Con-

Program Counter [PC]: Contains Staring address of the next instruction to be executed.

Memory Address Register: Contains memory address for either read/write operations.

DR/MDR/MBR: holds the instructions or data.

Instruction Register: Contains instruction which is currently executed by the CPU.

Accumulator: It contains temporary result of ALU operations.

GPR: for processing the data.

PSW: Stores the status of ALU result.

Stack pointer: Contains top of stack address.

Steps in Instruction Cycle

1. Instruction address calculation } Fetch Cycle
2. Instruction fetch }
3. Decoding
4. Operand address calculation } Execute Cycle
5. Operand fetch }
6. Data processing }
7. Result Storage. }

* If interrupt occur, then after completing current instruction execution, the interrupt will be Serviced.
* When interrupt occur it push the PC value into stack as a return address and control transferred to "Interrupt Sub Routine."

System Bus

1. Address line/bus: Carry address towards memory and I/O. (Unidirectional)
2. Data line/bus: Carry data (bidirectional)
3. Control lines/bus: Carry control signals (individually unidirectional and collectively bidirectional).

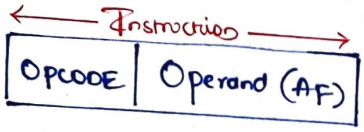
→ Memory is represented as $2^n \times m$. n = # address lines, m = # data lines.

* Address line Specifies the capacity of memory.
* Data line Specifies the capacity of data (Cell size).
* n bit address lines can represent 2^n memory cells, Range 0 to $2^n - 1$
* Default configuration in memory is byte addressable, and default in the CPU is word.

Pins: Processor Contain set of hardware pins to perform Operations.

1. Active low pin: Enabled when i/p is 0 or clock pulse in low state.
2. Active high pins: Enabled when i/p is 1 or clock pulse in high state.
3. Dual pins: 1 :- then memory Operations.
   0 :- then I/O operations.
4. Time multiplexed pins: this pin Carries multiple meaning, but one at a time. Address pins are time multiplexed with data pin to carry the data and address (one at a time)

Opcode: Operational Code, n bit opcode can perform 2^n operations.



→ N Operations, then
Opcode = $\lceil \log_2 N \rceil$ bits.

* If memory capacity is M, then address field = $\lceil \log_2 m \rceil$ bits

Elements of Machine Instructions

* Operation code
* Source operand reference
* Result operand reference
* Next Instruction reference

Storage.

→ Register (Fastest)

→ Memory → Cache memory
→ Main memory
→ Secondary memory

Faster: Registers > Cache > Main memory > Secondary memory.

Size: Secondary memory > Main memory > Cache memory > Registers.

* Register → Reg. A.F.
* Memory → Memory A.F

Immediate field: n bit.
→ Unsigned range = 0 to $2^n - 1$.
→ Signed range = -2^{n-1} to $+(2^{n-1}-1)$

Instruction Set Architecture

1. Stack organisation (0AF)
2. Single accumulator (1AF)
3. General register organisation.
   - Register memory reference (2AF)
   - Register - Register reference (3AF)

Expand Opcode Technique

1. primitive instruction (lower bit in opcode)
2. Derived instruction (higher opcode bit)
3. Further derived instruction (highest opcode bit)

Expand Opcode Steps

1. Identify the primitive instruction in the CPU
2. Calculate the total no of possible Operation
3. Identify the free opcode after allocating existing instruction
4. Calculate the no of derived instruction possible by multiply

$\Rightarrow$ Free opcode $\times 2^{\text{Increment bit in opcode}}$

* Addressing is a technique used to calculate the efficitive address.

Addressing Mode:

1. Immediate Am
2. Direct/Absolute Am
3. Memory indirect Am
4. Register direct Am
5. Register indirect Am
6. PC-relative Am.
7. Base Register Am
8. Indexed Register Am
9. Implied / Implicit Am
10. Auto decrement Am
11. Auto increment Am

1. Immediate Am: Operand are present in address field of instruction.

2. Direct/Absolute Am: Operand are present in memory and address field of the instruction contain effective address.

3. Memory Indirect Am: Operand and effective address present in memory, instruction Contains address of effective address.

