Prerequisite:

1) Rounding off

13.46
↳ $0 \le n \le 4$ same
↳ $5 \le n \le 9$ +1 previous digit

2)

Unit	Time	Bit/Byte/address
Kilo (K)	10^3	2^{10}
Mega (M)	10^6	2^{20}
Giga (G)	10^9	2^{30}
Tera (T)	10^{12}	2^{40}
milli (m)	10^{-3}	2^{-10}
micro (μ)	10^{-6}	2^{-20}
nano (n)	10^{-9}	2^{-30}
Pico (p)	10^{-12}	2^{-40}

3) 2^0 - - - - 2^5 - - - - - 2^9 2^{10}

1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024(0)

Basics:

1) Architecture
- CPU design → instruction set
  ↳ Addressing modes
- Data format

2) organization
- IO organization
- Memory
- Performance improvement (Pipelining)

3) Data (in binary)
- Number
  - Fixed point representation
    - Signed
      - ↳ 1's comp.
      - ↳ 2's comp.
      - ↳ Sign magnitude
    - Unsigned
  - Floating point repr.
- Character
  - ↳ ASCII
  - ↳ EBCDIC (Extended binary coded decimal interchange code)

NOTE: 1 Byte = 8 bit
B → Byte ; b → bit

* Everything in memory is in binary. CPU need to know format to understand it.

4) Buses
- Address bus (Unidirectional) CPU to memory/IO
- Data bus
- Control bus (Bidirectional) → (⇄)

5) control signal send by.... to CPU:-
- IO devices : interrupt signal
- Memory : wait, Ready signal

6) There is only 1 each GPR, in system

7) 32 bit architecture means AC is of 32 bits ie max i/p size to ALU is 32 bits. & MDR size = 32 bits.
word size of CPU = 32 bits

8) word size of CPU & memory are different thing.
If memory is given word addressable & its word size not given then consider it equal to word size of CPU.

9) Flag register is updated after each ALU operation.

10) Program status word (PSW) = PC + Flag

11) Address register & Data register are used to access memory.

MicroOperation:

1) Smallest operation that CPU can perform in single step.

2) Denoted by Register transfer lang.

3) Time taken to execute 1 uop is called 1 CPU cycle time.

4) when 2 uop are independent, they can be performed simultaneously.

5) Types of uop:
- Arithmetic
- Logical
- Shift

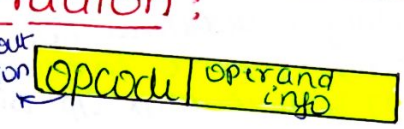
6) Types of Shift upp:
- Logical shift } unsigned no
- Circular
- Arithmetic } signed no
  ↳ after shift sign should remain same
  If sign changes it gives Arithmetic left shift overflow error.

7) Memory → Byte addressable (Default)
↳ word ← 1 word will be read or written at a time

NOTE: No. of Inst. from I_a to I_b = $(b-a+1)$

8) If memory has n cells (bytes or words) then addr size (in bits) = $\lceil \log_2 n \rceil$
CPU of that system should have min size of
AR, SP, PC = $\lceil \log_2 n \rceil$ bits
Addr bus should have atleast $\log_2 n$ lines

Instruction:



i) If opcode is of n bits, max no. of unique operations CPU can perform = Max no. of inst" supported by CPU = 2^n = size of ISA (type)

2) Inst" set Architecture (ISA) is set of all type of inst" supported by CPU.

3) Change of inst" is costlier process, so avoid it.

4) Max 3 addresses can be specified within each instruction. (4 addr not used now)

5) AC based archi supports only 1 address or 0 address inst"
Stack based archi" → 0 " "

6) If a system supports x address inst" then it can support all addr inst" below x.

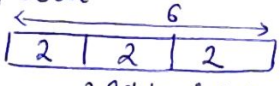
7) If a system supports x address inst" then there must be atleast 1 inst" of this type supported.

8) If there are 'n' no of GPR's, no. of bits for register reference = $\log_2 n$

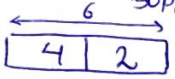
9) Multiple inst" supported: Always start solving with inst" having min no of opcode bits.

10) In n bit data
- Unsigned nos: max = $(2^n - 1)$
  min = 0
- Signed nos: max = $+(2^{n-1} - 1)$
  min = $-(2^{n-1} - 1)$ signed magnitude & 1's comp
  (-2^{n-1}) 2's comp

11) • Fixed length inst" ⇒ variable length opcode
• Variable len inst" ⇒ fixed len opcode
(size of opcode = $\log_2$ (Total no. of inst" supported)

12)


2 Addr inst"



1 Addr inst"

- To get max 1 addr inst", use one 2 addr inst"
- To get min 1 addr. inst", use 3, 2 addr inst"

Addressing Mode:

1) Effective address: operand addr or target address.
Its always memory addr not register reference.

2) 6 phases of inst" cycle:
- *Inst" fetch → Fetch cycle
- *Inst" decode
- Eff addr calculation → Execution cycle
- Operand fetch
- *Execution
- Write back result

3) In execution phase of branch (jump) inst" condition is checked and value of PC is updated accordingly

4) Special purpose registers (SPR's) are implicitly accessed by CPU, no need of explicit address.

5) Types of addressing modes:
- Implied (INCA)
- Immediate - to initialize registers with constant
- Direct / Absolute
- Indirect - used to implement pointer (2 memory access)
- Register - no effective address
- Register indirect - used to shorten inst" length (compared to direct mode)

NOTE: Register access time is negligible.

