✳ Each hardware implementation of a thread, is a logical CPU.

◇ Eight great Ideas in Computer Architecture:-

- Design for Moore's Law:
- Use abstraction to simplify design.
- Make the common case fast.
- Performance via pipelining.
- Performance via parallelism.
- Hierarchy of memories.
- Dependability via redundancy.

▫ Response Time: The time required to give responce by CPU.

▫ Throughput: Total work done per unit time.

✳ Response time and throughput are affected by - changing processor of higher version.

◇ Performance $= \frac{1}{\text{Execution Time.}}$

$$ET = N \times CPI \times T.$$

$$ET = IC \times CPI \times T$$

$$ET = \frac{N \times CPI}{F.}$$

CPI Example:-

Q. Comp. A: Cycle Time = 250 ps, CPI = 2.0
Comp. B: Cycle Time = 500 ps, CPI = 1.2
Same ISA; which computer is faster and by how much?

→ $CPU\ Time_A = N \times CPI_A \times T_A$

$= N \times 2 \times 250 \times 10^{-12} = 500 \times 10^{-12} \times N.$

$CPU\ Time_B = N \times CPI_B \times T_B$

$= N \times 1.2 \times 500 \times 10^{-12} = 600 \times 10^{-12} \times N.$

$\therefore \frac{ET_B}{ET_A} = \frac{600 \times 10^{-12} \times N}{500 \times 10^{-12} \times N} = \frac{6}{5} = 1.2.$

∴ Computer A is 1.2 times faster than Computer B.

- If different instruction classes take different no. of cycles,

$$\text{Clock Cycles} = \sum_{i=1}^{n} (CPI_i \times \text{Instruction-Count}_i)$$

- Weighted average CPI:

$$CPI = \frac{\text{Clock Cycle}}{\text{Total Instruction Count}} = \sum_{i=1}^{n} \left(CPI_i \times \frac{\text{Instruction-Count}_i}{\text{Total Instruction Count}}\right)$$

Relative frequency.

Q. Alternative compiled code sequence using instruction classes A, B, C.

Class	A	B	C
CPI for Class	1	2	3
IC in seq. 1.	2	1	2
IC in seq. 2.	4	1	1.

Seq. 1.
Clock Cycles = 1×2 + 2×1 + 3×2
= 10.
Avg. CPI = $\frac{10}{2+1+2}$ = 2.

Seq. 2.
Clock Cycles = 1×4 + 2×1 + 3×1
= 9.
Avg. CPI = $\frac{9}{4+1+1}$ = 1.5.

Q. A 400 MHz processor was used to execute a benchmark program with the following instruction mix and clock cycle count.

Instruction Type	Instruction Count	Clock Cycle
Integer Arithmetic	450000	1
Data Transfer.	320000	2
Floating Point	150000	2
Control Transfer	80000.	2

Determine the effective CPI, and execution time for the program.

Total Clock Cycles = $45 \times 10^4 \times 1 + 32 \times 10^4 \times 2 + 15 \times 10^4 \times 2 + 8 \times 10^4 \times 2$
$= 10^4 (45 + 64 + 30 + 16)$
$= 155 \times 10^4$ Cycles.

: Average $CPI = \frac{155 \times 10^4}{(45+32+15+8) \times 10^4} = \frac{155}{100} = 1.55$

$ET = IC \times CPI \times T = \frac{IC \times CPI}{F}$

$= \frac{10^6 \times 1.55}{4 \times 10^8} = \frac{1.55}{4 \times 10^2} = \frac{15.5}{4} \times 10^{-3}$

T. = 3.875 ms.

Instruction type	CPI.	Instruction mix.
Arithmetic and logic	1	60 %.
Load/store in Cache	2	18 %.
Branch	4	12 %.
Cache Miss Memory reference.	8	10 %.

Average CPI = $1 \times 0.6 + 2 \times 0.18 + 4 \times 0.12 + 8 \times 0.1$

$= 2.24.$

MIPS:- (Million Instruction Per Second).

Rate.

MIPS rate = $\frac{1}{\text{ET of } 10^6 \text{ Instructions}} = \frac{1}{10^6 \times CPI_{Av} \times T} = \frac{F}{CPI_{Av} \times 10^6}$

∴ MIPS rate = $\frac{f}{CPI_{Av} \times 10^6}$.

CO&A Pg.56 William Stallings.

