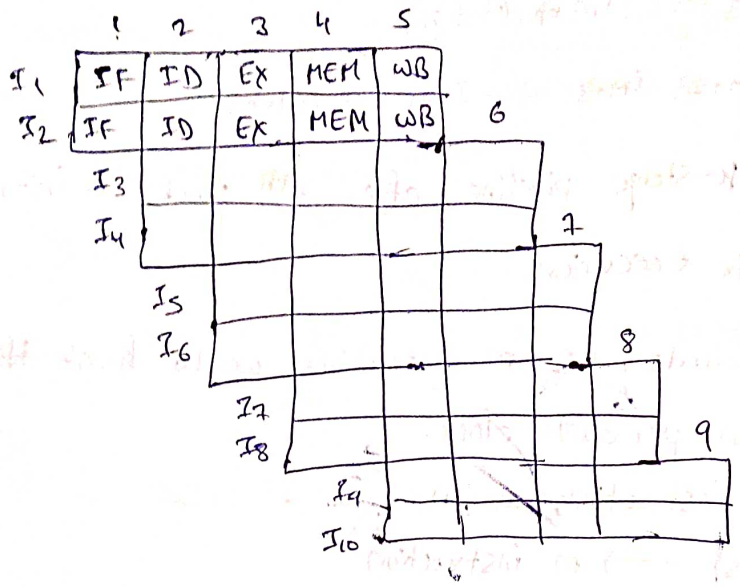
$$\text{Speed up} = \frac{\text{time without pipeline}}{\text{time with pipeline}}$$

$$S = \frac{n*k}{k + \frac{n-m}{m}}$$

if $n >>> 1$ then

$$S = \frac{n*k}{\frac{n}{m}} \Rightarrow S = m*k$$

2-way interleaved pipeline



∴ pipline 1: I_1, I_3, I_5, I_7, I_9

pipeline 2: $I_2, I_4, I_6, I_8, I_{10}$

∴ 10 instruction are executed in 9 clocks.

ie, CPIavg < 1

→ However to do this kind of execution the instructions must be independent.

(Qss) Consider 4-way interleaved super scalar architecture where 4 pipelines of 6-stages are operated. Assume 1000 independent instructions are to be implemented. how many clocks are taken by this super scalar architecture. Assume there are no hazards

Sol:

After 6 (k) clocks 4 (n) instruction are executed.

From now on 4 instructions are executed per each clock.

∴ no of clocks req = $6 + \frac{1000-4}{4}$

$= 6 + \frac{996}{4} = 6 + 249 = 255$

Also speedup ≅ m*n

4*6 = 24

$\frac{1000 * 6}{255} = 23.52$

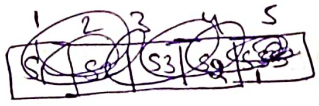
Non-linear Pipeline:

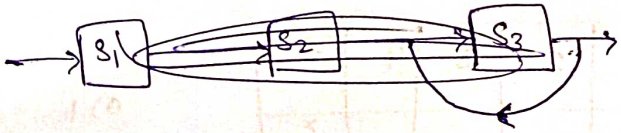
- In non-linear pipeline an instruction can visit the same stage more than once and hence CPI$_{avg}$ > 1
- Reservation table indicate the usage of the stages at different clocks. We can detect conflict free scheduling using reservation table.
- The scheduling gap blw the successive instructions is always greater than 1

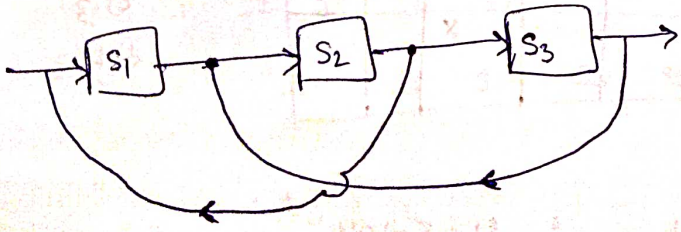
Eg:

3-stage pipeline reservation table

Stage / clock	1	2	3	4	5
S1	X				X
S2		X		X	
S3			X		







Conflict: More than one instruction trying to occupy same stage at same clock.

Latency: The scheduling gap in clocks b/w successive instructions without causing conflicts in the instruction pipeline.
Many possible values are there for latency.

Minimum Available Latency (MAL):

Min no of clocks req for scheduling non-linear pipeline without resulting in conflicts.

Forbidden Latency of a stage (Hidden conflicts):

The time gap in clocks b/w successive usages in the stage. It denote conflicts.

Forbidden vector:

The binary representation of the forbidden latencies of all the stages. The length of the forbidden vector indicate the maximum value of forbidden latency among the stages. If the binary bit is one in position 'k' then it denote conflict for k^{th} clock.

→ For the previous example of reservation table,

forbidden latency of stage S1 = 5-1 = 4

Qs6) For the following reservation table of a pipeline with stages S1, S2 and S3 the minimum available latency is ____

Stage \ clock	1	2	3	4	5
S1	X				X
S2		X		X	
S3			X		

a) 1
b) 2
c) 3
d) 4

Sol:

forbidden lantency of S1 = 5-1 = 4

" " " S2 = 4-2 = 2

" ____ S3 doesn't have forbidden latency (∵ it is used only one)

Max forbidden latency = 4

∴ length of forbidden vector = 4

forbidden vector:

Clock	1	2	3	4
binary bit:	0	1	0	1

∴ forbidden vector : 0101

From this we can say, if latency is 2 or 4 it results in conflict

∴ Ans : 3

Method 2:

If latency = 2

I_1: S1 S2 S3 S2 S1

I_2: S1 S2 S3 S4 S5

Conflict

If latency = 4

I_1: S1 S2 S3 S2 S1

I_2: S1 S2 S3 S4 S5

conflict

we know that latency is always greater than 1

~~∴ latency =~~

latency = 1

S1 S2 S3 S2 S1

S1 S2 S3 S2 S1

S1 S2 S3 S2 S1

S1 S2 S3 S2 S1

S1 S2 S3 S2

conflict

If latency = 3

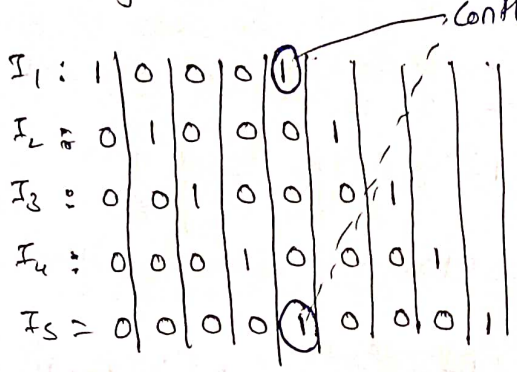
I_1: S1 S2 S3 S2 S1

I_2: S1 S2 S3 S4 S5

∴ Ans: 3

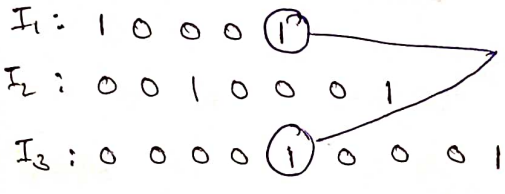
Binary string representing stage utilization

Stage 1 & latency = 1



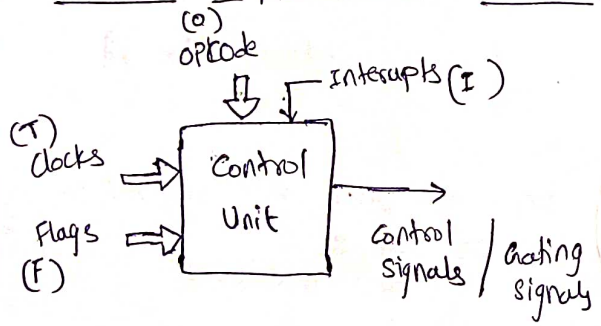
conflict in 5th clock for stage 1, blw instructions I_1 & I_5 for latency = 1

latency = 2



conflict for stage 1 in 5th clock blw instruction I_1 & I_3 for latency = 2

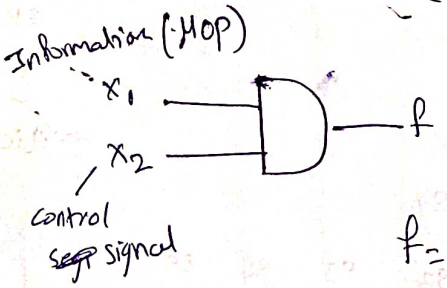
Micro Operations & Control Signals



These signals can be for both internal/external devices

∴ Control signal

$$C = F(T, F, O, I)$$



Mop: micro operation

$f = x_1$, if $x_2 = 1$

$f = 0$, if $x_2 = 0$

∴ x_2 acts like a gate

∴ Control signal do the job of allowing or blocking the information.

Program - Instruction - Micro Operation

* program is collection of sequential instructions.
* Each instruction is collection of sequence of sub operation
  i.e. (Fetch, Exe, interrupt)
* The time req to execute an instruction is called instruction cycle.
* Each suboperation is sequence of elementary basic operations (atomic) which are known as micro operations. These operation can't be further divide.
* The collection of microoperations of fetch is called fetch microprogram.
  Silly we have execute microprogram & interrupt microprogram.
* Microoperation is smallest unit of instruction.
* Microoperations are denoted with RTL (Register transfer language)
* Each micro operation requires set of control signals which needed at the same time.
  So CU generates these signals.

→ If CU uses only Hlw to generate signals, this is called Hlw CU design. This is RISC system.
  Also since only Hlw is involved it faster.

→ If CU uses memory & Hlw to generate signals, this is called Memory control unit. This is CISC system.
  since memory is involved it is slower.

Control Memory

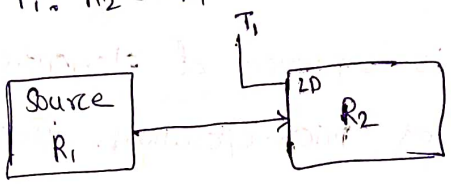
Register Transfer Language

Condition : Micro-operation

condition : destination ← source.

Mop: Transfer data from register R_1 to register R_2 at clock,

RTL: T_1 : $R_2 \leftarrow R_1$



LD: load

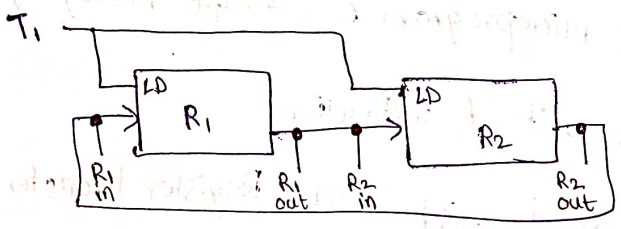
Swap at H/w level.

T_1 : $R_1 \leftarrow R_2$, $R_2 \leftarrow R_1$

S/w level

temp = a
a = b
b = temp

slower



→ Here R_1 in, R_1 out, R_2 in, R_2 out are control signals.

So in this way we can swap the contents in one microoperation.

(Q57) what is the proper arrangement of following instructions if they are implemented in 5-stage instruction pipeline (IF, ID, EXE, MEM, WB) for optimal performance.

I_1 : LD R2, 100(R1)

I_2 : ADD R3, R1, R2

I_3 : LD R4, 100(R5)

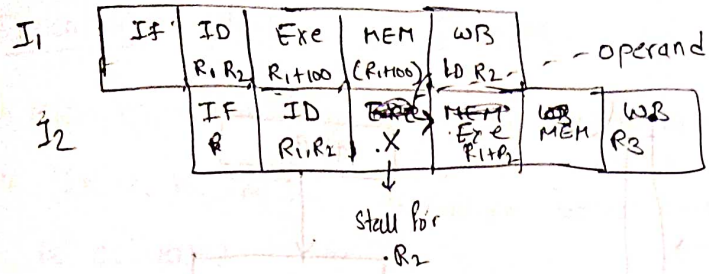
I_4 : ADD R6, R4, R7

a) I_1, I_2, I_3, I_4

b) I_1, I_3, I_2, I_4

c) I_1, I_3, I_4, I_2

d) I_3, I_2, I_1, I_4



Executing I_2 after I_1 gives a stall

Silly Executing I_4 after I_3 also gives a stall

However I_1 is independent from I_3 & I_4

I_2 " " " "

$\therefore I_1\ I_3\ I_2\ I_4$

I_3 fills the stall

$\therefore$ opt (b)

Let us consider a basic system with

PC, IR, acc, memory & MAR, MDR

Constraint: Memory is accessible only to MAR & MDR

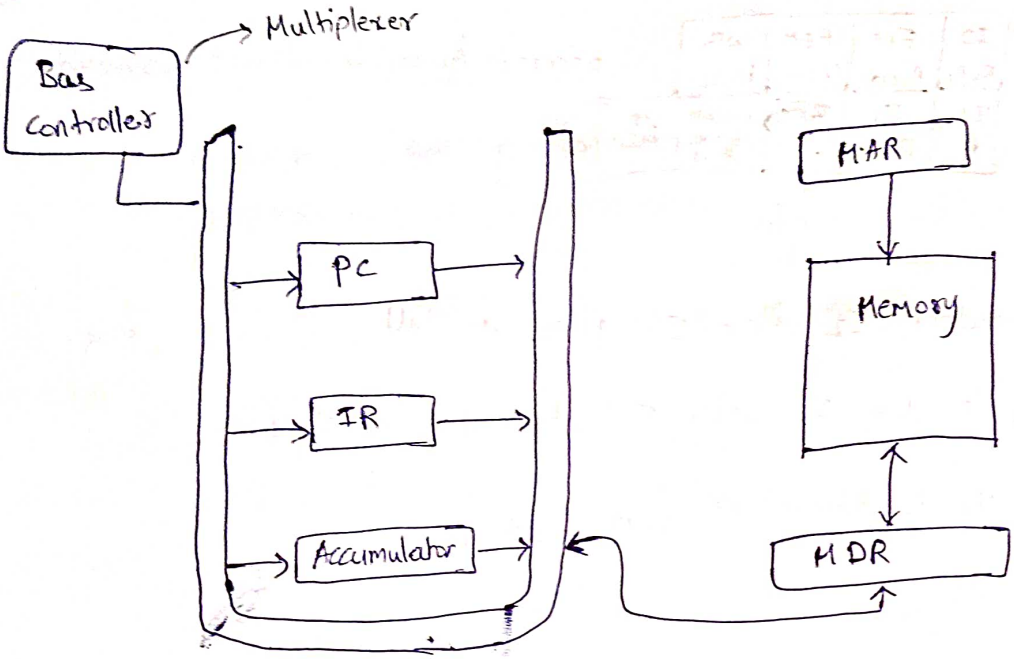
→ single bus architecture is used to connected registers & memory

Now for this hypothetical system, we develop micro instructions for

→ fetch,

→ execute: ADD, Branch, RET etc.