- Autoincrement / Autodecrement (Post increment) (Predecrement) used to access array / loop
- Indexed / Index register (2 inst" executed)
  Eff addr = Base addr + index value
  (inst") (index register)
  - Used to access an array element
  - 1 time memory access.
  - 2 inst" executed, 1 to set index register value and to fetch operand.

- PC Relative mode :- intrasegment jump
  Eff. addr = PC + Offset
  ↳ present in inst"
- Base register mode - intersegment jump
  Eff. addr = Base addr + Offset
  ↳ Base register ↳ inst"

* Forward jumping → +ve offset
  Backward " → -ve "

6) Auto incr, decr, indexed, scaled PC relative, Base register modes are computable, rest are non computable

7) PC relative & Base register mode supports reallocation.

8) When inst" is fetched, PC is incremented by size of current inst".

NOTE: If inst" size not given, take by default 1.

CPU: 1 inst" → 6 phases → each phase has many uop"

1) CPU cycle: time taken to perform 1 uop".

2) Clock rate / clock freq = $\frac{1}{\text{CPU cycle}}$ (unit Hz)
   no. of uop" performed in per sec cycles

3) Avg inst" execution time
   = CPI avg * 1 CPU cycle
   For n inst", = n * CPI avg * 1 Cycle time

4) MIPS (Million inst" per sec)
   $MIPS = \frac{\text{no of inst"}}{\text{Total exec time}} * 10^{-6} = \frac{10^{-6}}{CPI_{avg} * \text{cycle time}}$
   $= \frac{\text{clock freq} * 10^{-6}}{CPI_{avg}}$
   - MIPS is not a good parameter. (to judge CPU Performance)

5) FLOPS - Floating point operations per sec. (good parameter)

6) ALU i/p: Operands, function code ↳ control signal
   o/p: Result, status

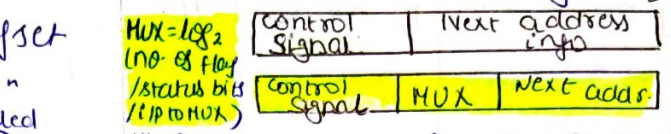
NOTE: If constant is present in inst" no need to put it in register to give to ALU. (directly give)

7) Control word: set of control signals to perform 1 uop".

8) Control word length is fixed for all operations.

9) Types of control unit:
   - Hardwired - faster, not flexible
   - Microprogrammed - flexible
   (control words r stored in control memory sequentially.

10) For each uop", 1 uinst" (or control word) is stored in control memory



11) Set of uinst" in control memory is called uprogram.

12) Types of microprogrammed CU:

Horizontal	Vertical (signals are grouped)
• 1 bit for each ctrl signal	• control word small
• control word large	• Decoder needed (for each grp)
• No decoder needed, faster	• slower
• Degree of ||ism more	• less

13) Signals which can't be grouped, r stored in horizontal manner.

14) Speed (no decoder)
Hardwired > Horizontal > vertical
(no memory access) (u-programmed)

15) If there r 10 signals in uprog CU & at a time at most 2 can be active make 2 grp each having (10 signal + 1) (1 for case when no signal is active)

16) Types of processor:

RISC	CISC
• Reduced inst" set computer	• Complex inst" set computer
• Less no. of inst"	• More
• Fixed length inst"	• Variable length
• Simple inst"	• Complex inst"
• Limited addressing modes	• More & complex addressing modes
• 1 cycle per inst"	• Multiple cycles
• More no of register	• Less no of registers
• Register to register arithmetic opr only	• Register to memory & memory to register
• Easy to implement using hardwired CU	• Difficult using hardwired CU.

NOTE: Stack grows upward means address increases.

NOTE: ROM is a control memory used in uprogrammed CU to store uinst".

* Expanding opcode is common in RISC to have fixed length inst".

IO Organization:

1) RAM is not IO device.
2) MM is directly connected to CPU & CPU connected to IO device via interface.
3) Speed: CPU > Main memory > IO devices

4) Memory mapped IO	IO mapped IO
• don't have separate address.	• have separate address
• Memory wastage	• no memory wastage
• distinguish memory & IO via address.	• distinguish by control signal
• More inst & addressing modes for IO also	• less for IO, more for memory
• inst & addr modes of memory r used for IO	• separate inst & addr modes for IO & mimory
• more IO devices	• less IO devices.
• ALU opr can be directly performed on IO data	• ALU opr can't be directly performed on IO data.

5) Baud → Bits/sec

6) Efficiency of transmission line = $\frac{\text{no of bits in data}}{\text{Total bits transferred (including synchronization bits)}} * 100\%$

7) Effective transfer rate = Efficiency * Transfer rate

8) In asynchronous data transfer, eff data transfer rate decreases, time increases.

9) Modes of Transfer:

(i) Programmed IO (Polling) (IO to CPU)

Time = time taken by IO to share its status register (based on IO speed) + Data transfer time (depends on IO speed)

NOTE: By default status register = 1 Byte

(ii) Interrupt initiated (IO to CPU)

- ISR - interrupt service routine (present in main memory)
- Part of driver of IO device.
- Internal interrupt - SIW interrupt
  External " - HIW "
- CPU don't know about external interrupt while executing inst".
- vector address is send via data bus.

NOTE: Among IO devices, harddisk has highest priority by default.