4. Register Direct Am: Am operand are present in register.

5. Register Indirect Am: Am operand present in memory and effective address is stored in register.

6. PC relative Am: { EA = Current PC value + AF (offset) }

7. Base register Am: { EA = Base register value + AF (offset) }

8. Index register Am: { EA = Index reg val + AF (offset) }

9,10 Auto Increment and Decrement Am: Similar to register indirect In which register value increment or decrements.

11. Implied/Implicit Am: Am Operand are present in the opcode Itself.

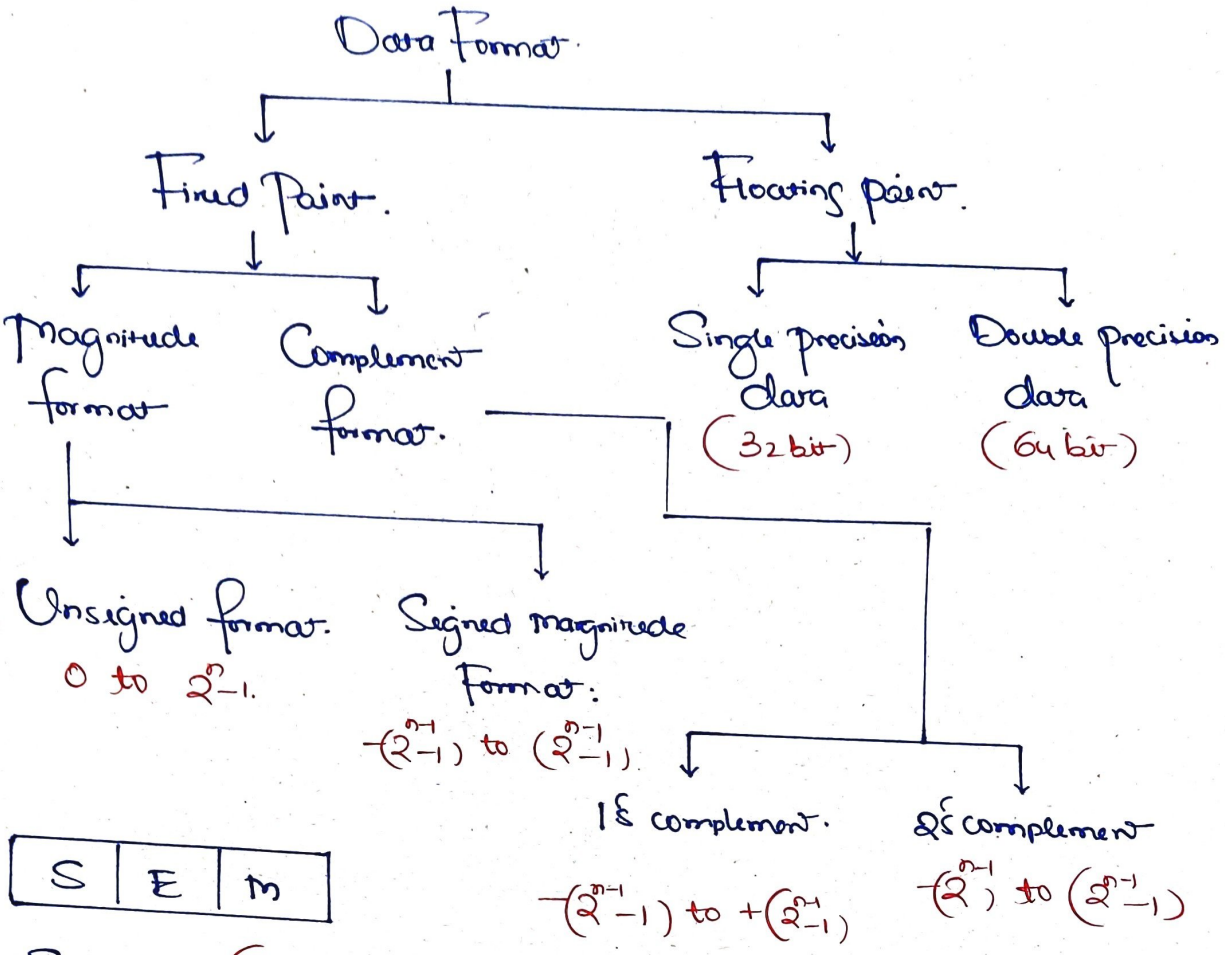
Constant → Immediate Am

Variable → Direct Am

Pointer → Indirect Am

Array → Index Am

Reallocation → Base Reg Am.