→ INTR (interrupt)



Bus controller tells which register has to put data on the bus or take data from the bus

Sizes of registers

PC = MAR, IR = acc = MDR

Fetch operation:

* Get instruction into IR from memory using the location given by PC.

Steps:

(i) Copy instruction address from PC into MAR

(ii) Read the instruction from memory to MDR

(iii) Copy the instruction from MDR to IR

(iv) Update PC to next instruction address

RTL notation:

T_1 — at clock 1

T_1: MAR ← PC

T_2: MDR ← memory (M[MAR])

T_3: IR ← MDR, PC ← PC+1 → two micro operations are done in parallel.

Executional phase Micro program

→ Micro operation req vary from instruction to instruction.

→ IR is having the instruction before executional micro program is activated.

ADD X instruction executional phase microprogram:

Add contents of memory location X to the accumulator and store the result in accumulator.

Steps:

IR: 

(i) place address X into MAR

(ii) Read the memory contents of location X into MDR

(iii) Add MDR with accumulator. and hold result in accumulator.

RTL:

T_1 : MAR ← IR[operand ref] } operand fetch

T_2 : MDR ← Memory

T_3 : A ← A + MDR

Microprogram for execution phase of BUN X

↳ Branch Unconditional

→ Control to be transfered from current instruction to the location X where the next required instruction is present.

Steps:

PC is to be updated to X

RTL:

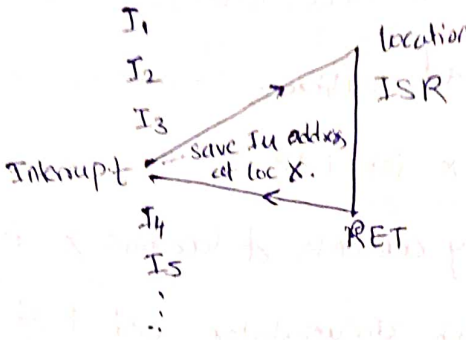
T_1 : PC ← IR[reference]

Interrupt Microprogram

→ All instructions check for interrupt at the end of instruction cycle.

→ when interrupt occurs the current PC contents have to be saved in predefined memory location (say) 'X'. ↳ i.e., return address

Then transfer control to beginning instruction address of ISR. ↳ (say Y)



So we need to store PC in memory & update PC.

RTL:

Saving return address in memory:

T_1 : MAR ← X ..... X is predefined address for saving return address (PC)

T_2 : MDR ← PC

T_3 : Memory ← MDR

T_4 : PC ← Y ..... now PC points to first instruction of ISR.

Microprogram for RET:

→ It takes return address from memory location x and control transfers to that location.

RET:

Steps:

i) Goto location X by placing X into MAR and collect return address in MDR

ii) Update PC with X which is in MDR

RTL:

T_1: MAR ← X

T_2: MDR ← Memory

T_3: PC ← MDR

Note:

Every register has in & out control signals. If "in" is high then value are stored into register. If "out" is high then data in the register is placed on the ~~bus~~ bus.

Eg: $R_1 \leftarrow R_2$

The above instruction stores value of R_2 in R_1.

∴ R_1 in & R_2 out signals have to be high simultaneously.

~~So contro~~

In this way control signals implement micro operations.

⇒ MDR ← Memory

⇒ MAR_{out}, MDR_{in} ~~are high~~, MEMR are activated

↳ memory read

So in this way for every ~~control~~ in microprogram CU activates appropriate control signals

Control Unit Design

⇒ CU is responsible to identify which control signal is required when and have a mechanism to activate that control signal at that time.

H/w CU Design

→ Each control signal is expressed as SOP (sum of Product Expression) and realized using dedicated Hardware.

Control signal, C is a function of clock (T), Instruction Phase (I_{ph}) and for which instruction.

Control signal, $C = F(T, I_{ph}; I)$

where $T = \{T_1, T_2, T_3 \dots T_k\}$

$I_{ph} = \{\underset{00}{\text{fetch}}, \underset{01}{\text{Exe}}, \underset{1x}{\text{interrupt}}\}$

$I = \{I_1, I_2, I_3, \dots I_m\}$

Eg: Consider HWCU design which support 2 instructions I_1, I_2 and using 4 control signal C_1, C_2, C_3, C_4

Let each instruction require 4-micro operations.

The following table describes the above details

Mop.	Control Signals	
	I_1	I_2
T_1	C_1, C_2	C_1, C_4
T_2	C_2, C_4	C_1, C_3
T_3	C_2, C_1	C_1, C_3
T_4	C_4, C_4	C_4, C_2

Here C is function clock (T) & instruction I.

$C = F(T, I)$

where $T = \{T_1, T_2, T_3, T_4\}$

$I = \{I_1, I_2\}$

Expression)

on Phase (IPH)

4

S

SOP for Control signals

$C_1 = I_1T_1 + I_2T_1 + I_2T_2 + I_1T_3 + I_2T_3 + I_1T_4$

$C_2 = I_1T_1 + I_1T_2 + I_1T_3 + I_2T_4$

$C_3 = I_2T_2 + I_2T_3$

$C_4 = I_2T_1 + I_1T_2 + I_1T_4 + I_2T_4$

Now consider simplification of these SOPs

$C_1 = \underbrace{I_1T_1 + I_2T_1} + I_2T_2 + \underbrace{I_1T_3 + I_2T_3} + I_1T_4$

At T_1, C_1 is require for both I_1 & I_2
silly at T_3

$\therefore C_1 = T_1 + I_2T_2 + T_3 + I_1T_4$

$C_2 = I_1T_1 + I_1T_2 + I_1T_3 + I_2T_4$

$C_3 = I_2T_2 + I_2T_3$

} remains same (no minimization possible)

~~$C_4 = T_2T_1 + I_2T_3$~~

$C_4 = I_2T_1 + I_1T_2 + T_4$

Eg:

MOP	Control Signals	
	I_1	I_2
T_1	C_1	C_1
T_2		C_1
T_3		C_1
T_4		C_1

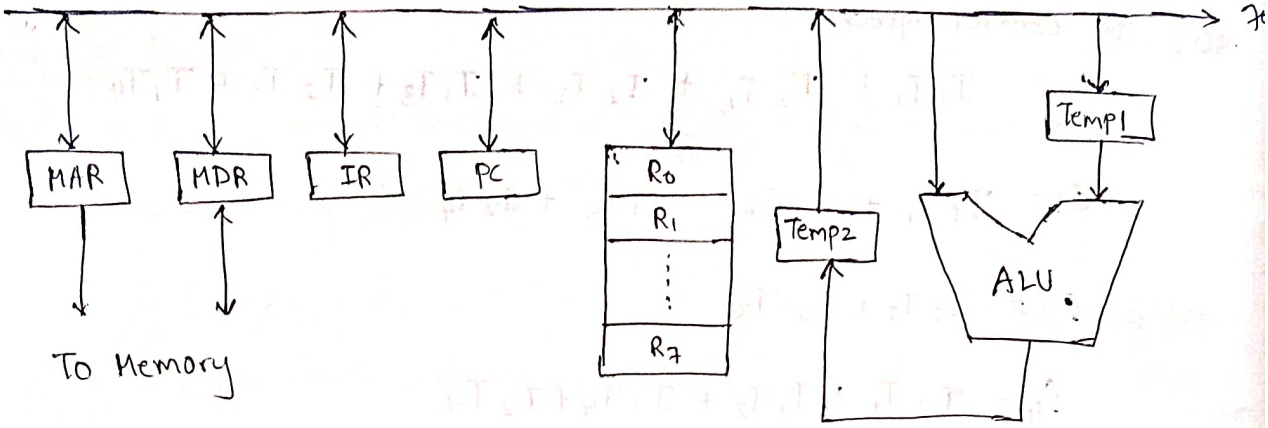
Now $C_1 = \underbrace{I_1T_1 + I_2T_1} + I_2T_2 + I_2T_3 + I_2T_4$

at T_1, C_1 is activated

for I_2, C_1 is always activate

$\therefore C_1$ can be minimized as $C_1 = T_1 + I_2$

Q58



Consider an instruction $R_0 \leftarrow R_1 + R_2$. The following steps are used to execute it over given data path. Assume that PC is incremented appropriately. The subscripts r and w indicate read and write operations respectively.

1. $R2_r$, $TEMP1_r$, ALU_{add}, $TEMP2_w$
2. $R1_r$, $TEMP1_w$
3. PC_r, MAR_w, MEM_r
4. $TEMP2_r$, $R0_w$
5. MDR_r, IR_w

which one of the following is the correct order of execution of the above steps.

a) 2, 1, 4, 5, 3
b) 1, 2, 4, 3, 5
c) 3, 5, 2, 1, 4
d) 3, 5, 1, 2, 4

Sol:

Initially Instruction fetch has to be done

So 3, 5 must be the first 2 instructions

~~now operands has to~~

Now fetching operand R_1, (∴ 2)

Now fetch R_2 & add. (∴ 1)

Now store (∴ 4)

∴ opt (c)

Advantages of Hwcu:

→ Since every thing is Hlw level implementation, control signal can be generated parallelly and hence it is faster. It is used in RISC system.

Disadvantage:

→ Even a minor modification requires re-design and re-connection of Hlw component. ∴ not flexible.

→ More cost of implementation.

→ Hence Hlw CU is not suitable for design & testing

Micro-Programmed CU Design:

→ Binary pattern of control signals is stored in control memory.
↳ (control word)

→ Control signals are generated after accessing control word from control memory and using hardware

→ Any changes confine to binary pattern and ~~hard ware~~ hardware remains unchange. Hence this design is flexible.

→ Based on how the binary pattern is derived Micro program can be
(i) horizontal
(ii) vertical
(iii) diagonal

i) Horizontal Micro program → (1 cw / Mop & 1 bit / control signal)

→ It assigns 1-bit per each control signal

	I_1
T_1	C_1, C_3

⇒

C_4	C_3	C_2	C_1
0	1	0	1

control word for first micro operation of instruction I_1.

Length of control word = no of control signals in the system.

Drawback:

→ Control words are lengthy.

i.e., for example if a system containing 64 signals, and if it requires only 1 signals, still we need to allocate 64 bits for it.

Thus if length is more, access time is more.

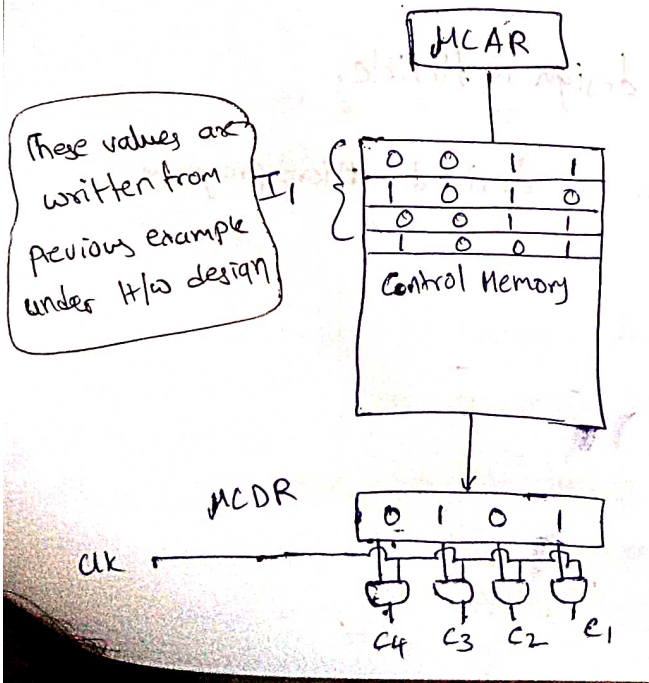
→ Control memory is inefficiently utilized.

↓

i.e., most of the bits in control word are 0's.

Advantage: Provides higher degree of parallelism (DOP)

DOP: no of control signals per control word.



MCAR: Micro Control Address Register

MCDR: Micro Control Data register

→ Control Memory holds one control word for every micro operation.

→ no of control words = no of Mop

Eg: If an ISA has 256 instructions in which each instruction has 16 Mop, then find the no of control words

Sol:

$256 * 16 = 4096$

∴ 4K control words

(ii) Vertical Microprogramming

→ control signal binary pattern is encoded and store in the memory.