- Priority based interrupt handling
  - Software sol"
  - Hardware sol"
    - Serial
      - Daisy Chaining
    - Parallel

Time = interrupt + interrupt service overhead time
↳ checking priority if not given, take it 0

(CPU is blocked for data transfer = interrupt overhead + data preparation time + data transfer)

10) Performance gain or Speed up = $\frac{\text{Older tech time}}{\text{newer (faster) tech time}}$
= $\frac{\text{Performance of new}}{\text{Per of old}}$

if Speed up > 1 ⇒ new one better
Speed up < 1 ⇒ old one better

11) Programmed IO & interrupt IO may have data transfer blw memory & IO via CPU.

iii) DMA: can generate address & send control signal to memory

- Direct data transfer from IO to memory
- In DMA, CPU is used only once at starting to initialize DMA controller with starting address & data count.

Modes of DMA transfer:

→ Burst Mode: constantly transfering data when gets control; no overlapping

→ Cycle Stealing Mode:
- DMAC gets bus control for 1 CPU cycle
- Overlapping of time
- *Data is prepared with IO speed, transferred with memory speed
  time = preparation + transfer

→ Interleaving mode: CPU gives bus control only when its free.

12) % of time CPU is blocked due to DMA
t_x → time taken by IO to prepare 1 data
t_y → time taken by memory to transfer 1 data

- In burst mode = $\frac{t_y}{t_x + t_y} * 100\%$
- In cycle stealing mode = $\frac{t_y}{t_x} * 100\%$
- In interleaving mode = 0%

13) During DMA transfer, CPU state (register values) is intact.

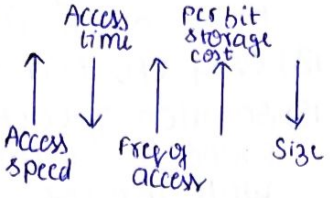
14) Speed:
Programmed IO < Interrupt < DMA
For very small data, DMA < Interrupt

15) In DMA, CPU can send HLDA even in blw of inst" execution.

16) In ques device speed given, use it to find Preparation time. Transfer time directly given or in terms of memory cycle.

Memory Organization:

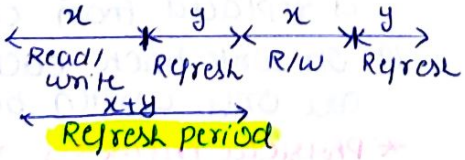
1) Internal memory (registers, cache)
   - RAM ROM Primary "
   - HDD Secondary "
   - Tertiary "



2) Memory representation: No. of cells * 1 cell size
3) By default - Byte addressable
4) Memory cycle time - time to access 1 memory location
5) Decoder used to access a specific cell using address.
6) RAM - volatile, ROM - non volatile
7) Memory access rate = $\frac{1}{\text{Memory cycle time}}$
8) Memory access decoder = a x b
   a = address size, b = no. of cells = 2^a
9) When CPU is turned On, program in ROM is runned for POST & Booting.
10) Bootstrap program's execution does booting.
11) Types of RAM:

Static (SRAM)	Dynamic (DRAM)
• Made of flip-flops	• Made of capacitors stores data in form of electric charge.
• Faster & costlier	• Slower & cheaper
• No refresh required	• Refresh needed periodically
• Idle power consumption less	• more
• Operational power consumption More	• Less
• used for cache memory	• used for MM.

12) In DRAM



% of time used for refresh = $\frac{y}{x+y} * 100\%$. % of time used for R/W = $\frac{x}{x+y} * 100$

13) Multiplication table for two n-bit unsigned number = $2^{2n} * 2n$ bits
14) Addition table for two n-bit unsigned number = $2^{2n} * (n+1)$ bits
17) During inst execution, DMA transfer can be done but not interrupt transfer
18) CPU sends 2 info to DMAC: Data count & starting address
19) Max data transferred using DMA without CPU intervention = $2^x - 1$
    x → no. of bits in data count
15) In RAM/ROM 1 pin is required for each line.
16) In RAM

Chip select	Read	Write	Operation
0	x	x	No opr
1	0	0	No opr
1	1	x	Read
1	0	1	write

17) If there's power pin, ground pin is also needed.
18) Total capacity = no. of chips * 1 chip capacity
19) Default storage unit is bits.
    Ex:- 32K x 1 RAM = 32K bit RAM
20) If require address more → Vertical arrangement
    If require data more on each cell → Horizontal arrangement
    If require address & data more → Hybrid arrangement
21) In vertical arrangement, MSB bits of address is used as chip select.
22) CPU can initiate memory request only when memory is ready.
23) In a refresh operation, one complete row of cells is refreshed.
24) 1 chip refresh time = no. of rows (or cells) * 1 refresh opr^n time
    = n chip refresh time
    All chips are refreshed parallely.
25) Associative memory (Content addressable memory) - faster than SRAM & expensive too.
    - used for TLB, cache
    - cells don't have address.
26) Speed: Associative memory > SRAM > DRAM
27) Spatial Locality of reference - accessing near by addresses again
    Temporal LOR - Accessing same address again.
28) If ram is of 64M x 16 bits then 64M cells r arranged in rows & columns.

Cache Memory: → Top level memory

1) Hit Ratio = $\frac{\text{No. of hits in cache}}{\text{Total memory reference (Hit + Miss)}}$

2) Miss Ratio = $\frac{\text{No. of miss}}{\text{(Hit + Miss)}}$

3) Hit Ratio + Miss Ratio = 1

4) General Formula: Avg memory access time
$T_{avg} = H * $ time required in case of hit $+ (1-H) * $ " " " of miss

5) Types of cache access:
Simultaneous & Hierarchical access

* H → Hit ratio
t_{CM} → cache memory access time
t_{MM} → main memory access time

6) Simultaneous access →
$T_{avg} = H * t_{CM} + (1-H)\, t_{MM}$

7) Hierarchical access → $T_{avg} = t_{CM} + (1-H)\, t_{MM}$
$T_{avg} = H * t_{CM} + (1-H)(t_{CM} + t_{MM})$

8) which formula to use:
if CM or MM access time given
- NO → Hit/miss time given → use general formula
- Yes → Hierarchy / Level keyword given
  - NO → use simultaneous access.
  - Yes → use Hierarchical access

9) If given, search time for Hit/Miss in cache is negligible → use simultaneous access.