S	E	M

S : Sign (0: +ve, 1: -ve)

E : Exponent → E = e + bias, if exponent is 'k' bit bias = 2^{k-1}

M : Mantissa.

Mantissa Alignment.

* 2^{+shift}. Power (+) = Right alignment, ⇒ increment the exponent.
* 2^{-shift}. Power (−) = left alignment ⇒ decrease the exponent.

Explicit Normalised : $\underbrace{0.1\ldots\ldots}_{\text{Mantissa}} \times 2^e$. ⇒ $(-1)^S \times 0.M \times 2^{E-bias}$

Implicit Normalised : $\underbrace{1.\ldots\ldots}_{M.} \times 2^e$. ⇒ $(-1)^S \times 1.M \times 2^{E-bias}$.

IEEE 754. Floating point Representation

→ Single precision (32 bit), Excess : 127

S	E	M
1 bit	8 bit	23 bit.

→ Double precision (64 bit), Excess : 1023

S	E	M
1 bit	11 bit	52 bit.

Single Precision (32 bits), * double precision is similar with bits changed.

Sign (1 bit)	E (8 bits)	m (23 bits)	Value.
0 or 1	E=0	m=0	±0
0 or 1	11111111	00000... 0	±∞
0 or 1	1 ≤ E ≤ ~~254~~ 254.	m= ... any no.	Implicit normalised $(-1)^S \times 1.m \times 2^{E-127}$
0 or 1	E=0	m≠0.	Denormalised number $(-1)^S \times 0.m \times 2^{E-bias}$
0 or 1	E=255	m≠0	Not a Number

Little Endian: lower address contain lower byte and higher address contain higher byte.

Big Endian: Lower address contain higher byte and higher address contain lower byte.

ALU: "Arithmetic and logic unit", performs arithmetic and logical operations, comparisons and conditions checking.

→ No. of multiplexer = no. of bit in the register.

→ Size of multiplexer = no. of registers.

Micro-program: Sequence of micro operations to perform some operation in hardware level.

Control signals can be implemented in the control unit by

1. Hardwired CU design: (faster CU design, eg: RISC)
2. Micro programmed CU design.

Horizontal Programming	Vertical Programming
* Control Signals expressed in a decoded format.	* Control Signals are expressed in encoded format.
* 1 bit → 1 CS N bit → N CS	* n bit → 2^n CS. N. CS → $\lceil \log_2 N \rceil$ bits
* Supports longer control word.	* Supports shorter control words.
* No external decoder required.	* External decoder required
* Supports high degree of Parallelism	* Supports low degree of Parallelism

Speed : Hardwired > Horizontal > Vertical

Time Consumption : Vertical > Horizontal > Hardwired

Program Execution Time is based on:

1. Cycle
2. Cycle time, cycle time $\propto \frac{1}{\text{Clock freq}}$

Program ET = Ic × CPI × Cycle time.

Ic = instr count

CPI = cycle per Instn

Program ET = $\left[\sum (Ic_i \times CPI_i)\right]$ cycle time.

$$CPI = \sum_{i=1}^{n} \frac{(CPI_i \times I_i)}{Ic}$$

$$\text{MIPS rate} = \frac{Ic}{T \times 10^6} = \frac{f}{CPI \times 10^6} \text{ MIPS.}$$

$T = Ic \times CPI \times \tau$.

$\tau = 1/f$.

Ic = Instruction Count

Performance gain = Perf. of new / Perf. of old.

Pipelining: tasks are executed in overlapping manner.

$ET.\ pipeline = [k + (n-1)] \times tp$ — k = # stages (segments), n = no. of instructions, tp = each stage delay

$ET.\ non\ pipeline = n \times tn$ — (n = # instructions, tn = each instruction execution time)

$Speedup\ factor\ [S] = \frac{ET.\ non\ pipe}{ET.\ pipe.}$ | $S = \frac{tn}{tp}$ (Only when very large no. of tasks executed / or no of tasks not given)

* In ideal case when pipeline stages are perfectly balanced the one task execution in pipeline $= ET_{non\ pipe} = k \times tp$.