Q. $f = 400$ MHz. CPI = 1.55. find MIPS rate.

→ MIPS rate = $\frac{f}{CPI \times 10^6} = \frac{4 \times 10^8}{1.55 \times 10^6} = \frac{400}{1.55} = 258.065$

Q. $f = 400$ MHz. CPI = 2.24. find MIPS rate.

→ MIPS rate = $\frac{f}{CPI \times 10^6} = \frac{4 \times 10^8}{2.24 \times 10^6} = \frac{200}{1.12} = 178.57.$

Q. The execution time in seconds of 4 programs on 3 computers are given below. Assume, $IC = 10^9$

	Execution Time.		
Programs	Comp. 1.	Comp. 2	Comp 3
Program A	1	10	20
Program B	1000	100	20
Program C	500	1000	50
Program D.	100	800	100.

Consider each Program Contributes to 25 %. Find Mips of all 3 Computer.

Comp. 1: $ET_{AV} = 0.25 \times 1 + 0.25 \times 1000 + 0.25 \times 500 + 0.25 \times 100$

$= 400.25.$

$\therefore \frac{MIPS}{ET_{Million}} = \left(\frac{400.25}{10^9}\right) \times 10^6 = \frac{400.25}{10^3}$

$\therefore MIPS = \frac{10^3}{400.25} = 2.498.$

Comp. 2 = $ET_{10^3} = (10+100+1000+800) \times \frac{1}{4} = \frac{1910}{4} = 477.5$

$\therefore MIPS = \frac{1}{ET_{10^6}} = \frac{10^9}{10^6 \times 477.5} = 2.09.$

Comp. 3:

$\therefore Mips = \frac{10^9 \times 4}{10^6 \times (20+20+50+100)} = \frac{10^3 \times 4}{190} = 21.052.$

Benchmarks:-

Amdahl's Law:-

$$\text{Execution Time}_{(new)} = \text{Execution Time}_{(old)} \left[1 - \text{Fraction Enhanced} + \frac{\text{Fraction Enhanced}}{\text{Speedup Enhanced}} \right]$$

$$\text{Speedup}_{\text{Overall}} = \frac{ET_{(old)}}{ET_{(new)}} = \frac{1}{(1-f) + \left(\frac{f}{S}\right)}.$$

$$\therefore \boxed{\text{Overall Speedup} = \frac{1}{1-f+f/s.}}$$

Q. $f = 0.5$, $S = 1.6$. Find Overall speedup.

→ $O.S. = \frac{1}{1-f+f/s} = \frac{1}{0.5+(0.5)/1.6} = \underline{1.2307.}$

Q. $f = 0.2$. $S = 10$. Find O.S.

→ $O.S. = \frac{1}{1-f+f/s} = \frac{1}{0.8+0.2/10} = \underline{1.2195.}$

Patterson. Pg. 54.

Q. freq. of FP operations = 25 %.
Average CPI of FP operations = 4.0
Avg. CPI of other instructions = 1.33
freq. of FSQRT = 2 %.
CPI of FSQRT = 20.
Compare bet^n two design Alternatives:
(i) decrease CPI of FSQRT to 2.
(ii) decrease average CPI of all FP to 2.5.
Use processor performance eq^n.

Problem 6:

Q. When parallelizing an application, the ideal speedup is speeding up by the number of processors. This is limited by two things: percentage of appl^n that can be parallelized and the cost of communication. Amdahl's law takes into account the former but not later.

a] No. of processors = $N = 8$. $\therefore S = 8N$.
80 % process is parallelable. $f = 0.8$

$$\therefore 0.5 = \frac{1}{1-f+\frac{f}{S}} = \frac{1}{1-0.8+\frac{0.8}{N}} = \frac{N}{0.2N+0.8}$$

Q. First process : 80 %.
Second process : 20 %.
1. First process's 40 % process is speed up by 2 times.
Find overall speedup:

→ $\frac{1}{(1-0.8\times0.4)+\frac{0.32}{2}} = \underline{1.19.}$

2. Second Processe's 99 % Process is speedup by 2 times.
Find overall speedup.

→ $\frac{1}{(1-0.99\times0.2)+\frac{0.198}{2}} = \frac{1}{1-0.198+0.099} = \underline{1.1099}$

Q. $S = 10$. Enhanced mode is used 50 % Time.
1. Find total speedup.

→ $O.S. = \frac{ET_{old}}{ET_{new}} = \frac{50\%+50\times10}{100\%} = \frac{550\%}{100} = \underline{5.5.}$

2. What % of original execution time converted to fast mode?

→ Let, x = req. answer.
$\therefore 1-x$ = remaining time = 50 % time.
x = enhanced time = 50 % time.