10) Access time of memory does not change. Tavg changes with hit ratio.

11) $t_{CM} \le t_{avg} \le t_{MM}$
↳ 100% H ↳ 0% Hit rate

NOTE: cache → only cache
cached memory → cache + MM
uncached " → only MM

12) cached memory access time = Tavg

13) Tavg when LOR is used:
Simultaneous: $T_{avg} = H * T_{CH} + (1-H)\, T_{block}$
Hierarchical: $T_{avg} = T_{CH} + (1-H)\, T_{block}$
block access time from MM

14) T_{block} = block size * T_{MM}

15) On 1 T_{MM}, 1 MM cell is accessed.

16) If H = 100% ⇒ $T_{avg} = t_{CM}$

17) Miss penalty: extra time taken to bring block in cache in case of miss

18) Only CPU can access cache.

19) Solution of cache write or write propogation problem:
write through, write back

20) Tavg in write through cache:
- $T_{avg\,read} = H * t_{CM} + (1-H) t_{MM}$ (Simultaneous)
- $T_{avg\,read} = t_{CM} + (1-H)\, t_{MM}$ (Hierarchical)
- $T_{avg\,write} = t_{MM}$ or $\max(t_{CM}, t_{MM})$
  ↳ always simultaneous access
- T_{avg} = % of read Opr. * $T_{avg\,read}$ + % of write Opr. * $T_{avg\,write}$
  → in write through cache H for write = 0

21) Eff. Hit ratio for both read & write Opr. = % of read Opr. * Hit ratio for only read Opr.
- For write, Hit ratio = 0

21) Every time there is read miss, block is brought from MM to CM.

22) Every time there is write miss, block is not brought from MM to CM (no write allocate)

23) For better performance use
- write through with no write allocate
- write back with write allocate

24) For no write allocate, miss penalty = 0.

25) cache line ⇒ cache block

26) Addressing latency / Latency time of memory system is time taken by decoder to select a cell for operation

27) In write through cache, block is replaced from cache directly.

28) In write back cache, dirty blocks are only written back to MM.

* Physical memory → MM
* If using cache doubles speed of the memory, it means $T_{avg} = \frac{T_{MM}}{2}$
* Cache is accessed using internal bus, system bus is needed to access MM.

Cache Mapping:

1) CPU always generate MM address. → By default

2) cache mapping
   Direct | set associative | Fully assoc.

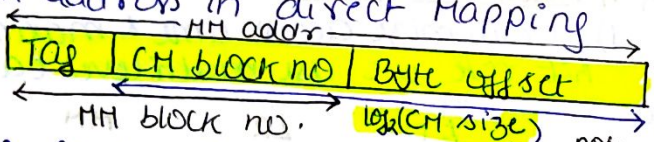
3) Mapping is applied on blocks.

4) Direct Mapping: (1way set assoc mapping)
   CM block no = MM block no % No of blocks in CM

5) Tag identifies among all MM blocks which maps to one index, which one is present in cache

6) NO of blocks in CM = Size of cache / 1 block size

7) MM address in direct mapping



8) NO of bits for Byte offset = $\log_2$ (Block size)

9) NO of bits for CM block no = $\log_2$ (no. of blocks in CM)

10) Block size of CM & MM is same.

11) Index in direct mapping → CM block number

12) Tag size in direct mapping = MM address size − $\log_2$(CM size)

13) Byte offset is not used to check hit/miss in any of the mapping

14) cache controller = H/w to check hit/miss + Metadata storage (like tag bits)

15) Metadata size or Tag directory size (for all mapping) = NO of blocks in CM * (Tag + extra bits)

16) Dirty bit - to know block is updated or not
    write through cache does not have
    0 → Not modified 1 → modified

17) CM is not addressable, it is indexable.

18) For a given cache size, block size & MM size, tag and CM block no is same for byte and word addressable MM only MM address & byte offset size changes.

19) If MM addr is given in decimal
- MM block no = ⌊Address / 1 block size⌋
- CM block no = MM block no. % no. of blocks in CM

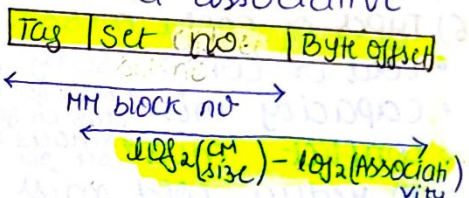
20) Set Associative Mapping:
CM set no = MM block no % no. of sets in CM

21) n-way set associative means n blocks at each set in CM.

22) NO of sets in CM = NO of blocks in CM / Associativity

23) Index in set associative → set no.

24) MM address in set associative mapping:

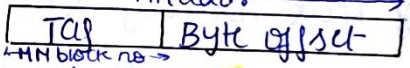


25) Tag bits is maintained for each block in cache.

26) Tag bit size in set assoc mapping = MM addr size − ($\log_2$(CM size) − $\log_2$k)
* k → Associativity

27) If associativity is doubled, set no size reduces by 1 & tag size increases by 1.

28) Fully Associative Mapping:



29) index bit = 0 bits

30) Mapping: x % 1 = 0 ← single set
(x: MM block no; 1 ← NO of set)

31) Tag bit size = MM addr size − $\log_2$(Block size)

32) Tag = MM block no.