$Throughput = \frac{n = task}{[k + (n-1)]\ tp}$ (tasks completed per unit time) | $Throughput = \frac{1}{tp}$ — when no of tasks not given.

In uniform delay: 1 Task execution in pipeline = 1 Task E. in non Pipeline.
↳ tp = Stage delay + buffer delay.

In non-uniform delay: 1 Task $ET_{Pipe} \geq$ 1 Task $ET_{non\ pipe}$.
↳ tp = max (Stage delay + buffer delay)

$Efficiency = \eta = \frac{S}{K}$ — S = speedup, K = no of stages

RISC Pipeline Stages

1. Instruction fetch (IF)
2. Instruction decode (ID)
3. Execute (EX)
4. Memory Access (MA)
5. Write Back (WB)

* Every dependency not create stalls
* ANTI and OUTPUT dependency never Create stalls.
* Data dependency may/may not create stalls.

* If the target address of the branch instruction is available at k^{th} stage: # Stall from branch instruction = $k-1$.

* Average Instruction ET = 1 + (Branch instruction frequency × # Stall due to branch instruction)

(due to control hazards).

CPI ≠ 1.

→ CPI = (1 + # Stall/instruction)

→ # Stall per instruction = Branch instruction frequency × Branch penalty.

→ Avg Instruction ET = $\left(1 + \frac{\#\text{stall}}{\text{Instructions}}\right) \times$ cycle time

→ Speedup factor with Stalls: $S = \frac{\text{Avg Instruction ET}_{\text{non pipe}}}{\text{Avg Instruction ET}_{\text{pipe}}}$

{ CPU Registers } —(words)— { Cache } —(Block)— { Main memory } —(Page)— { Secondary memory }

mapping: Physical Address.

Paging: Virtual/logical address

Hit Ratio = $\frac{\text{No. of hit}}{\text{Total No. of access}}$.

Avg. memory Access Time (Tavg)

Tavg = hit × Time taken by memory during hit + (1 − hit) (time taken by memory during there is a miss)

Simultaneous Access: Tavg = h × tc + (1 − h) tm.

h: hit ratio

tc: Cache access time

tm: main memory access time

→ $T_{avg} = h_1 t_1 + (1-h_1) h_2 t_2 + (1-h_1)(1-h_2) t_2 t_3 \ldots (1-h_{n-1}) h_n t_n$

Hierarchical Access.

→ $T_{avg} = h_1 t_1 + (1-h_1) h_2 (t_2+t_1) + (1-h_1)(1-h_2) h_3 (t_3+t_2+t_1) + \ldots\ldots (1-h_{n-1}) h_n (t_n + t_{n-1} \ldots t_2+t_1)$.

→ $T_{avg} = h * t_c + (1-h)[t_{m1} + t_c]$

→ 1 word access time = T_{avg}.

$\#words/sec = \frac{1}{T_{avg}}$ { data transfer rate.

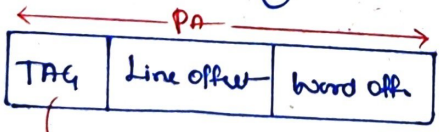
→ No. of mm blocks = $\frac{\text{mm size}}{\text{Block size}}$

→ No. of Cm blocks = $\frac{\text{Cm size}}{\text{Block Size}}$

Line offset = $\lceil \log_2 \#\text{lines} \rceil$

Word offset = $\lceil \log_2 \text{Block size} \rceil$

Direct Mapping.



TAG = $\frac{\text{mm Size}}{\text{Cm size}}$, TAG memory: ⇒ #cm lines × Tag bits

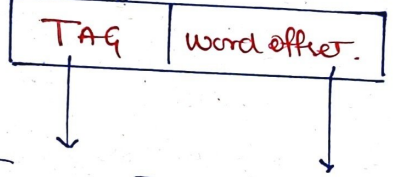
Mapping function: $k \bmod N = i$

k: mm block no.

N = no of cm lines

i = cache line no

← Physical address →

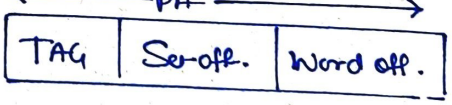


$\lceil \log_2 \text{mm size} \rceil$ $\lceil \log_2 \text{Block size} \rceil$

TAG memory size = #lines × Tag bits

(depends on mapping technique)

Set Associative Mapping



#Sets = $\frac{\#\text{lines}}{N\text{-way}}$ = $\frac{\text{Cm size}}{\text{Block size}}$.