$1-x = \frac{x}{10}$

$\therefore x = 0.909$
$x\% = 90.91\%$

· **Problem 7:**

Unpipelined processor. 1 ns cycle. (clock cycle).

ALU	Branches	Memory	operation.
4	4	5	cycles.
40 %	20 %	40 %	relative freqns

0.2 ns overhead is added to clock, by processor due to skew.
Ignoring latency impact, how much speed up in the instruction execution cycle (pipeline processor) will be gained in pipelining.

$$S = \frac{ET_{up}}{ET_p} = \frac{1\text{ ns} \times \left[\frac{(40+20)}{100} \times 4 + \frac{40}{100} \times 5\right]}{(1+0.2)\text{ ns}} = \frac{4.4}{1.2} \approx 3.7 \text{ times.}$$

· **Problem:**

We have single thread machine. 7 ns.
5 break up.
IF : 1 ns. ID : 1.5 ns.

Amdahl's Law for Fixed load :-

· **Degree of Parallelism :- (DOP)**

Kai_Hwang.
Pg. 134 (108)

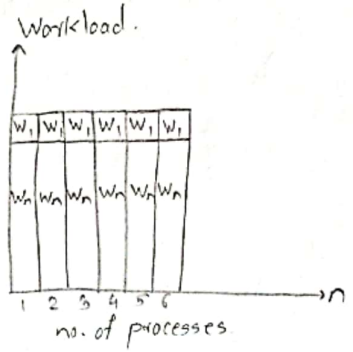
Amdahl's Law:

$$S_n = \frac{n}{1 + (n-1)\alpha}$$

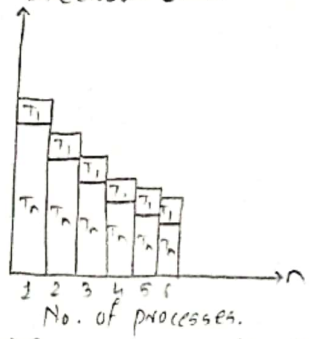
Here, system is used either in a pure sequential mode on one processor with a probability "α", or in a fully parallel mode using n processors with a probability $1-\alpha$.

Now,

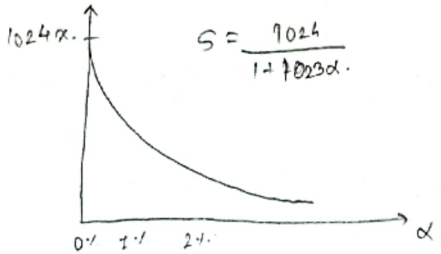
$$W_1 + W_n = \alpha + (1-\alpha) = 1.$$



(a) Fixed Workload.



(b) Decreasing execution time.



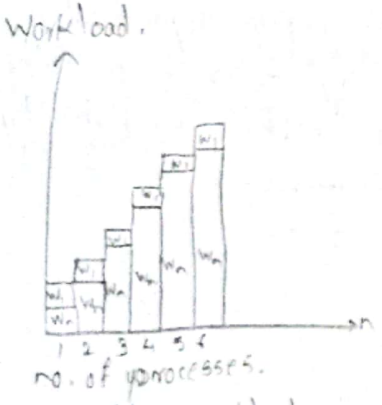
As $n \to \infty$, $S_n \to \frac{1}{\alpha}$.

$\therefore S_n$ is upperbounded by $1/\alpha$.

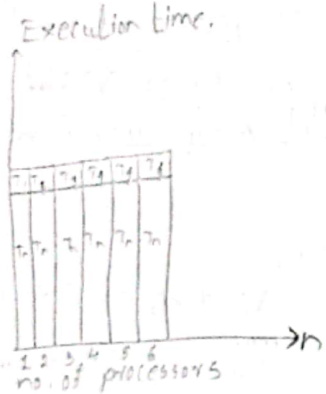
$\therefore \alpha$ is called as sequencial bottleneck of program.