33) Size of tag is max in fully assoc & min in direct mapping.

34) Size of index is max in direct mapp. & min in fully assoc.

35) Max difference in tags (of directly mapping & fully assoc) = $\log_2$(NO of block in CM)

NOTE: If given CPU is 32 bit arch, it never means memory is word addressable. By default memory is byte addressable.

Block Replacement:

1) Direct mapping don't need replacement policy. (because block placing at 1 index)
2) Replacement policies:
   - FIFO
   - Optimal
   - LRU
3) In case of miss penalty, block is transferred to CM byte by byte if MM is byte addr. Otherwise word by word. (MM & CM accessed multiple times to transfer 1 block from MM to CM)
4) Transfer rate ≡ Bandwidth
5) Finding transfer rate of MM from miss penalty:
   Find cycles required to transfer block → Time → transfer rate
6) Types of cache misses: (block accessed 1st time)
   - cold or compulsory miss
   - capacity miss → block got replaced because CM got full
   - conflict miss → replaced due to set got full

   (check in this order)
7) To reduce cold miss - increase block size
8) At a time max cold miss possible = no of blocks in MM
9) To reduce capacity miss - increase CM size
10) To reduce conflict miss - increase associativity.
11) There is no conflict miss in fully associative cache.

Direct Mapping Hardware:

1) NO. of MUX for tag selection = tag bit size
2) Size of MUX = NO. of blocks in CM : 1
3) NO. of comparator = 1
4) Size of comparator = tag bit size
5) Hit latency = 1 MUX delay + 1 comparator delay
   ↳ time to declare Hit ↳ All MUX works parallelly

K-way set associative mapping HW

1) Num. of MUX for tag select = k bunch = k * tag bit size
2) Size of MUX = NO. of sets in CM : 1
3) NO. of comparator = K
4) Size of comparator = tag bit size
5) OR gate = 1 (k input OR gate)
6) Hit latency = MUX delay + comparator delay + OR gate delay

NOTE: 4 i/p OR gate ≡ three 2 ip OR gate
2 i/p OR gate ≡ 2:1 MUX

Fully associative mapping HW:

1) MUX not required (no need to select set)
2) NO. of comparator = NO. of blocks in CM
3) Size of comparator = tag bits
4) OR gate = 1 (i/p = no. of blocks in CM)
5) Hit latency = comparator delay + OR gate delay

NOTE: If any of the delay is not given, take it zero.

- Associative memory can be used to implement fully associative cache.

Multilevel cache:

1) To minimize access time & maximize hit rate we use multilevel cache.

	L1 cache (smaller)		L2 cache (bigger)		MM
Access time:	t_1	<	t_2	<	t_3
Hit ratio:	H_1	<	H_2	<	H_3

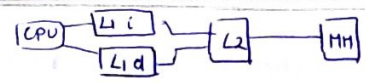
2) Simultaneous access:
$T_{avg} = H_1 t_1 + (1-H_1) H_2 t_2 + (1-H_1)(1-H_2) t_3$
3) Hierarchical access:
$T_{avg} = t_1 + (1-H_1)[t_2 + (1-H_2) t_3]$
$= H_1 t_1 + (1-H_1) H_2 (t_1+t_2) + (1-H_1)(1-H_2)(t_1+t_2+t_3)$
4) Prob of access of L1 = H_1
   " " " L2 = $(1-H_1) H_2$
   " " " MM = $(1-H_1)(1-H_2)$
5) Total access time = Number of memory reference * Tavg

Rel^n b/w content of L1 & L2:

1) Inclusion policy: content of L1 should be present in L2, value inside block may be different.
2) Exclusion policy: content of L1 need not be present in L2.
   ↳ L2 contains only blocks which are replaced from L1. so L2 is called victim cache

* Smaller chip has lesser access time.
* Smaller block has less miss penalty.

Dual cache



1) Tavg for instruction (Hierorchical)

$T_{avg\ inst} = H_{i\text{-}cache} * T_{ic} + (1 - H_{i\text{-}cache})[H_2 * (T_{ic} + T_2) + (1 - H_2)(T_{ic} + T_2 + T_{MM})]$

2) $T_{avg\ data} = H_{dc} T_{dc} + (1 - H_{dc})[H_2(T_{dc} + T_2) + (1 - H_2)(T_{dc} + T_2 + T_{MM})]$

3) $T_{avg} = \left(\%\text{ of inst}^{n}\text{ access} * T_{avg\ ins}\right) + \left(\%\text{ of data access} * T_{avg\ data}\right)$

4) When block size of L1 & L2 is different in case of miss in L1, block will be brought from L2 to L1 whose size is equal to block size of L1.

5) In one cache access, one entire block is accessed.

NOTE: $T_{avg} \propto$ Miss rate

$T_{avg} \propto \frac{1}{\text{Hit rate}}$

x NOTE x

1) Pointer, recursion → indirect addr mode
Array, structure, record → indexed addr mode

2) To send 16 bit addr via 8 bit buses first 8 bit is send then 8 bit.
8 MUX (2:1) required.

3) In ques to find min register, if inst^{n} is given $R_2 \leftarrow R_2 + R_3$ don't perform it as $R_2 \leftarrow R_3 + R_2$.
↳ unless given to optimize

4) To have single data transfer using DMA, system bus is accessed twice one for data transfer from IO to DMAC & again from DMAC to MM.

5) (in DMA) when transfer is complete, DMAC sends interrupt signal to CPU.

* By default array is row major order.
* A sequence $\langle a_i \rangle$ is bounded above by K iff all $a_i \leq K$.
• Bounded lower by K iff all $a_i \geq K$
* Jump stmt may jump to same inst^{n}.