Set offset = $\log_2$ Sets.

TAG = PA − (S.O + W.O)

Mapping: $k \bmod S = i$

(k: cache set, i: cache set no.)

* In direct mapping: Hit latency = latency of tag comparator.
* In set associative mapping: Hit latency: latency of tag comparator + latency of mux.

Direct mapping.	Set Associative
* No. of comparator = 1.	* No. of comparator = N-way
* Size of comparator = # bit in Tag.	* Size of comparator = # bit in Tag. Tag bit in N-way = Tag bit in direct mapping + $\log_2 N$ associative.

Local Miss Rate = $\frac{\#\text{miss in cache}}{\#\text{access to that cache}}$

Global Miss rate = $\frac{\#\text{miss in the cache}}{\text{Total \# CPU generated reference.}}$

Access Time in Multilevel Cache

Level 1: $T_{avg} = h_1 t_1 + (1-h_1)(t_m + t_1)$

or

$T_{avg} = t_1 + (1-h_1)(t_m)$

Level 2: $T_{avg} = h_1 t_1 + (1-h_1) h_2 (t_2 + t_1) + (1-h_1)(1-h_2)(t_m + t_2 + t_1)$

or

$T_{avg} = t_1 + (1-h_1)\left[t_2 + (1-h_2) t_m\right]$

Write Through

* Word updation time "t_w" = max (word updation time in cache, word updation time in mm)

→ $T_{avg}(\text{read}) = h_r t_c + (1-h_r)(t_m + t_c)$ (read hit: read data; read miss: read data from CPU (read allocate))

→ $T_{avg}(\text{write}) = h_r t_w + (1-h_w)(t_m + t_c)$ (write hit: word updation; write miss: write allocate: word updation)

→ $\eta_{wt} = \frac{1}{T_{avg\ WT.}}$ | Simultaneous Access : $T_{avg\ read} = h r t c + (1-h) t_m$.

Write Block Updating Technique

→ $T_{avg(read)} = h r t c + (1-h)\left[\%\ \text{data modified}\ (t_x + t_m + t_c) + \%\ \text{clean bit}\ (t_m + t_c)\right]$

→ $T_{avg(write)} = h w t c + (1-h)\left[\%\ \text{data modified}\ (t_y + t_m + t_c) + \%\ \text{clean bit}\ (t_m + t_c)\right]$.

→ $T_{avg(WB)} = F_r \times T_{avg(read)} + F_w \times T_{avg(write)}$

(F_r: % of read operation; F_w: frequency (%) of write operation.)

→ $\eta_{WB} = \frac{1}{T_{avg\ WB}}$

Rotational latency: Amount of time taken to rotate the track when the read/write head comes at to exact position.

⇒ Rotational latency = $\frac{1}{2}$ Rotation time

Disk Access Time = ST + Rotational latency + DTT. + Overhead (If any)

(ST: Seek time; DTT.: disk transfer time.)

→ Cylinder no 'c' = $\left\lfloor \frac{\text{Given Sector}}{\#\text{sector/cylinder}} \right\rfloor$

→ Surface no. 'h' = $\left\lfloor \frac{\text{given Sector} \% \ \#\text{sectors/cylinder}}{\text{Total Sector per track}} \right\rfloor$

→ Sector Number = $S + ST * h + ST \times TC \times C$.

(S: # Sector; ST: Sector per track; h: Surface no.; C: Cylinder no.)

I-O model

1. Programmed I/O. (Cpu time = Speed of I/O device)
2. Interrupt driven I/O (Cpu time = I/O interface time (latency))
3. DMA

* In programmed I/O and Interrupt driven I/O we cannot access the memory directly, we need help of CPU.

Data Count: The no. of bytes/words transfer from I/O to memory until the count value becomes '0'.

⇒ x: preparation time, y = transfer time

* % time CPU busy = $\left(\frac{x}{x+y}\right) \times 100$
* % time CPU blocked = $\left(\frac{y}{x+y}\right) \times 100$.