□ Fixed Time Speedup:-

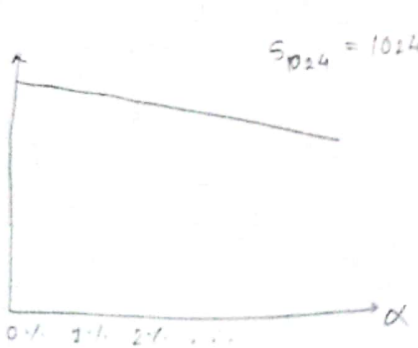
$$S_n' = \frac{W_1' + W_n'}{W_1 + W_n} = \frac{\alpha + n(1-\alpha)}{\alpha + (1-\alpha)} = n - \alpha(n-1).$$



(a) Variable workload.



(b) fixed time.



$S_{1024} = 1024 - 1023\alpha$.

□ Fixed Memory Speedup:-

$$S_n^* = \frac{W_1^* + W_n^*}{W_1^* + W_n^*/n} = \frac{W_1 + G(n)W_n}{W_1 + G(n)W_n/n}.$$

$\therefore$ Amdahl's Law and Gustafson's Law are special cases of fixed-memory model.

Performance Summary

- The big picture:-

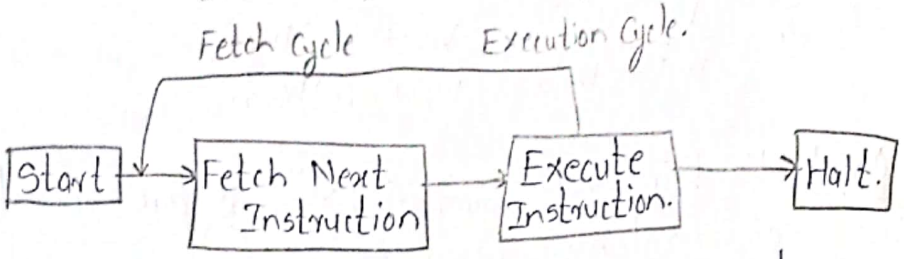
  William Stallings Pg. 65

- Central Tendencies of Data:-
  - Mean (Average)
  - Median. (Middle of sorted array)
  - Mode. (Datapoint with highest frequency).

Ⓜ von Neuman Architecture.

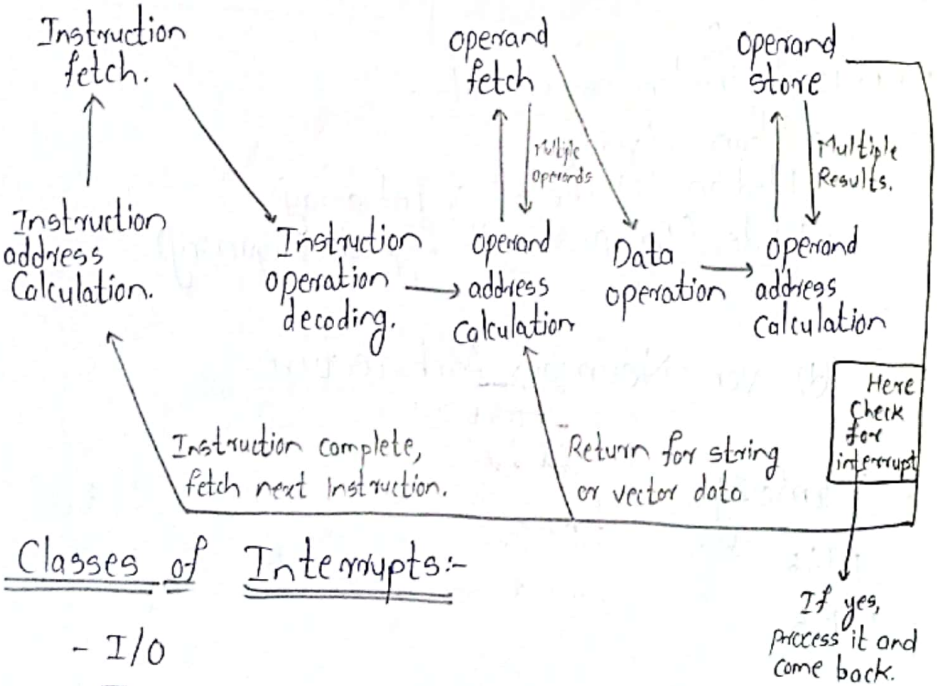
- Memory.
- MAR
- MBR

Basic Instruction Cycle.