Magnetic Disk

1) Each recording surface has 1 read/write head.
At a time only 1 is active.

2) NO. of surface = 2 * NO. of platters

3) Sector of a track → disk sector

4) NO. of tracks on disk = 2 * NO. of platters * NO. of tracks per surface

5) NO. of sectors = 2 * NO. of platters * NO. of tracks per surface * NO. of sectors per track.

6) consecutive sectors collection → cluster

7) By default, each track sector has constant capacity
• Sectors of inner track has more density.
• constant angular velocity.

8) If sectors have variable capacity then has constant density & constant linear velocity. (Per unit track has constant capacity, so calculate by circumference)

9) NO. of Bytes on disk = NO. of sectors * 1 sector capacity

10) Smallest addressable unit → Sector

11) 1 sector is accessed at a time.

12) NO. of bits in disk address = $\log_2$ (NO. of sectors in disk)

13) Disk access time = Seek time + Rotational latency + 1 sector transfer time + other delays
↳ if not mentioned in ques take 0

14) Rotation of disk is only in 1 direction.

15) If current & target head are given, then find rotational latency using rotation speed.
• If not given then use average rotational latency.

Avg rotational latency = $\frac{1 \text{ Rotation time}}{2}$

16) 1 sector transfer time = $\frac{1 \text{ Rotation time}}{\text{NO. of sectors per track}}$

* Disk keeps rotating with some speed whether read/write takes place or not.

17) Rotational latency = $\left(\frac{1 \text{ rotation time}}{\text{NO of sectors per track}}\right)$

* NO of sectors to rotate (final - initial)

18) TO calculate rotation time from RPM, divide 60000 by RPM
Answer will be in msec.

19) Disk transfer rate is used in
- DMA to find data preparation time
- Programmed IO for finding status check time.

20) Multiple sector access time:
- Sequentially stored (cluster)
  t = 1 seek time + 1 rotational latency + n * 1 sector transfer time
- Random
  $t = n[$1 seek time + 1 rotational latency + 1 sector transfer time$]$

21) NO of cylinders = NO of tracks per surface.

22) Disk scheduling is done to have min seek time.

23) Addressing is done as
Cylinder → surface → sector
<C, h, S>
C → cylinder no, h → surface no, S → sector no

24) Sector no for given addr <C,h,S>
= C * no of sectors per cylinder + h * no of sectors per track + S

25) If sector no given S, find address
C = ⌊S / no of sectors per cylinder⌋
h = ⌊(S % no of sectors per cylinder) / sectors per track⌋
↳ remainder = S

26) If file is stored on n sectors, to find last sector no
Find starting sector no from addr, add (n-1)
↳ From sector no, find address of last sector.

27) In Disk + DMA ques
- Find transfer rate from track capacity & 1 rotation time.
- Find transfer time then solve as DMA ques.

28) If disk rotates with constant linear velocity, each track has different capacity. Find according to circumference of tracks.
Capacity / cm circumference is constant.

29) If disk rotates with constant angular velocity, capacity of each track is constant.

30) If no of platters not given take it as 1.

* We use concept of cylinder to store a file so that seek time required only once.
* Latency → delay
* If seek time is not given calculate average seek time.
* virtual memory size is limited by disk size & size of MAR (address register).
* seek latency is not proportional to seek distance due to arm starting & stopping inertia.
* Larger disk block/sector size means less no of blocks to fetch & hence better throughput.
* Disk requires device driver.
* RAID (Redundant Array of Independent disk) is used to provide fault tolerance & high speed (parallel accessing possible).
* Formatting of disk is done to store some metadata to make disk ready to use. So, formatted disk has slightly less capacity than unformatted disk.
↳ In each sector some metadata is stored.
For eff data transfer rate consider sector capacity = Sector capacity − Metadata
* Eg: Recording width = outer r − inner r = 4 − 2 = 2
if track width = 0.1 ⇒ No of track = 2 / 0.1 = 20

Floating Point Representation

1) Number is Represented as

Sign	Exponent	Mantissa

Sign: 0 +ve, 1 -ve
Exponent: biased form
Mantissa: normalized fraction part

2) Exponent is stored as +ve no.

$e + bias = E$

(original) signed — (stored) unsigned

3) If K-bits is exponent field

$bias = 2^{K-1}$

4) If $bias = n \Rightarrow E$ is of $(\log_2 n + 1)$ bit

5) Explicit normalisation: (By Default)

$0.1\_\_\_\_$ → Mantissa

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

6) Implicit normalisation:

$1.\_\_\_\_$ → Mantissa

$value = (-1)^S * 1.M * 2^{E-bias}$

7) excess-(64) form
→ bias

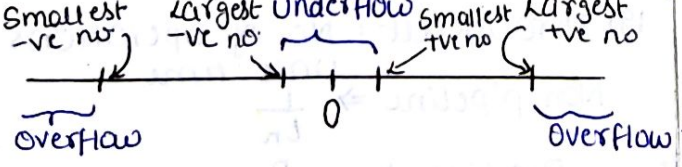
8) M is part after decimal so if required 0's are padded at end.

NOTE: (0x)49A8 or 49A8(H)
→ means its hexadecimal

9) Implicit norm. is better than explicit as it has more range.

10) More bits in exponent ⇒ more range
More bits in matissa ⇒ more accuracy

11)



12) conventional representation can not store zero and very small numbers. (underflow & overflow)

NOTE: ~~A digit floating point arithmetic~~ numbers are represented by signed ~~A~~ digit integer & an exponent. Rounding occurs when the digits of no exceed n.