Thus if we have k ~~control~~ ~~Mops~~ control signal

then we require $\log_2 k$ bits

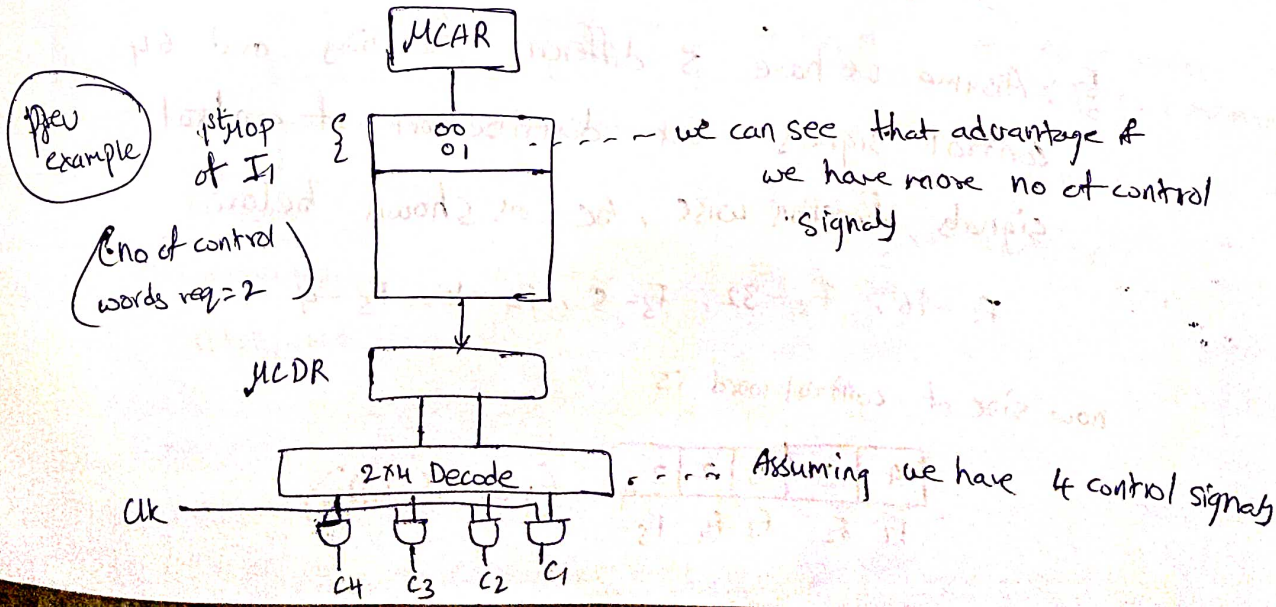
i.e., size of control word = k

→ Hence it results in shorter control words.

Eg: C_1 - 00
C_2 - 01
C_3 - 10
C_4 - 11

→ However for generation control signal, external decoder is need in addition to hardware.

→ So it has shorter control word, but cost is more.



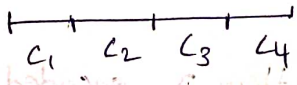
Here each signal is generate one after other.

→ For example consider we need C_1, C_2, C_3, C_4 in 1 μs.

If they are generated ~~sequentially~~ parallelly then 4 MHz clock will be sufficient.

However if it is sequential then we need 4 MHz (time period = 0.25 μs)

i.e.,



Hence vertical micro programming require faster clocks.

Limitations:

→ Degree of Parallelism = 1 (i.e., 1 control signal / 1 word)

→ Entire control word can generate only one signal at a time.

Diagonal Micro programming

→ Control signals are encoded functional wise.

→ Hence control wordsize is bigger than vertical and smaller than horizontal microprogramming

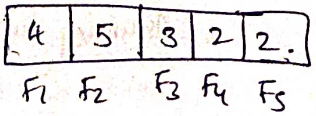
~~Assume we have~~

→ Here we divide control ~~function~~ signals based on the function they do.

Eg: Assume we have 5 different functions and 64 control signals. Let distribution of control signals, function wise, be as shown below

$F_1 - 16$, $F_2 - 32$, $F_3 - 8$, $F_4 - 4$, $F_5 - 4$

now size of control word is



(i.e., 16 bits)

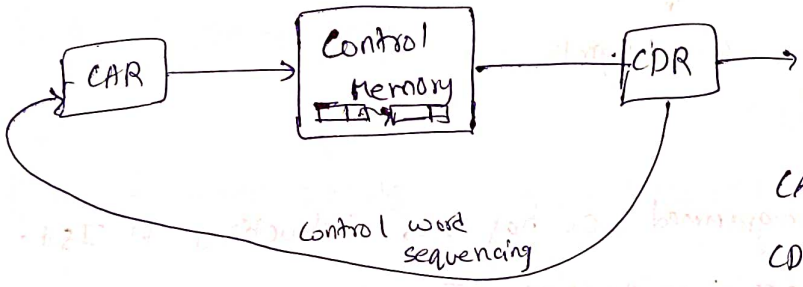
Here DOP = 5

→ DOP is higher than vertical & less than horizantal.

Control Word Sequencing

→ Unless all control words for all micro operations are generated in the sequence, the instruction will not be implemented.

→ Control word sequencing takes the responsibility of providing address of next word to be accessed from control memory.



CAR: Control address Register

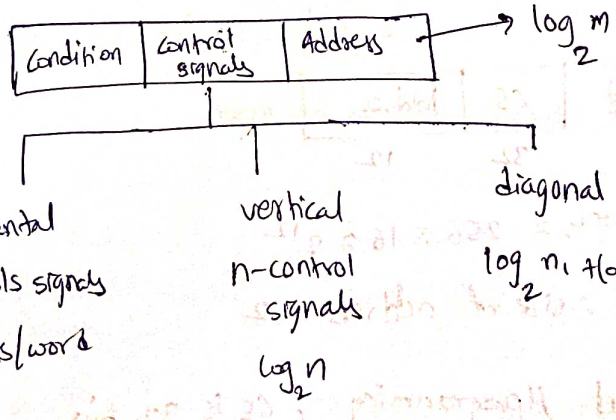
CDR: Control Data Register

A portion of CDR is used to get the next address for CAR.

→ Each control word has link to next address to be accessed.

→ The last instruction has NIL.

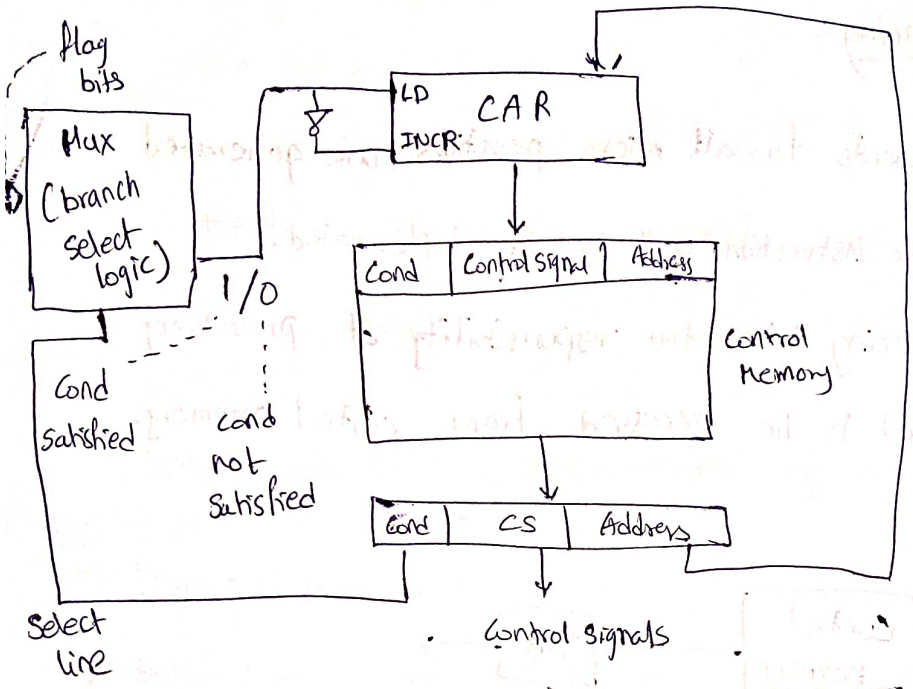
→ 1-Address control instruction is used in most designs



m – no of words in control memory

$n_1 + n_2 = n$

→ If condition is satisfied, next control word address given by address field. If condition is not satisfied, next CW address is obtained by increamenting CAR.



Q59 Horizontal microprogrammed CU has 256 instructions in ISA. Each instruction requires 16 MOPS. The system supports 16 Flag conditions and 32 control signals. The control word format is:

Cond	CS	Address

(i) find no of bit in control words

(ii) In byte organized CM how many bytes are needed for control memory

Sol:

Cond	CS	Address
4	32	12

no of control words = $256 \times 16 = 2^{12}$

∴ size of address = 12

In horizontal Mprogramming, CS is 32 bits

16 conditions ⇒ $\log_2 16$ bits = 4 bits

(i) no of bits = 48 ~~(ii) no of bytes = 6~~

(b) size of control memory

$= 4k \times 48$ bits

$= 4k \times 6$ bytes

$= 24$ kB

(Q60) Control signal field is divided in 6 groups.

If no of control signals per group is given below

G_1	G_2	G_3	G_4	G_5	G_6
14	15	7	6	5	6

what will be the min no of bits save for control field w.r.t.o horizontal Mprogramming.

Sol:

total no of control signals $= 14+15+7+6+5+6$

$= 53$ bits

$\therefore$ no of bits in horizontal $= 53$

no of bits in diagonal = | 4 | 4 | 3 | 3 | 3 | 3 |

$\therefore$ no of bits in diagonal $= 4+4+3+3+3+3$

$= 20$

$\therefore$ no of bits saved $= 53-20 = 33$.

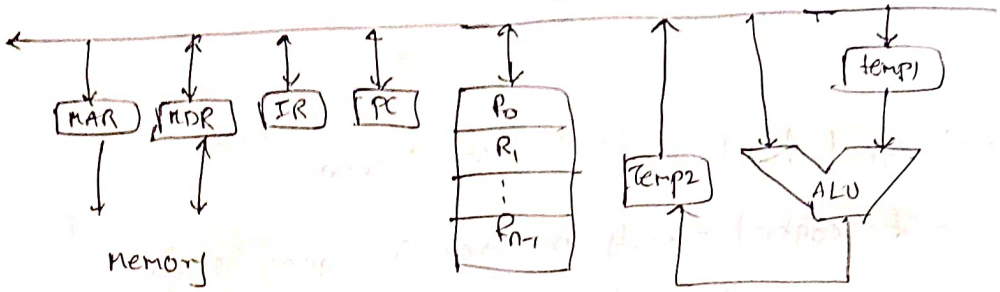
no of bits for vertical $= \log_2 53 = 6$

here no of bits saved $= 53-6 = 47$

06/09/20

(061) Find no of clock req for $R_1 \leftarrow R_1 + R_2$ i.e., ADD R_1, R_2.

The instruction has to be fetched from memory.



operand fetch into temporary ALU register temp1. Result after addition to be moved from Temp2 to R_1. Assume that any transfer using data bus takes one clock

Sol:

$MAR \leftarrow PC$ — Add doesn't require any clock

$MDR \leftarrow$ Memory

$IR \leftarrow MDR$

$temp1 \leftarrow R_2$ — Add doesn't require any clock

$temp2_{in}$, ADD

$R_1 \leftarrow temp2$

∴ 6 clocks

Nano Programmed Control Unit:

→ It is most flexible CU design and uses two level control memory

i.e., (i) Micro-Control Memory.

(ii) Nano-control memory.

→ Micro-control memory is responsible for control word sequencing.

→ Nanon-control memory is responsible for control signal generation.

→ It follows horizontal micro-programming but reduces effective control memory. This due to saving only distinguishable micro operation.

control words in nano control memory.

→ Because of 2-level control memory it is the slowest CU design.

→ It is used to store bootstrap programs or firmware of the application programs.

→ It provides same degree of parallelism as that of horizant programming.

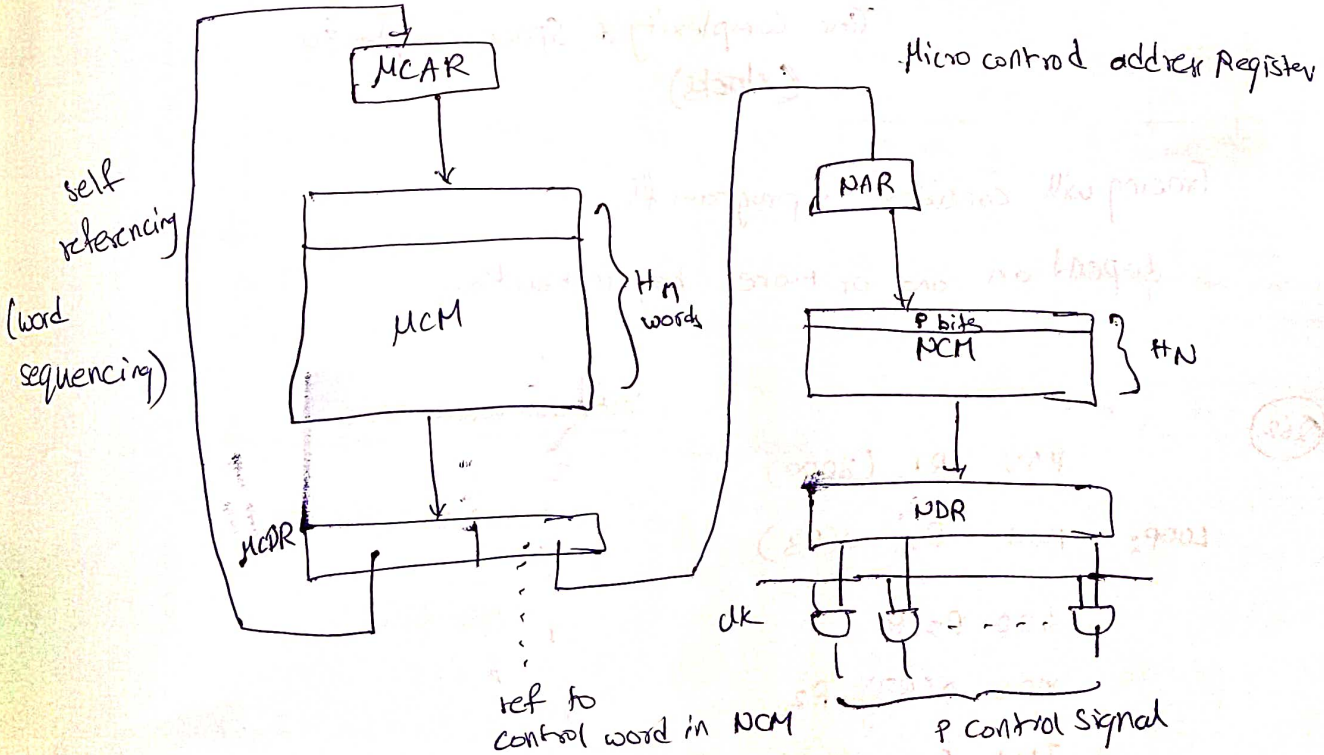
→ Effective control memory size = micro control memory size + nano control memory size

$= H_M (\log H_M + \log H_N) + H_N * P$

H_M : total words in micro control memory

H_N : total words in nano control memory

P: no of control signal to be generated.