IEEE 754 Floating point repres.

(overcomes drawback)

1) uses implicit normalisation.

2) Single Precision: (32 bits)

S	E	M
1	8	23

Bias = 127

3) Double Precision: (64 bits)

S	E	M
1	11	52

Bias = 1023

4) All 1's and all 0's are preserved value of E for special no representation. (← so bias is 1 less)

5) Range of E of normalized no.:
000…01 to 111…10

6)

S	E	M	Number
0	00…0	00…0	+0
1	00…0	00…0	-0
0	11…1	00…0	+∞
1	11…1	00…0	-∞
0/1	11…1	≠00…0	Not a number
0/1	00…0	≠00…0	Denormalized
0/1	≠00…0 And ≠11…1	XX…X	Normal (implicit) Normalised number

7) value (Implicit nor)
$= (-1)^S * 1.M * 2^{E-bias}$

8) For normalised no.
- min value of E = 00…01, M = 00…0
- max value of E = 11…10, M = 1.1…1

9) Very small nos which can't be normalized further are stored in denormalized number.

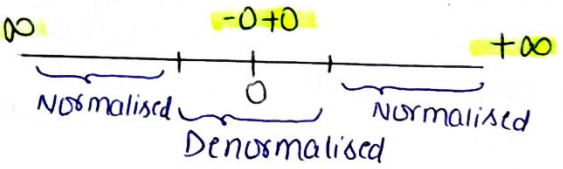
10) Min e (single precision) norm no.
$= E_{min} - bias = 1 - 127 = -126$

Min e (double precision) norm no
$= -1022$

11) Single Precision: (Denormalized no)
Value = $(-1)^s * 0.M * 2^{-126}$
Double Precision:
Value = $(-1)^s * 0.M * 2^{-1022}$

12) For denormalized no.
$E = 00\cdots0 = E_{min} = E_{max}$
$M_{min} = 00\cdots1$
$M_{max} = 11\cdots1$

13)



For max/min type ques remember this number line.

Pipelining: used when same process is done over multiple i/p

1) Flynn's classification of computers (von neumann)
- SISD (single inst stream, data stream (execute)) ↳ Pipelining
- SIMD
- MISD (↳ hypothetical)
- MIMD (↳ Superscalar computers, multiple pipelines)

2) 1 Operation performed in all segment is said as 1 task completed.

3) Pipeline cycle Time: min time in which all segment can perform their respective suboperation.

4) Cycle time → t_p ; n task, k segment pipeline
No. of cycles to compute 'n' task = $(k+n-1)$
Time needed = $(k+n-1) * t_p$

5) Non pipeline system takes t_n time to perform a task.
Time (for 'n' task) = nt_n

6) Speed up = $\frac{\text{Slower one time}}{\text{Faster one time}} = \frac{\text{Non Pipeline time}}{\text{Pipeline time}}$
$S = \frac{nt_n}{(k+n-1)t_p}$ $S \le n$

7) If $n \gg (k-1)$, neglect $(k-1)$ to fill the pipe
$S_{ideal} = \frac{t_n}{t_p}$ ← Max possible speedup

8) Special case: If time taken to perform 1 task in pipeline and non-pipeline is same i.e delay of all segment same
$t_n = k t_p$
then $S = k$

9) $S \le k$

10) If n not given or asked max S or ideal S ⇒ $S = \frac{t_n}{t_p}$
If n given ⇒ $S = \frac{nt_n}{(k+n-1)t_p}$

11) Clock rate or Frequency = $\frac{1}{\text{Cycle time}}$

12) In ideal condition, pipeline gives 1 output per cycle.

13) If delay of all segments are different, we use buffer for synchronization.

14) t_p = max(segment delay) + Buffer/Register delay

15) t_n = sum of all segments delay
↳ registers not required

16) If all segments are having same delay & there is no buffer
$S_{ideal} = k$

17) To have better performance, try to take segments with almost equal delay.

18) Latency: After how much time next i/p is given (Time taken by a task to complete)
Non pipelined ⇒ t_n
Pipelined ⇒ t_p

19) Throughput: No. of operations per unit time
Non pipeline ⇒ $\frac{1}{t_n}$
Pipeline ⇒ $\frac{n}{(k+n-1)t_p}$
(ideal case ⇒ $\frac{1}{t_p}$)

20) CPI of pipeline in ideal case without hazard = 1.

NOTE: segment ≡ stage both r same thg

Instruction Pipeline:

1) When branch inst" is detected in decode phase, all inst" which are fetched after branch inst" are removed from pipeline.

2) All branch inst" have stall cycles. $= i-1$ whether jump is taken or not.

3) Stall cycles / Bubbles: Extra cycles

4) If after ith stage of branch inst; execution outcome is available then no. of stall cycles = $i-1$ (due to branch)
= no of phase from decode to execution
= exe - fetch

5) If there are x branch inst", total stalls = x * no of stalls for ↓ branch inst"

Pipeline Hazards:

1) Situation that prevent next inst" to execute & leads to ~~Hazard~~, stalls.

2) Type:
- Structural hazard / Resource conflict
- Data hazard / Data dependency
- Control hazard / Branch difficulty

3) Structural hazard:
- when 2 segment try to use same resource at a time.
- no solution

4) Data hazard:
- Result of an inst" is used as i/p in next inst".

5) Sol" of Data dependency:

Software sol"
(for pipeline which can't detect data dependency)
(Done by compiler)
- Delayed load
  - ↳ inst" reordering
  - ↳ using no opr. inst"
  (delay dependent inst")

Hardware sol"
(for pipeline which can detect data dependency)
- H/w interlock (Default)
  - ↳ Have stall cycles
- Operand forwarding or bypassing
  - ↳ operand directly forwarded from AC to ALU.
  - ↳ NO stalls (for ALU to ALU dependency)

6) Stalls per dependency (Cd) = segment no. of write back − segment no. of (OF) Operand fetch

7) Total stalls = no of dependent inst" * stalls per dependency

8) No of stalls per dependency (Not consecutive) = Cd − No of inst" b/w dependent inst"
if its ≤ 0 ⇒ no stall

9) Load & store inst" is performed completely in execution phase. (no need of write back phase). (in operand forwarding)

10) operand forwarding can't be used if operand is not coming from ALU (load & store)

11) Sol" of Control Hazard:

Hardware sol"
- Branch prediction
  - ↳ stalls will be considered if prediction is wrong.
  - ↳ Static (always take or always don't take)
  - ↳ Dynamic (predict)

Software sol"
- Delayed branch (done by compiler)

12) Data ~~Hazard~~ dependency classification:

(i) RAW: True dependency
- H/w sol": Operand forwarding
- may have stalls
- Inorder dependency

(ii) WAW: False dependency (Output data dependency) (due to memory access)
- H/w sol": register renaming
- no stalls
- out of order dependency

(iii) WAR: False depe. (Anti dependency)
- H/w sol": register renaming
- no stalls
- out of order dependency

13) RAR don't have any conflict.

15) No of stalls due to structural hazard = extra cycles (x) each instruction is taking

16) CPI = Total no of cycles / Total no of inst"

NOTE: I_1: $R_1 \leftarrow R_2 + R_3$
I_2: $R_1 \leftarrow R_1 + R_4$ (*RAW)
I_3: $R_3 \leftarrow R_1 + R_0$
(It's only 1 dependency I1-I2 as dep. is b/w inst" not register)

17) Asynchronous pipeline: in a stage, different inst" takes different no. of cycles.

*CPI :

- Pipeline without hazard, $CPI = \frac{(k+n-1)}{n}$

  Ideal condition, CPI = 1

- Pipeline with hazard, $CPI = \frac{(k+n-1+x)}{n}$

  x → no. of stall cycles
  n → total no. of inst^n

  Ideal case (ignore k-1), $CPI = 1 + \frac{x}{n}$

18) Avg inst^n Exec time = CPI * Cycle time

19) $CPI_{ideal} = 1 + \frac{x}{n} = 1 + \left(\text{fraction of inst}^n \text{ causing stall} * \text{stall per inst}^n\right)$

  f, s

  $= (1-f)*1 + f*(1+s)$

20) In ideal case, with hazard

  $S = \frac{t_n}{CPI_{avg} * t_p}$ ← 1 inst^n exe time in non pipeline; ← 1 inst^n exe time in pipeline

21) classic RISC Pipeline :

- IF
- ID - Inst^n decode & Register read
  Branch inst^n :- decode, target addr calculation, condition evaluation, PC updation.
- EX - ALU operation executed
- MEM - memory accessed for load/store
- WB - write back result of ALU opr^n or load operation in register

22) Non linear pipeline : segments are not used sequentially. Ex: S1 S2 S3 S2 S3

23) Non linear pipeline :

- Find forbidden latency (which may cause collision)
- Find permissible latency
- Find collision vector
- Draw state diagram
  For each state, if 0 is at i^th place, right shift by i & take OR with original collision vector.
- Find greedy cycle
- Find avg " " , its min is ans. (↗ Min avg latency)

24) For a single task, non pipelined may work better than pipeline

  $Time_{non\ pipelined} \leq Time_{pipelined}$ [5] [6] [7] $t_n = 18$, $kt_p = 21$

25) Register renaming can eliminate all WAW & WAR hazards. → True

26) In ideal case without hazard

  Efficiency of pipeline = 100%.

- with hazard

  $Efficiency = \frac{S}{S_{max}} = \frac{S}{K}$ (S ← speedup; K ← no. of segments)

  Given speedup = η
  Max speedup

*Asynchronous pipeline ex :-

No of cycles	S1	S2	S3	S4 ← stages
I1	2	1	1	1
I2	1	3	2	2
I3	2	1	1	3
I4	1	2	2	2

	S1 (+1)	S2 (+1)	S3 (+1)	S4
I1	2	3	4	5
I2	3 (+1)	6 (max +3)	8	10
I3	5 (+2)	7	9	13
I4	6 (+1)	9	11	15 ← Total cycles

*Pipeline with single segment cannot provide parallel processing.

*For single i/p, non pipeline can perform better than pipeline.

*All stages are perfectly balanced → means all stages have equal delay

* $CPI_{avg} = 0.0x(1) + 0.0y * 0.0a(1+2) + 0.0y * 0.0b(1)$

  (x% → No stall; y% → a% → 2 stall, b% → No stall)

*Total exec time = no of inst^n * Avg inst exec time

*General pipeline : IF, ID, OF, EX, WB

EX: branch inst condition check, ~~load & store inst~~ completes on 5 stage pipeline

*Latency of an inst^n ^ is always less than on 5 stage pipeline. False

* 7 stage pipeline is always faster than 5-stage^n. False

* R1 ← M[1000]   (IF ID OF EX WB)
  R2 ← R1 + R3

- with operand forwarding, stalls = EX-OF = 1
- w/o " , stalls = WB-OF = 2

* w/o operand forwarding, always fetch operand after write back.

*Pipeline depth = no of segment = K

*Efficiency : % of time every stage of the pipeline is being used