→ Nano Control memory has distinguish control words and Micro control memory bag references to those

Characteristic	Control Unit Arrangement (in Descending order)
1. Speed	H/w, $H_{\mu p}$, $V_{\mu p}$, NPCU
2. Flexibility	NCPU, $V_{\mu p}$, $H_{\mu p}$, H/w
3. Control Memory Space	$H_{\mu p}$, Nano, $V_{\mu p}$, H/w

Assembly Program Tracing

→ Find contents of the register
→ " " " " specific memory locations
→ Find memory space arrangements

Time Complexity & Space Complexity
(Clocks)

Tracing will consider "program flow"
depend on one or more key instructions

Q62

		instruction word size
	MOV R1, (3000)	2
LOOP:	MOV R2, 0(R3)	1
	ADD R2, R1	1
	MOV 0(R3), R2	1
	INC R3	1
	DEC R1	1
	BNZ LOOP	2
	HALT	1

Memory location 3000 has value 10

$R_3 = 2000$

Memory contents from 2000 to 2010 are having value 100.

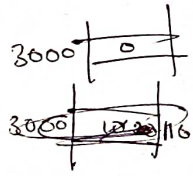
(i) How many memory accesses are done for data

a) 10 b) 11 c) 20 d) 21

Sol:

```
        MOV R1,(3000)  ------ 1
LOOP:   MOV R2,0(R3)   .... 1
        ADD R2,R1
        MOV 0(R3),R2   .... 1
        INC R3
        DEC R1
        BNZ loop
        HAL
```

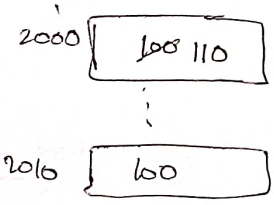
2 accesses within loop



R_1 9

R_3 2000 / 2001

loop: R_2 100 110



from value of R_1 we can say that the loop executes 10 times

$\therefore$ It 2#10 = 21

(ii) When machine halts what is the value in memory location 2010

a) 100 b) 101 c) 109 d) 110

when the program is about to halt the value in R_1 is 1

ADD R_2, R_1 ---- R_2 101

$R_2 = 100 + 1 = 101$

MOV $0(R_3), R_2$ 2009 101

But loc 2010 is not affected $\therefore$ opt @

(iii) what are the final values of R_1, R_2, R_3

R_1 --- 0

R_2 --- 101 (from (ii))

R_3 --- 2010

(iv) If the program is stored in word organized memory Instruction fetch takes 2clocks/word and execution for memory related instruction is 2 clocks and other one clock. How many clocks are req for this non pipelined implementation. Assume HALT instruct req only fetch.

Sol:

MOV R1, (3000) ------ 2*2+2 = 6 clocks

Loop: MOV R2, 0(R3) ------ 2+2 = 4

ADD R2, R~~3~~ ------- 1+1 = 3.

MOV 0(R3), R2 ------ 2+2 = 4.

INC R3 ---------- 2+1 = 3.

DEC R1 ---------- 2+1 = 3

BNZ loop ---------- 4+1 = 5

} 22*10 = 220

~~HARD~~

HALT ---------- 2

∴ 220 + 6 + 2 = 228 ~~do~~ clocks.

(v) If the above program is stored in byte organized memory from decimal address 1000 onward. If interrupt is occured during INC R3. what will be the return address (ie., the address ~~stored~~ saved in stack). Instruction word length ~~is~~ is 32 bits.

Sol:

```
MOV  R1,(3000) ---- 1000 to 1007
LOOP: MOV R2,0(R3) --- 1008 to 1011
      ADD R2,R1 ---- 1012 to 1015
      MOV 0(R3),R2 --- 1016 to 1019
      INC R3 ------ 1020 to 1023
      DEC R1 ------ (1024) to 1027
```

1word = 4 ~~bits~~ byte
mem is byte organized

∴ Interrupt is occured during INC R3, execution of INC R3 will be finished and starting address of DEC R1 will be saved.

(vi) If interrupt occurs during BNZ instruction, possible return addresses are

Sol:

$R_1 = 0$ ---- then 1008

$R_1 = 1$ ---- then 1038

(vii) If interrupt occurs during halt the return address is

a) 1036 b) 1039 c) 1040 d) 1037

Sol:

HALT instruction means same instruction is repeated again and again. i.e., in the implementation PC ~~reg~~ never gets incoeamented.

∴ Beginning address of HALT is return address

i.e., 1036

Register Allocation

→ If variables are maintained in CPU registers it takes less time to acess & process data.

→ However the no of registers are limited. Hence allocation plays a significant role.

→ Register allocation policy minimizes the no of spills. A spill is required when a live variable is stored into memory

→ two allocation policies
- Local Register allocation
- Global register allocation

Local Register allocation:

Belady's algorithm

→ Place all the registers in free list as and when it is allocated place that in allocated list

(i) A variable request for register allocation consult free list, if free register is available allocate it. Also map the register to allocated list

(ii) If no free register is available (i.e., free list is empty) then identify that allocated register which containing 'Dead' variable. Release it and allocate to the variable.

Dead variable: variable which is no longer req.

(iii) If no dead variable is found then identify the allocated register whose variable is required after longest time. Spill it & allocate it.

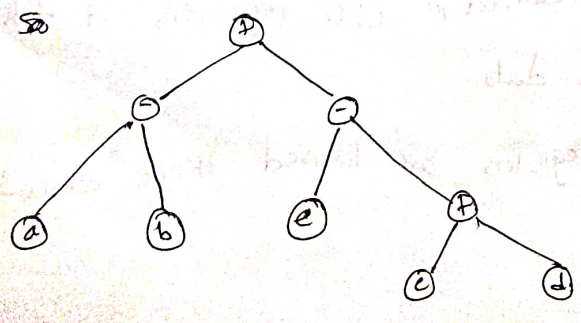
Spill: saving to memory.

(Q63) Load & Store instruction are only supported memory operations.

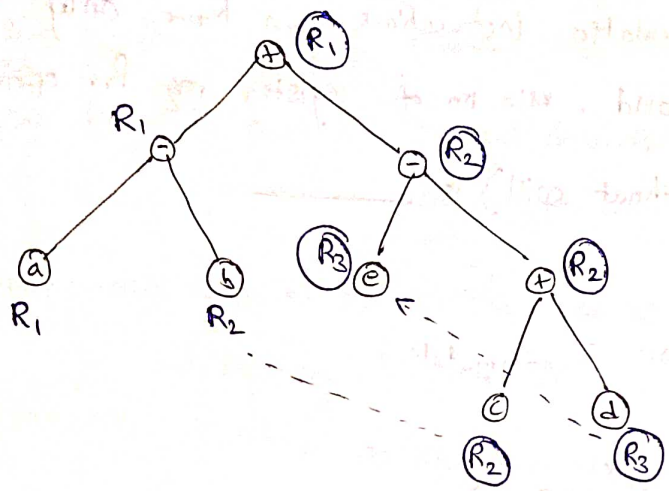
Arithmetic operation are performed only on registers operands.

If no spill is allowed min no of register req is ____

So



Method 1:



∴ 3 registers.

Method 2:

$R_1 = a$

$R_2 = b$

$R_1 = R_1 - R_2$

$R_2 = c$

$R_3 = d$

$R_2 = R_2 + R_3$

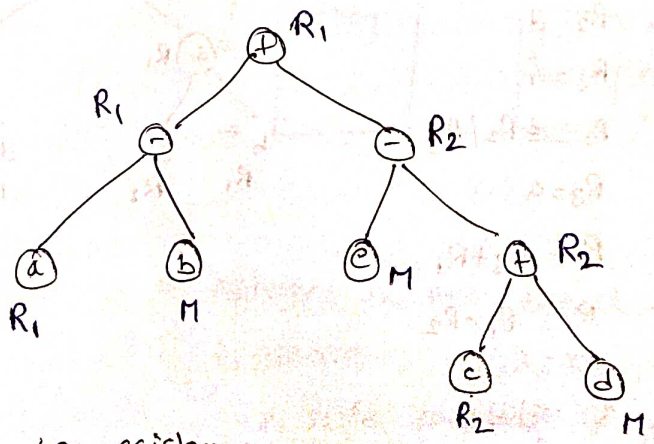
$R_3 = e$

$R_2 = R_3 - R_2$

$R_1 = R_1 + R_2$

(ii) Now for the same question, if one memory operand is allowed then min no of register req is:

Sol:



∴ 2 registers

Q64 Consider expression $(a-1)*((b+c)/3+d)$ for a load-store architecture, arithmetic instructions can have only register or immediate operand. Min no of register req for optimal code generation (without spill) is _____

Sol:

~~for~~ Multiplication is commutative

$((b+c)/3+d)*(a-1)$

$R_1 = b$
$R_2 = c$
$R_1 = R_1 + R_2$
$R_1 = R_1/3$
$R_2 = d$
$R_1 = R_1 + R_2$

$R_2 = a$
$R_2 = R_2 - 1$

$R_1 = R_1 * R_2$

Q65 $x = a + b*c - d/f$

No spill allowed

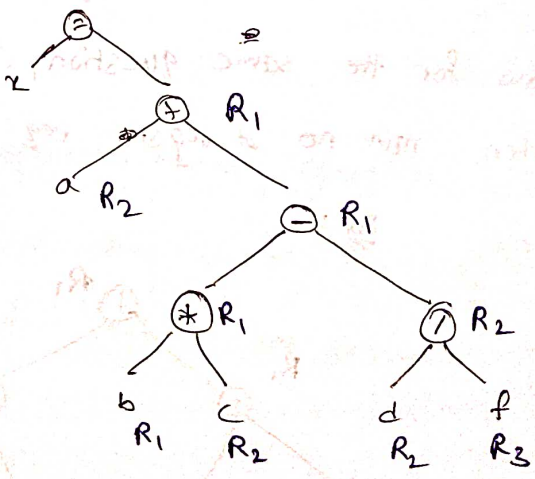
Operands must be registers.

Find min no of registers req.

Sol:

$R_1 = b$
$R_2 = c$
$R_1 = b*c$
$R_2 = d$
$R_3 = f$
$R_2 = R_2/R_3$
$R_3 = a$
$R_1 = R_3 + R_1$
$R_1 = R_1 - R_2$
$x = R_1$

∴ 3 registers



IO Organization

→ CPU has to interact with external devices using IO ports.

→ CPU can interact with external devices only through address.
i.e., address assigned to devices.

→ Assigning addresses to IO devices can be done in two ways

(i) Memory Mapped IO

→ address space is partitioned into memory space & IO space

→ adv:

* all addressing modes can be extended to I/O
* address are distinct. So just from the address we can say whether it is IO address or memory address.

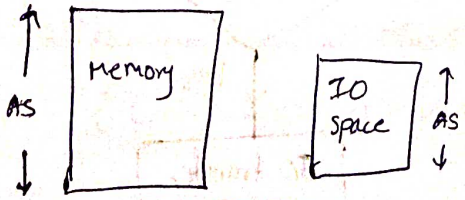
∴ There is no need of address separation
i.e., no need of IO/$\overline{M}$

Disadv:

* less IO expandability.

(change in IOS)

(ii) IO Mapped IO (isolated IO)



Disadv:

* Addresses are not distinct. So address separation is needed.
i.e., IO/$\overline{M}$ control bit is needed.

* Limited IO address & IO addressing modes
∴ limited flexibility

In the case of RISC only direct addressing mode is possible.

Adv:
→ More Expansion of IO.

→ How to move the data from/to IO devices?

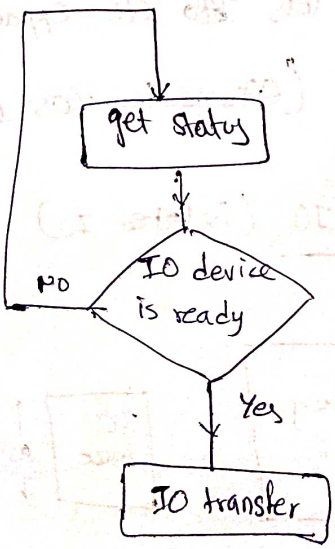
3 ways:

- Program Controlled Transfer (Program Driven)
- Interrupt Driven Transfer
- DMA.

(1) Program Driven Data transfer

→ Simple & useful for all devices.

→ It will get the status device & check whether device is ready or not. This checking goes on until the device is ready.



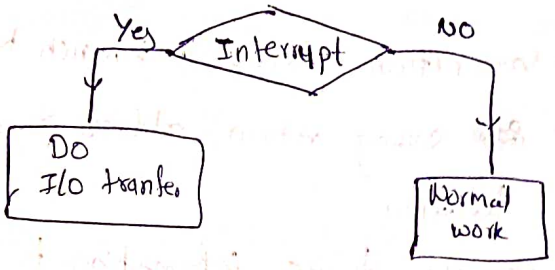
→ Program instructions are used to transfer a word.

Disadv: Most of the time of CPU is spent in waiting

iii) Interrupt Driven Transfer:

→ Whenever device is ready, it will interrupt processor (i.e., Device controlled data transfer)

→ So here CPU will not be in the loop



→ Processor is most efficiently used in interrupt driven (CPU) data transfer.

Drawback:

More complexity in interrupt implementation.

Interrupt implementation issues:

i) It needs to address the issues associated with processor

ii) " " " " " " " devices

w.r.to devices

→ when it should interrupt?

Asynchronous

i.e., it can interrupt any time.

→ How to interrupt?

By changing signal at H/w pin

(level trigger (or) edge trigger)

w.r.to processor

→ when to recognize

Complete the execution of current instruction & recognize.

→ How to recognize?

* whenever interrupt occurs interrup flag is set.
* At the end of every instruction the flag is check.

→ what to do after recognizing?

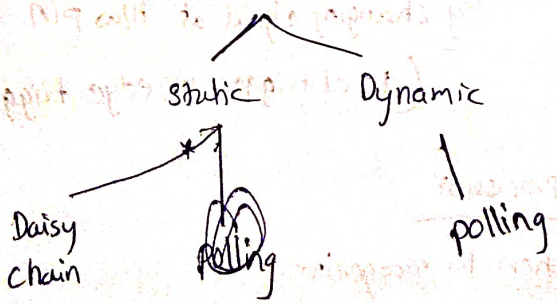
* save return address & branch to ISR
* saving return address is done at a default location
* Use the device information to know the location of the ISR. Such devices which tell the location of the interrupt are called vector interrupt device.
* Second way to find the location of ISR is using a default address.
  Such devices are called Scalar Interrupt Device.

→ How to resolve if more the one interrupt occurs?

uses priorities & serve highest priority interrupt.

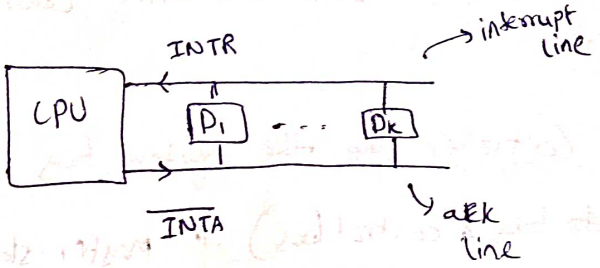
→ who will assign the priority? and what kind?

* OS may assign them.
* priorities



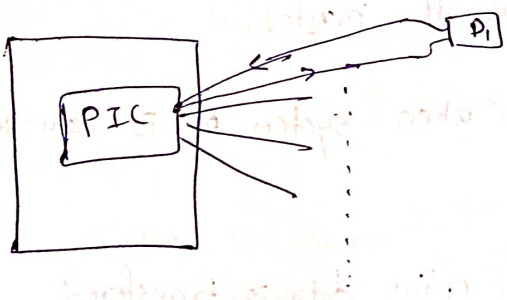
Daisy chain :

highest priority devices are connected closer to the processor.



* Here the position decides the priority.
* Starvation

Polling :



PIC : Priority Interrupt Controller.

→ Every Device has its own INTR & $\overline{INTA}$ line

→ Services are done in a round robin fashion.

∴ no starvation

→ Now when one of multiple interrupts are served what about other request?

Pended mode :

→ Save in interrupt queue & serve

Non-pended

→ other are Rejected.

(ii) Direct Memory Access (DMA)

→ Here processor is isolated & DMA controller coordinates data transfer

→ Processor & DMA Controller use the system bus (Add bus, data bus & control bus) in master-slave mode

→ During DMA controller is Master & and during normal operation processor is the master.

→ DMA mode is based on when system bus is returned to CPU

Burst mode

→ return after entire data is transfered
~~& prefered ft~~

Cycle stealing mode

→ After transfering each word.
* prefered for lower IO burst.

Block transfer mode

→ After a block transfer.

→ Count register specifies no of words of the block to be transfer.

If it is 8 bit then, max possible for transfer is $2^8 = 256$ words.

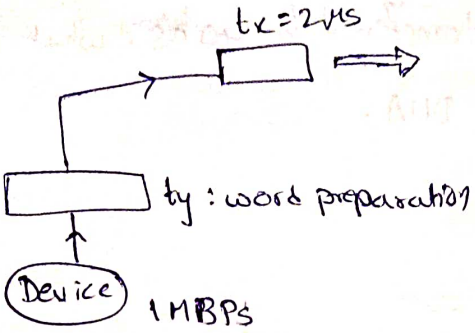
→ DMA deprives ~~copy~~ the system bus availability to CPU, hence it will be Blocked due to DMA.

→ In cycle stealing mode

$$\% \text{ time processor block} = \frac{t_x}{t_x + t_y} * 100$$

t_x: word transfer time t_y: word preparation time.

during word preparation CPU can use the system time but not ~~during~~ during word transfer time.



Hex if word is 8 bytes then

t_y (word prep) = 8 * 1 = 8 μs

t_x = 2 μs

% time CPU blocked = $\frac{2}{10} * 100 = 20\%$.

~~during DMA~~

→ In ~~a~~ burst mode the system bus is blocked entirely for DMA.

% time of CPU contribution : $\frac{t_x}{t_x + t_y} * 100$

t_x : CPU time for initialization & termination of DMA

t_y : Data transfer time by DMA

→ Acknowledgement of interrupt is done at the end of each suboperation of the instruction.

In case DMA there is no need to save address and all ~~like last~~ unlike the case with ~~int~~ interrupt.

(Q66) Consider a disk with data transfer rate 50 MBPS. It is operated with cycle stealing mode of DMA. Here wherever 64 bit information is available it transfered in 40 ns. what is the % time CPU blocked due to DMA.

Sol:

~~50 MBPS~~ ---.

50MB ——— 1S

1B ——— $\frac{1}{50\times10^6}$ = $\frac{1000}{50\times10^9}$ = 20ns

64 bit = 8 bytes - - - - 160 ns

∴ % time blocked = $\frac{40}{40+160}$ = 20%

(Q67) In (Q66) if DMA controller is having 14 bit count register and it is required to transfer 64 MB. minimum no of times system bus exchanged b/w DMA controller and CPU is ———

Sol:

Max no of bytes that can be transfered at once

= 2^{14} bytes (for min, we need to transfer 2^{14} bytes)

Total transfer req = 2^{26} bytes

~~For every transfer the bus is exchanged twice~~

no of transfers = $\frac{2^{26}}{2^{14}}$ = 2^{12} = 4096

∴ no of exchanges = 4096

Q68 A processor is operating with 50MHz clock. The DMA requires 1000 clocks for initialization & termination. The disk with data transfer rate 1MBPS is used to transfer 2kB file using DMA. What is % of CPU time used in this transfer.

Sol:

This is burst mode

50MHz

$\Rightarrow$ time period = $\frac{1}{50 \times 10^6} = 20\,ns$

$\therefore$ 1000 clocks --- $2 \times 10^4\,ns$

$= 20\,\mu s$ --- CPU is used

1MB ---- 1S

1kB ---- 1ms

2kB: ---- 2ms --- CPU is blocked

$\therefore$ % time CPU used = $\frac{20\mu s}{20\mu s + 2ms}$

$= \frac{20}{2020} = \frac{1}{101} \cong 0.96\%$

Q69 Let variables A, B, C, D denote CPU temperature high, non-maskable interrupt, high priority and maskable words. which is the correct relation.

a) A → BC b) A → CD c) A → B D d) A → BCD

Sol:

CPU high temparature issue has high priority.

and thus it will be non-maskable

$\therefore$ A → BC

Disk Memory

* Semi Random access memory
* information is recorded interms of tracks & sectors.
* unit of transfer in disk is sector.
* Based on the recording density the disk construction can be
  (i) Constant track capacity (default consideration)
  (ii) variable track capacity
* In constant track capacity variable recording density is used (max for innermost track and min for outermost track)

  Here disk will have different angular velocity for different tracks. This changing of velocity is called angular velocity modulation. (angular velocity is more for outer tracks and less for inner tracks)

  However, track coverage time is same.

* In variable track capacity
  - → inner tracks will have lesser data
  - → recording density is constant across every track.
  - → Angular velocity is constant

    so it takes less time for covering inner tracks.

* To perform read/write on disk
  (i) Read/write head is moved onto the concerned track. (seek time)

(ii) Disk is rotated so that concerned sector is under read/write head. (rotational latency)

(iii) perform data transfer.

∴ total time for disk access = seek time + rotational latency + data transfer time.

→ In general seek time > rotational latency > transfer time

→ the capacity of disk

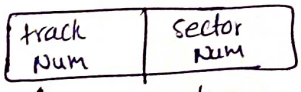
$$C = T * S * B$$

T: no of track

S: sectors / track

B: bytes / sector.

→ Disk Addressing (sector is addressed)



→ If format overhead per sector is k bytes,

the disk capacity = $C - (T * S * k)$

→ Data transfer rate:

No of bytes transfer per unit time.

In one rotation disk covers one track

∴ in one rotation time it covers one track capacity.

→ Taking DMA into consideration

for word preparation time we consider the ~~data~~ time from transfer rate (calculated through rotation speed)

A disk in which both sides data is stored is called platter

Disk System

→ It is collection of disk stacked on one another. Information is distributed interms of cylinders.

→ cylinder is logical collection of similar position tracks.

→ Thus in one rotation entire cyclinder is covered

→ sectors are addressed cylinder wise. Each sector has a logical address and linear address.

logical address $\langle C, H, S \rangle$

C - cylinder No.

H - Surface No.

S - sector No..

Cylinders are No number from inward to outward

→ Each disk has 2 surfaces.

→ Relation blw linear & logical address

~~Last~~ logical add (C, H, S)

linear addres $L = C * m + H * n + S$

m = no of sector per cyclinder

n = no of sectors per track

If there are p surfaces then

$m = p * n$

→ ~~Logical~~ linear to logical.

Linear address = l

m) l (C --- cylinder number

n) q_1 (H --- Surface number

$q_2 = S$ --- sector number

→ capacity of disk system → no of surfaces

$= P * T * S * B$

(or)

$= K * M * B$

K → no of cylinders

M → sectors/cycliner

B → bytes/cylinder

→ no of bits for addressing disk system

Surface NO	track No	Sector NO
$\log_2 P$	$\log_2 T$	$\log_2 S$

(or)

Cylinder NO	Sector Num
$\log_2 K$	$\log_2 M$

--- sector no among entire cylinder

(GATE) Hard disk

16 surfaces

16384 cylinders,

64 sectors / track

Sector capacity is 512 bytes.

Data is organised cylinder wise. address format <C, H, S>

A file of size 42797 KB is stored from <1200, 9, 40>

what is the cylinder of last sector of file.

So a) 1281 b) 1282 c) 1283 d) 1284

Sol:

Assume file is store from 0th location

now calculate last sector

sectors/cylinder
= 16 * 64
= 1024

file size = 42797 KB

sector size = 512 bytes

∴ file size = 42797 * 2 = 85594 bytes

sectors/cylinder

1024) 85594 (83 --- cylinder number

sectors/track ---- 64) 602 (9 ---- surface number

26 ---- sector number

<83, 9, 26>

now we add <83, 9, 26> to <1200, 9, 40>

1200	9	40
83	9	26
1284	3	2

40 + 26 = 66
= 64 + 2

9 + 9 + 1 = 19 = 16 + 3

~~Method 2:~~

~~Calculate linear add of starting add~~

~~add no of sectors to it.~~

~~Now convert it to logical~~

~~linear add of starting add = 1200 * 1024 + 9 * 64 + 40~~

=

Number Representation

Numbers represented can be

1. Fixed Point --- radix position is fixed
   ∴ position need not be store
2. Floating Point ... radix position may change
   ∴ position has to be stored.

They may be signed or unsigned

→ for n-bit ⊛ number, it will have 2^n combinations.

These combinations have to be interpreted as per the number.

→ Fixed point numbers can be integers or fractions

(10001.) (.00101)

→ For n-bit fixed point unsigned number range is 0 to 2^{n-1}

The decimal value = $\sum b_i * 2^i$

→ For n-bit fixed point unsigned fraction range is

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

decimal value = $\sum b_i * 2^{-i}$

~~Eg: find max value of 8 bit fraction~~

Consider .11111111

$V = 1/2 + 1/2^2 + \dots + 1/2^8 \approx \frac{\frac{1}{2}}{1 - \frac{1}{2}} = 1$

If fraction is having infinite bites

max value = 1

Max value of 8 bit fraction

$$= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^8}$$

$$x = 1 - \left(\frac{1}{2^9} + \frac{1}{2^{10}} + \cdots\right)$$

$$= 1 - \frac{1}{2^8}\left(\frac{1}{2} + \frac{1}{2^2} + \cdots\right)$$

$$= 1 - \frac{1}{2^8}$$

$$= 1 - 2^{-8}$$

∴ Max value of n-bit fraction = $1 - 2^{-n}$

Here this 2^{-8} (2^{-n}) is called resolution

i.e., minimum change required to be recognized by the system.

The resolution is equal to the weight of least significant

→ resolution ↓ ⇒ accuracy ↑

Eg: A 12 bit fraction has less resolution than 8 bit fraction.

(Q81) A real number (unsigned) has i bits for integer and j bits for fractional field. what is its range.

Sol:

0 to $(2^i - 1) + (1 - 2^{-j})$

0 to $2^i - 2^{-j}$

Signed Numbers

(i) Signed - Magnitude

(ii) 1's complement

(iii) 2's Complement

Common things:

→ All representations use most significant bit to denote sign.

0 ..... +ve

1 ..... -ve

→ for all +ve number all representation have same pattern and same value.

Variations

→ There are two patterns for zero in Sign Mag. & 1's comp.

Sign Mag. & 1's comp have same range whereas 2's comp have only one pattern for '0' and hence it covers one extra -veg number in range.

→ range:

$-(2^{n-1}-1)$ to $(2^{n-1}-1)$ ..... Sign Mag & 1's Comp

(-2^{n-1}) to $(2^{n-1}-1)$ ..... 2's comp

→ Sign Mag & 2's comp representations follow weighted system whereas 1's comp is non-weighed code

→ In sign-mag sign bit doesn't have any weight but in 2's comp sign bit has -ve weight.

(082) An 8-bit binary pattern have has value of $(-47)_{10}$ in sign Magn. what its value if interpreted in 2's comp.

Sol :

$(-47)_{10}$

$(47)_{10} = (101111)_2$

$(-47)_{10} = 10101111$

10101111 in 2's comp

$-128 + 47 = -81$

Sign Extension :

→ It allows storing smaller number in bigger register.

→ Extended bits do not change either sign or value

→ In 2's comp & 1's comp sign bits are copied in extended positions.

07/09/20

Floating Point Representation

→ A floating point number contains integer part, fractional part and radix point has to be stored.

→ Most of the systems use mantissa-exponent combination to represent floating number

→ mantissa cover both integer portion and fractional portion and represented as normalized sign-magnitude fraction

(fraction begins with '1' in binary) (in other system beginning with non-zero)

→ Exponent represent relative position of radix point.

→ Most of the systems use biased exponent for denoting normal signed exponent (true exponent)

E = e + bias

01/10/20

Number System:

- Binary numbers
  - unsigned
  - signed numbers
    - Sign magnitude representation
    - 1's complement representation
    - 2's complement representation.

bit pattern	unsigned	SM	1's	2's
0 0 0	0	0	0	0
0 0 1	1	1	1	1
0 1 0	2	2	2	2
0 1 1	3	3	3	3
1 0 0	4	-0	-3	-4
1 0 1	5	-1	-2	-3
1 1 0	6	-2	-1	-2
1 1 1	7	-3	-0	-1

→ Find min no of bits req to represent $(32)_{10}$ in 2's comp form.

Sol:

If n is no of bits the max value represented is

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

$$\therefore 2^{n-1}-1 \geq 32$$

$$2^{n-1} \geq 33$$

$$n-1 \geq 5.\ldots \Rightarrow n \geq 6.\ldots$$

$$\Rightarrow n = \lceil 6.\ldots \rceil = 7$$

→ Find min no of bits req to represent $(-32)_{10}$ in 2's comp form.

Sol:

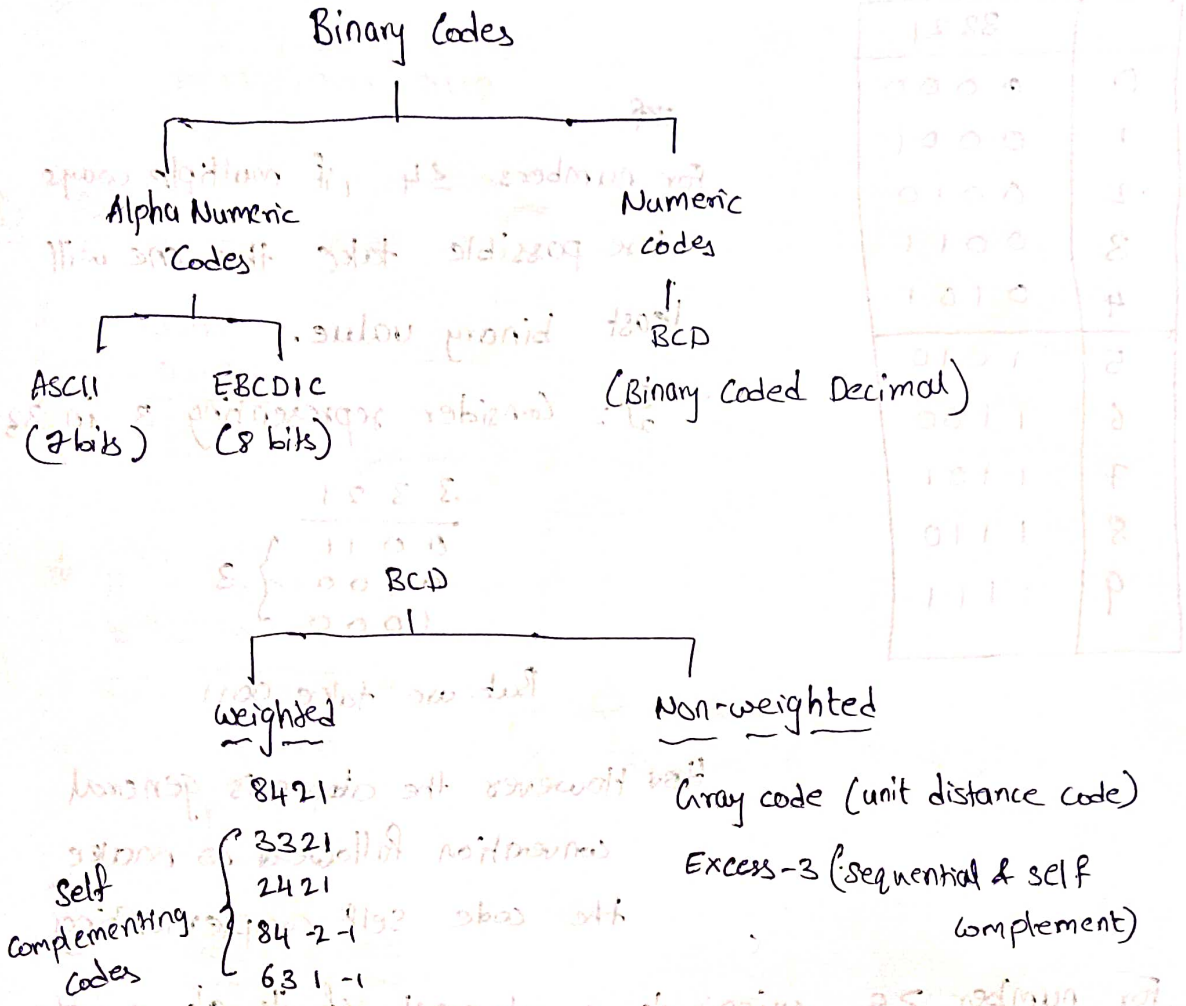
min number possible is -2^{n-1}

$$\therefore -2^{n-1} \leq -32 \Rightarrow 2^{n-1} \geq +32$$

$$n-1 \geq 5$$

$$n \geq 6 \therefore \underline{6 \text{ bits}}$$

02/02/20



Self Complementing Code:

In self complementing code, the code of a digit and the code of 9's complement of digit are 1's complement to each other.

~~Weighted~~

Excess-3:

	8421	Ex's-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Ex's-3 is sequential
i.e., adding '1' gives next number.

∴ Ex's-3 is both sequential & self complementing

3321:

	3321
0	0000
1	0001
2	0010
3	0011
4	0101
5	1010
6	1100
7	1101
8	1110
9	1111

For numbers ≤4, if multiple ways are possible take the one will least binary value.

Eg: Consider representing 3 in 3321

3	3	2	1	
0	0	1	1	} 3
0	1	0	0	
1	0	0	0	

But we take 0011

However the above is general convention followed to make the code self complementing

For number ≥5, using 1's complement of its 9's complement

Note:

In case of weighted codes,

for every self complementing code, sum of weights = 9

However reverse need not to be true.

→ 3321 is self complementing but not sequential.

Eg: Represent $(863)_{10}$ in

(i) BCD (ii) 3321 (iii) Ex's-3

(i) BCD

1000 0110 0011

(ii) 3321

1110 1100 0011

(when asked BCD, it generally refers to 8421.)

(iii) Ex's-3

1011 1001 0110

BCD Addition

→ 3 - 0011
6 - 0110
9 - 1001

→ 3 - 0011
7 - 0111
10 - 1010 → Invalid pattern
0110 so add 6
10000

0001 0000
1 0 i.e., 10

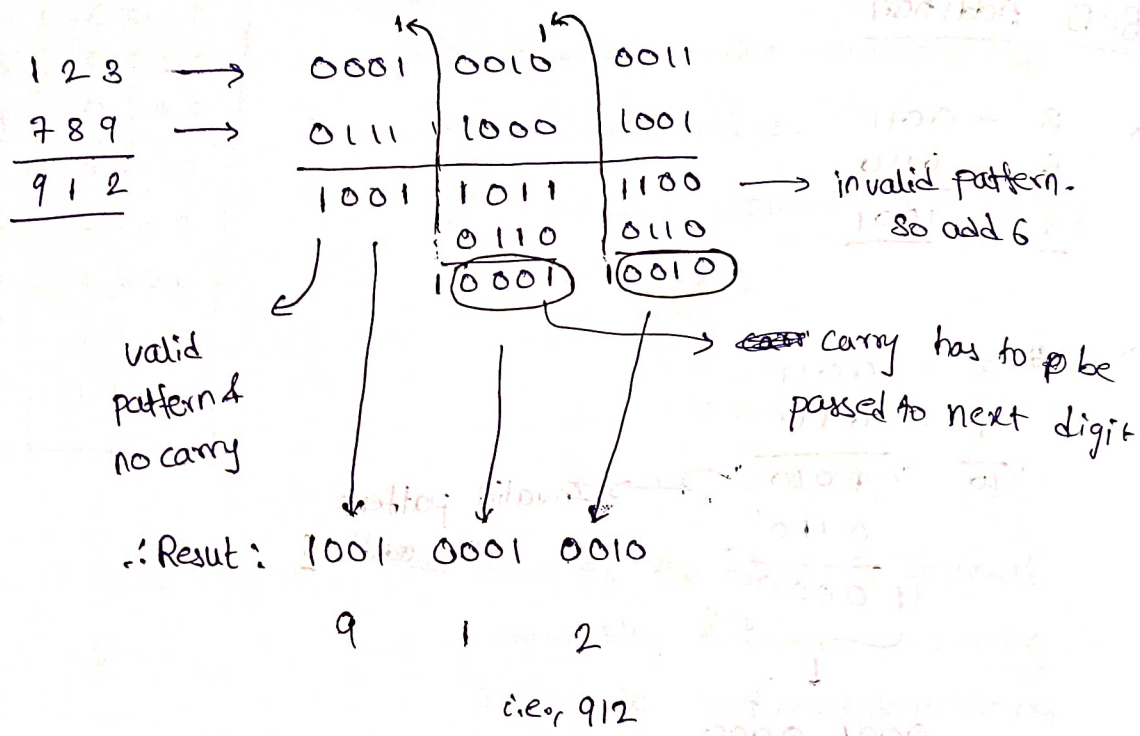
→ 9 - 1001
9 - 1001
10010 → valid pattern
0001 0010
1 2
i.e., 12

Here pattern is valid but still result is wrong
~~The reason~~ It is because of Carry.
So we add 6 again

10010
00110 ← add 6
11000
0001 1000
i.e., 18

→ So we add 6, if invalid pattern occurs or if there is a carry. i.e., the correction used in BCD addition is 6.

Eg:



Excess-3 addition:

→ If carry occurs → correction is add +3.

→ If there is no carry → add (-3)

→ while performing this addition make sure that no of digits in i/p is same as no of digits in o/p

Eg: 7+3=10

so add 07
03
10

Reason:

no carry:

x+3
y+3
x+y+6
-3
x+y+3

Eg: 7 - 0 1 1 0
2 - 0 1 0 1
9 1 1 1 1 → no carry
0 0 1 1 so subtract 3.
1 1 0 0
↳ 9 in ex3-3.

Eg: 07 — 0011 1010
03 — 0011 0110
10 — 0111 0000

carry occured
So add +3

(-) 0011 (+) 0011
0100 0011

no carry
So subtract 3

∴ Result: 0100 0011

Gray Code

- Reflexive
- Unit Distance code
- Cyclic Code
- Non-weight code

1-bit
0
1

2-bits
0 0
0 1
1 1
1 0

give upper half MSB as 0
lower half MSB as 1

3-bits
0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

for 3-bits write all 2-bit codes in upper half and use reflection for lower half
MSB of upper — 0
MSB of lower — 1

Silly this procedure can be extended to any no of bits.

→ Hence gray code is called Reflexive.

→ Every successive numbers differ exactly in one bit position.
So gray code is unit distance code.

→ Also the distance blw last & first number is also 1.
Therefore gray code is called cyclic.

Conversions b/w binary & gray codes:

Binary to gray:

Eg: $(12)_{10} - (1100)_2$ ... (i)

1 0 1 0 → 12 in ~~binary~~ gray code

Let $b_4\ b_3\ b_2\ b_1$ be binary code

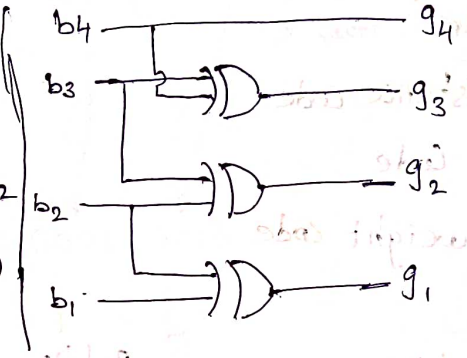
Let $g_4\ g_3\ g_2\ g_1$ be corresponding gray code

then $g_4 = b_4$

$g_3 = b_4 \oplus b_3$

$g_2 = b_3 \oplus b_2$

$g_1 = b_2 \oplus b_1$



Eg:

1 1 1 1 — 15

1 0 0 0 — 15 in gray

Eg: 1 1 0 0 1 0 0 0 — 200 in binary

↓

1 0 1 0 1 1 0 0 — 200 in gray

Gray to binary:

Eg:

$g_4\ g_3\ g_2\ g_1$

1 0 1 0 → 12 in gray

1 1 0 0 → 12 in binary

$b_4\ b_3\ b_2\ b_1$

In binary to gray

$g_3 = b_4 \oplus b_3$

$\Rightarrow b_3 = g_3 \oplus b_4$

$b_4 = g_4$

$b_3 = b_4 \oplus g_3$

$b_2 = b_3 \oplus g_2$

$b_1 = b_2 \oplus g_1$

$b_k = b_{k+1} \oplus g_k$

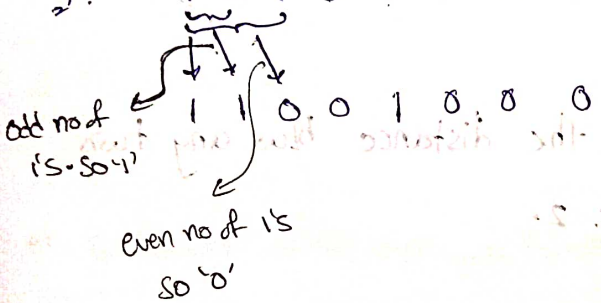
$$b_4 = g_4$$

$$b_3 = b_4 \oplus g_3$$
$$= g_4 \oplus g_3$$

$$b_2 = b_3 \oplus g_2$$
$$= g_4 \oplus g_3 \oplus g_2$$

$$b_1 = b_2 \oplus g_1$$
$$= g_4 \oplus g_3 \oplus g_2 \oplus g_1$$

Eg: 1 0 1 0 1 1 0 0



While converting gray to binary

b_k is XOR of the gray code bits before g_k (including g_k)

So just put '1' if no of 1's = odd
'0' if no of 1's = even

Note:

$$g_n = b_{n+1} \oplus b_n$$

$$b_n = b_{n+1} \oplus g_n$$
(or)

$$b_n = g_n \oplus g_{n+1} \oplus g_{n+2} \oplus \cdots \oplus g_k$$ → Most significant bit.

In binary to gray
$g_3 = b_4 \oplus b_3$
$\Rightarrow b_3 = g_3 \oplus b_4$

Error Handling

- Error Detection
- Error Correction

→ If error changes pattern from one valid pattern to other valid pattern, then we cannot detect the error. (if we are not using any extra information)

→ If error changes a pattern from valid to invalid then error can be detected.

Error Detection: (~~Bit error~~)

→ To detect ~~error~~ 1-bit errors, the distance blw any two valid codes should be atleast 2.

For example

0001
0001
1001
1100

} These patterns cannot be used to detect one bit error detection. because distance blw 0001 & 1001 is 1.

Eg:

0000
0011
1010
1001

} Here distance blw any two valid codes is 2.
∴ This pattern ⑥ can be used for detecting 1-bit errors.

→ Similarly to detect 2-bit errors, the distance blw any two valid patterns has to be atleast 3.

→ similarly to detect ~~n~~ t-bit errors, the distance blw any valid codes has to be atleast t+1.

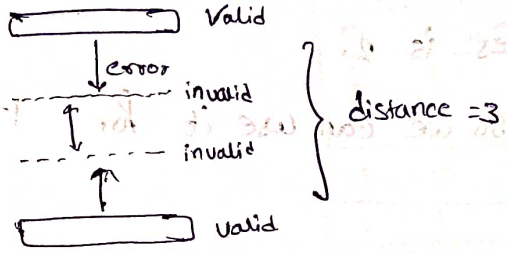
→ In a code, d_{min} distance blw any to valid patterns is called Hamming distance.

→ So to detect t-bit errors,
Hamming distance has to t+1.

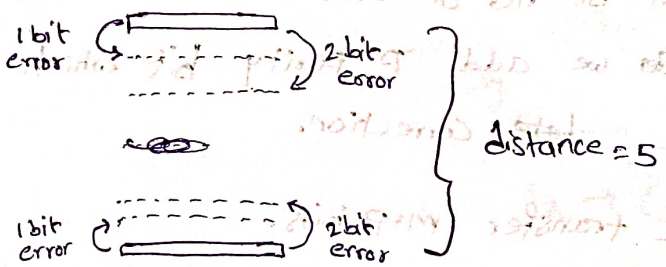
03/10/20

Error Correction:

→ To correct 1-bit error, the minimum distance b/w any two valid codes has to be 3



→ To correct 2-bit error, the minimum distance b/w any two valid codes has to be 5.



→ Sly to correct n-bit error, the minimum distance b/w any two valid codes has to be 2n+1.

→ We increase distance b/w valid codes by increasing no of bits.

For example consider gray code.

3-bit gray code	P ---- Parity bit (even parity)
0 0 0	0
0 0 1	1
0 1 1	0
0 1 0	1
1 1 0	0
1 1 1	1
1 0 1	0
1 0 0	1

even parity: bit is added such that no of 1's is even

By adding this parity bit the minimum distance b/w any two codes is 2.

So we can use it for 1-bit error detection.

Hamming Code:

→ used for 1-bit Error correction.

→ Let m be no of bits to be transfed.

we add 'p' parity bit which are used in error correction.

→ So we transfer m+p bits.

Out of this any one bit can get corrupted or there could be no error.

∴ m+p bits can get transfered in m+p+1 way

'p' bits has to handle all these cases

$$\therefore \boxed{2^p \geq m+p+1}$$

Eg: Calculate min no of parity bits required to transfer 4 bits.

Sol:

$m=4$, p

$\Rightarrow 2^p \geq m+p+1 \Rightarrow 2^p \geq p+5$

$\Rightarrow \boxed{p=3}$

Working:

Let us consider we need to transfer 0101

i.e., $m = 4 \Rightarrow P = 3$

Let us consider even parity

total bits to be tr

m_1	m_2	m_3	m_4
0	1	0	1

1	2	3	4	5	6	7
P_1	P_2	m_1	P_4	m_2	m_3	m_4
0	1	0	0	1	0	1

$P_1 \rightarrow P_1\ 3\ 5\ 7$

$0\ 1\ 1$

$\therefore P_1 = 0$ (even parity)

$P_2 \rightarrow P_2\ 3\ 6\ 7$

$0\ 0\ 1$

$\therefore P_2 = 1$

$P_4 \rightarrow P_4\ 5\ 6\ 7$

$1\ 0\ 1$

$\therefore P_4 = 0$

$\therefore$ Data transfered is 0100101

Assume that bit at position 5 is corruped

i.e., Data recieved is 0100001

1	2	3	4	5	6	7
0	1	0	0	0	0	1

Now reciever calculate C_1, C_2, C_3

$C_1 \rightarrow (4,5,6,7)$ has to be of even parity

$(0\ 0\ 0\ 1)$

It has odd parity

$\Rightarrow$ one of 4,5,6,7 has been corrupted

$\therefore C_1 = 1$ (:: error)

check bits

C_1 (P_3)	C_2 (P_2)	C_3 (P_1)	error position
0	0	0	no error
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

from the table,

P_1 makes sure bits positions (1,3,5,7) has even parity

Sily

$P_2 \rightarrow (2,3,6,7)$

$P_3 \rightarrow (4,5,6,7)$

$C_2 \rightarrow (2, 3, 6, 7)$

1 0 0 1

↳ even parity

∴ no error

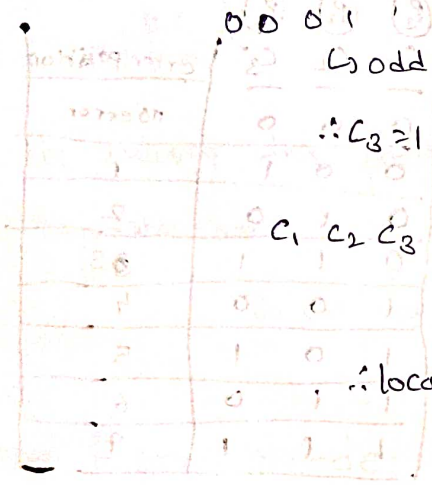
$C_2 = 0$

$C_3 \rightarrow (1, 3, 5, 7)$

0 0 0 1

↳ odd parity

$\therefore C_3 = 1$



$C_1\ C_2\ C_3 = 101$

i.e., 5

∴ location 5 is error.

Eg: let d.M = 1100

1	2	3	4	5	6	7
P_1	P_2	m_1	P_3	m_2	m_3	m_4
0	1	1	1	1	0	0

Data transfered

let data recieved: 0 1 0 1 1 0 0

1 2 3 4 5 6 7

$P_1 \rightarrow (0\ 0\ 0)$ ✗ $C_3 = 1$

$P_2 \rightarrow (1\ 0\ 0\ 0)$ ✓ $C_2 = 1$

$P_3 \rightarrow (1\ 1\ 0\ 0)$ ✓ $C_3 = 0$

$\therefore C_1\ C_2\ C_3$

0 1 1

i.e., error at 3rd position

→ Assume we need to transfer 5 bits

⇒ m = 5

$2^p \geq m + p + 1$

$2^p \geq 6$

⇒ p = 4 ∴ no of bits transfered = m + p = 9

p = 4 gives 16 combinations

However since only 9 bits are transfered,

they can get transfered in 16 ways.

So only 16 combinations are used i.e., 0 to 9.

(P_4)	(P_3)	(P_2)	(P_1)	
C_1	C_2	C_3	C_4	
0	0	0	0	— 0
0	0	0	1	— 1
0	0	1	0	— 2
0	0	1	1	— 3
0	1	0	0	— 4
0	1	0	1	— 5
0	1	1	0	— 6
0	1	1	1	— 7
1	0	0	0	— 8
1	0	0	1	— 9

∴ $C_4 \rightarrow (1, 3, 5, 7, 9)$ ↓ P_1

∴ $C_3 \rightarrow (2, 3, 6, 7)$ ↓ P_2

$C_2 \rightarrow (4, 5, 6, 7)$ ↓ P_3

$C_1 \rightarrow (8, 9)$ ↓ P_4

Let M = 11011

1	2	3	4	5	6	7	8	9
P_1	P_2	1	P_3	1	0	1	P_4	1
0	0	1	0	1	0	1	1	0

$P_1 \rightarrow$ 1 3 5 7 9 ⇒ $P_1 = 0$

P_1 1 1 1 1

$P_2 \rightarrow$ 2 3 6 7

P_2 1 0 1 ⇒ $P_2 = 0$

$P_3 \rightarrow$ 4 5 6 7 ⇒ $P_3 = 0$

P_3 1 0 1

3rd position

∴ Data transfered: 00101011

let datar received: 001010011

C_1 - (8 9) - (1 1) ✓ ⇒ $C_1 = 0$

C_2 - (4 5 6 7) - (0 1 0 0) ⇒ $C_2 = 1$

C_3 - (2 3 6 7) - (0 1 0 0) ⇒ $C_3 = 1$

C_4 - (1 3 5 7 9) - (0 1 1 0 1) ⇒ $C_4 = 1$

∴ $C_1 C_2 C_3 C_4$

0 1 1 1

∴ error at 7th position.

Overflow:

→ Overflow is a condition where the result of an operation doesn't fit in the given space.

Overflow in unsigned numbers

→ In unsigned numbers addition

overflow occurs ⟺ Carry occurs.

Overflow in 2's complement number system:

→ adding one +ve & one -ve never results in overflow

→ If adding two -ve numbers result in +ve then overflow has occured.

silly if adding two +ve numbers result in -ve then overflow has occured.

+ve + -ve
-ve + +ve
} → no overflow

+ve + +ve → +ve
-ve + -ve → -ve
} → no overflow

+ve + +ve → -ve
-ve + -ve → +ve
} → overflow occured

$A_3\ A_2\ A_1\ A_0$

$B_3\ B_2\ B_1\ B_0$

$S_3\ S_2\ S_1\ S_0$

Carry → $A_3 B_3 \bar{S}_3 + \bar{A}_3 \bar{B}_3 S_3$

Instead of using above method we can use Cin & Cout to determine overflow

Cin

Cout $A_3\ A_2\ A_1\ A_0$

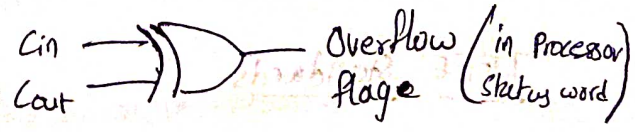
$B_3\ B_2\ B_1\ B_0$

$S_3\ S_2\ S_1\ S_0$

Carry occurs if Cout & Cin are different

Carry doesn't occur if Cout & Cin are same

Carry → $C_{in} \oplus C_{out}$



Floating Point Representation:

→ The floating point numbers are stored in mantissa (M), exponent (E) form.

→ Most of the notations represent mantissa as normalised sign magnitude fraction.

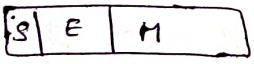
→ The normalization can be explicit or implicit.

→ The exponent is denoted in biased form

* The biased exponent is unsigned number, which can represent ~~sing~~ signed exponent of original number.

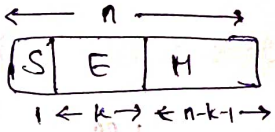
True Exponent = Biased exponent − Bias

→ The floating number is stored in the following way



S - Sign; E - Biased Exponend, M - Mantissa

→ If we use n bits



Biased Exponent: $0 \leq E \leq 2^k - 1$

Bias: 2^{k-1}

IEEE Standards

1985: Single precision, double precision

2008: half presion, quadraple precision (not needed for gate)

1985 standard:

→ The base of the system is 2.

→ Two kinds of precision
- single precision (32 bits)
- Double precision (64 bits)

→ The floating point number can be represented in
- * fractional form (.0 11 x ...)
- * Implicit normalised form. (1.M)
  - M → Mantissa

→ Certain mantissa and exponent combinations does not represent any number (Not a Number - NAN)

→ ~~Use~~ Used to represent division by 0. x/0

~~Single Precision~~

Floating point Representation:

100.1

S	M	E
0	1001	11

0.1001×2^{3}

S	M	E
0	01001	100

0.01001×2^{4}

So here in the above 100.1 has 2 representations

Silly we can have more representations for one number.

To overcome this we use normalization.

Normalization (Explicit Normalization)

Eg: 100.1

Move the radix point to the left of the 1st '1' from left-side

1001×2^3

S	E(4)	M(5)
0	0011	10010.

Eg: 0.0010.1

0.101×2^{-2}

Exponent can even be negative.

So we use 2's complement

S	E(4)	M(5)
0	1110	10100

Exp → -2

0010

2's comp → 1110

we calculate the number using below formula

$$(-1)^S \times 0.M \times 2^E$$

$= (-1)^0 \times 0.10100 \times 2^{-2}$

Note:

As exponent can be negative, while comparing two numbers it is bit difficult.

So to make it easy we use biasing. (i.e., add -2^{n-1} to E)

n is no of bits used for exponent.

Eg: Consider 0.101×2^{-2}

no of bits for exponent be 5.

biasing is done by adding $+2^{5-1} = 2^4 = 16$ to exponent

$-2 + 16 = 14$ → biased exponent.

S	E	M
0	01110	1010

$$\text{value} = (-1)^S \times 0.M \times 2^{E-\text{bias}}$$

$$\text{bias} = 2^{n-1}$$

$n \rightarrow$ no of bits used in exponent field

Consider $(4.875)_{10}$

$\cong (100.111)_2$

$\cong 0.100111 \times 2^3$

$$\begin{array}{r} 0.875 \\ 0.875 \\ \hline 1.750 \quad (1) \\ 0.750 \\ \hline 1.500 \quad (1) \\ 0.500 \\ \hline 1.000 \quad (1) \end{array}$$

Assume no of bits in exponent field = 4

" " " " Mantessa field = 5

∴ so we need to discard LSB '1' from mantissa

∴ we has only 5 bits

S	E	M
0	1011	10011

$E = 3 + \text{bias}$

$= 3 + 2^{4-1}$

$= 3 + 8 = 11 = (1011)_2$

Now let us calculate the value

$$\text{value} = (-1)^S \times 0.M \times 2^{E-\text{bias}}$$

$$= (-1)^0 \times 0.10011 \times 2^{11-8}$$

$$= 0.10011 \times 2^3$$

$$= 100.11$$

$$= 4.75$$

but original value is 4.875

Now have error.

To overcome this we can use implicit normalization.

Implicit Normalization :

↳ If nothing is given take implicit as default

Here we place the radix point on the right of first '1' from left side.

It because we already know that there will be one, we don't need to store it again.

100.111

$= 1.00111 \times 2^2$

biased exponent $= 2+8 = 10 = (1010)_2$

S	E	M'
0	1010	00111

$$\text{Value} = (-1)^S \times 1.M \times 2^{E-\text{bias}}$$

(implied / implicit)

Here value $= (-1)^0 \times 1.00111 \times 2^{10-8}$

$= 1.00111 \times 2^2$

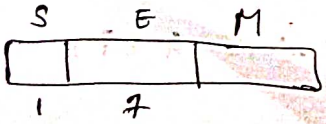
$= 100.111_2$

$= 4.875$

→ Implicit Normalization has no representation for '0'.

(Q) Consider a 16 bit register of the following format is used to store a floating point number. Mantissa (M) is denoted as normalized signed magnitude fraction, exponent (E) is expressed in excess-64 form. Base of the system is 2.

1) How many bits are allocated for fractional mantissa.



Excess 64

$\Rightarrow 2^{n-1} = 64$

$n-1 = 6$

$n = 7$

$\therefore$ bits for Mantissa $= 16 - 1 - 7 = \underline{8}$

2) what is the expression for decimal value?

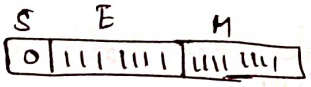
$(-1)^S \times 1.M \times 2^{E-64}$

3) what is value of the largest number that can be represented in base 10.

Sol:

Sign is positive

exponent & Mantissa has to be as large as possible



$(-1)^0 \times 1.1111\ 1111 \times 2^{127-64}$

$1.1111\ 1111 \times 2^{63}$

$1111\ 1111 \times 2^{55}$

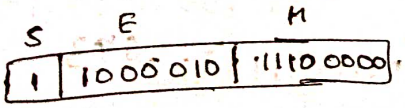
$(2^9-1) \times 2^{55} = 2^{64} - 2^{55}$

4) what is 16 bit pattern for $(-7.5)_{10}$

$-(111.1)_2$

1.111×2^2

biased exp $= 2 + 64 = 66 = (1000010)_2$

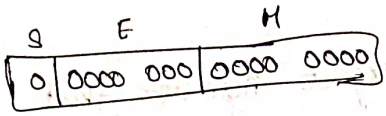


→ Increasing no of bits in exponent increases the range.

→ Increasing no of bits in mantissa increases accuracy i.e, precision

5) What is the difference b/w 1st smallest +ve number and 2nd smallest positive number.

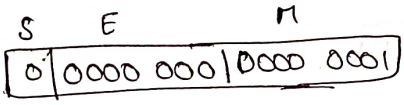
1st Smallest positive



$(-1)^0 \times 1.0000\ 0000 \times 2^{0-64}$

$= 1 \times 2^{-64} = 2^{-64}$

2nd Smallest positive



$(-1)^0 \times (1.000000001) \times 2^{-64}$

$(1 \times 2^{-64}) + (0.0000\ 0001) \times 2^{-64}$

difference $\longrightarrow (0.0000\ 0001) \times 2^{-64}$

$2^{-8} \times 2^{-64} = 2^{-72}$

6) Find difference b/w 1st highest +ve number & 2nd highest positive number.

1st highest $\longrightarrow (-1)^0 \times (1.1111\ 1111) \times 2^{127-64}$

$(1.1111\ 1111) \times 2^{63}$

$(1111\ 1111) \times 2^{55}$

$(2^9 - 1) \times 2^{55}$

2nd highest $\longrightarrow (-1)^0 \times (1.1111\ 1110) \times 2^{127-64}$

$(1.1111\ 1110) \times 2^{63}$

$(1.1111\ 1111) \times 2^{63} - (0.0000\ 0001) \times 2^{63}$

diff → $(0.0000\ 0001) \times 2^{63}$

$2^{-8} \times 2^{63} = 2^{55}$

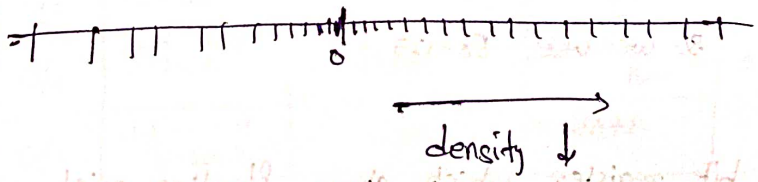
Here difference is very huge.

Note:

→ The numbers ~~are densly~~ that are closed to '0' are densely distributed and as we move away from '0' the numbers are sparsely distributed.

→ So number that are close to '0' have less error.

→ This happens because numbers which are close to zero has less exponent value (magnitude)



7) Find pattern for $(-16.125)_{10}$

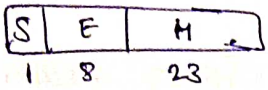
$= -(10000.001)_2$

$= -1.0000\ 001 \times 2^4$

biased exp → $4 + 64 = 68$

S	E	
1	1000100	0000 0010

Single Precision



(Excess 127)

S(1)	E(8)	M(23)	value
0/1	0000 0000 E=0	0000 ..... 0 M=0	± 0
0/1	1111 1111 E=255	0000 ... 0 M=0	±∞
0/1	1≤E≤254	M = xxx ... x	Implicit Normalized form → $(-1)^S (1.M) \times 2^{E-127}$
0/1	E=0	M ≠ 0	fractional form → $(-1)^S (0.M) \times 2^{-126}$
0/1	E=255	M ≠ 0	NAN

↳ This is reserved & it doesn't represent anything

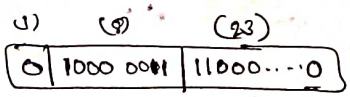
If we use ex-128

then it may result in value of E = 255

So we use Ex-127

(Q) Consider the 32 bit register which stores floating point numbers in IEEE single precision format.

1) what is the value of the number, if 32 bits are as given below



∴ It falls under 1≤E≤254

∴ Implicit normalized form

~~E = 131 - 127~~ E = 131 - 127

= 4

$(-1)^S (1.1100...0) \times 2^4$

11100.0000...0 = 28

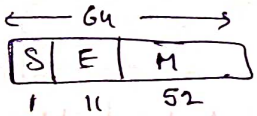
2) what is the value of the number in decimal for given bit pattern

$$\underbrace{0}_{S}\ \underbrace{11111111}_{E}\ \underbrace{10000\cdots0}_{M}$$

$E = 255$ & $M \neq 0$

$\therefore$ It is NAN

Double Precision:



$(Ex - 1023)$

S(1)	E(11)	M(52)	value
0/1	000...00 E=0	0000...00 M=0	±0
0/1	111...11 E=2047	0000...00 M=0	±∞
0/1	1≤E≤2046	M=XXX...X	Implicit norm. Form → $(-1)^S (1.M) 2^{E-1023}$
0/1	000...00 E=0	M≠0	Fractional form → $(-1)^S (0.M) \times 2^{-1022}$
0/1	111...11 E=2047	M≠0	NAN

→ $(-1)^S (1.M) \times 2^{E-124}$

→ $(-1)^S (0.M) \times 2^{-124}$

→ This is reserved if it doesn't represent anything

value of E=255

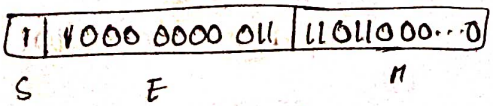
oating point numbers

2 bits are as given below

54

④ Consider the 64 bit register which stores floating point numbers in IEEE double precision format

1) what is the value of below 64 bit pattern



$E = 1024 + 3 = 1027$

$\therefore$ Implicit normalized form

$(-1)^1 \times (1.1101100\ldots0) \times 2^{1027-1023}$

$-(1.11011) \times 2^4 = -11101.1 = -29.5$

2) Find value

$E = 2047$

$M = 0$

$\therefore \infty$

(8) Represent $(117)_{10}$ in IEEE single & double precision format

$(117)_{10} = (1110101)_2$

$= 1.110101 \times 2^6$

Single Precision :

Biased exp $\rightarrow 6 + 127 = 133$

i.e., 10000101

S(1)	E(8)	M(23)
0	10000101	11010100...0

Double Precision :

Biased exp $\rightarrow 6 + 1023 = 1029$

i.e., 1000 0000 101

S	E(11)	M(52)
0	1000 0000 101	11010100...0

② What is the range of floating point numbers using fractional form of IEEE 754 single precision format.

Sol:

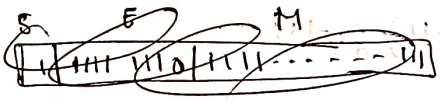
Smallest

E=0 M≠0

∴ fractional form

value → -

Smallest



value → $(-1)^1 (1.111\ldots11)_{23} \cdot 2^{254-127}$

$= -(1.111\ldots11)\, 2^{127}$

$= -(1111\ldots11)\, 2^{124}$

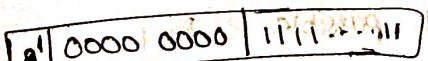
$= -(2^{24}-1)\, 2^{124}$

$= -(2^{148} - 2^{124})$

Given fractional form:

⇒ E=0, M≠0

Smallest: (-ve)



$(-1)^1 (0.1111\ldots11)_{23} \cdot 2^{-126}$

$= -(01111\ldots1) \times 2^{-149}$

$= -(2^{24}-1) \times 2^{-149}$

$= -(2^{23}-1) \times 2^{-149}$

Smallest (+ve) :

0	0000 0000	000 ... 1

$(-1)^0 \times (0.000\cdots01) \times 2^{-126}$

$2^{-23} \times 2^{-126} = 2^{-149}$

Largest +ve :

0	0000 0000	111 ... 11

$(-1)^0 \times (0.1111\cdots11) \times 2^{-126}$

$\underbrace{(1111\cdots11.)}_{23} \times 2^{-149}$

$(2^{23}-1) \times 2^{-149}$

$\simeq (1-2^{-23}) \times 2^{-126}$

(Q) IEEE 754 single precision format is used to store floating point numbers, what is the value of A - B?
where A is smallest representable using implicit normalized form, B is largest representable fractional form. A, B are positive.

Sol)

A : E ≠ 0 & E is smallest possible ⇒ E = 1

0	0000 0001	0000 001

$(-1)^S \times (1.000\cdots000) \times 2^{1-127}$

$(1.0000\cdots000) \times 2^{-126}$

1×2^{-126}

2^{-126}

2^{-126}

B :
2

from previous question

value of B = $2^{-126}(1 - 2^{-23})$

$\Rightarrow A - B = 2^{-126}(1 + 2^{-23} - 1 + 2^{-23})$

$= 2^{-126}(2^{-23})$

$= 2^{-149}$