- Fetch Cycle :- ⊛ Control Signals → Hardware / Micro-program

▫ Instruction Cycle State Diagram :-



- Classes of Interrupts :-
  - I/O
  - Timer
  - Program, etc.

⊛ Intel and AMD, fixed legth Instruction Instruction or Variable length Instruction?

RISC :- Reduced Instruction Set Computer.
CISC :- Complex Instruction set Computer.

▫ Machine Instruction Characteristics :-

- Types of Instructions, based on no. of addresses :-

1. 3 address instruction.

   OP d, operands.

2. 2 address instruction.

   OP d, s

3. 1 Address instruction.

   OP s

4. 0 Address instruction.

   OP.

$Z = K + B * C.$

eg. 3 Address.	2 Address.	1 Address.	0 Address.
	MOV R, B		First Convert it into post fix.
MPY T, B, C.	MPY R, C.	LOAD C	K B C * +
ADD Z, T, K.	ADD R, K	MPY B.	PUSH K
	MOV Z, R.	ADD K.	PUSH B
		STOR Z.	PUSH C
			MUL
			ADD
			POP Z.

⊛ Don't Change the Values of the registers.

eg. $Y = \frac{A-B}{C+(D \times E)}$

3-address ⟶ 4 instructions.
2-address ⟶ 6 instructions.
1-address ⟶ 8 instructions.
0-address ⟶ 10 instructions. $(A-B) \div (C+(D \times E))$

Postfix: ⟶ $AB-CDE*+\div$
10 instructions. (no. of operators + no. of operands + 1 (Pop the answer))

eg. $Z = (((P-(Q+S)*T)/(U/(V-W))))$

3-address:	2-address	1-address.	0-address.
ADD Z, Q, S.	MOV R_1, S	LOAD Q	7+6+1 = 14.
MPY Z, Z, T.	ADD R_1, Q.	ADD S	
SUB Z, P, Z.	MUL R_1, T.	MPY T_1	
SUB R_1, V, W.	MOV Z, P	STOR Z.	14-Instructions.
DIV R_2, U, R_1	SUB Z, R_1	LOAD P	
DIV, Z, Z, R_2	MOV R_1, V	SUB Z	
	SUB R_1, W.	STOR Z	
6-instructions.	MOV R_3, U.	LOAD V	
	DIV R_3, R_1	SUB W.	
	DIV Z, R_3	STOR R_2	
	10-instructions.	LOAD U	
		DIV R_2	
		STOR R_2	
		LOAD Z	
		DIV R_2	
		15-instructions	

□ Expanding Opcodes :-

When fixed length instructions are used, then smaller length opcodes (without operands) may waste the memory. (bits).

The idea of expanding opcodes is to make some opcodes short, but have a means to provide longer ones when needed.

When opcode is short, a lot of bits are left to hold operands.

If there is no operands (such as Halt), all bits can be used for opcode.

eg. 16 Registers of 16 bits.

→ 14 of type 3 address instruction.

0000 — — — } → 14.
1101 — — — }

→ 30 of type 2 address instruction.

escape code.
111 | 00000 — — } → 30.
111 | 11101 — — }

→ 28 of type 1 address instruction.

1111111 | 0000 — } → 28.
1111111 | 1011 — }

→ 64 of type 0 address instruction.

1111 1111 11 00 0000
1111 1111 11 11 1111 } → 64.

By Solving this all, we can add to get total no. of address:-

Type 3: 14×2^{12}
Type 2: 30×2^{8}
Type 1: 28×2^{4}
Type 0: 64

} Sum = 65,536
$= 2^{16}$ (size of Register)

- Machine with 16-bit instructions and 16 registers.

Instruction format can have several structures:-

- Opcode + Memory address (such as MARIE):
  4 KB bytes addressable space.

Opcode. Address.

- Opcode + Register Address.
  We need 4 bits to select one of 16 registers.

Opcode Address 1 Address 2 Address.

example 2:-

15 instructions with 3 address.
14 instructions with 2 address.
31 instructions with 1 address.
16 instructions with 0 address.
Can we encode this instruction set in 16 bits?

→

Sum = $15 \times 2^{12} + 14 \times 2^{8} + 31 \times 2^{4} + 16$

Sum = 65536. = 2^{16}.

Yes, we can encode it in 16 bits.

0000 — — — } → 15 3-address codes.
1110 — — —

1111 0000 — — } → 14 2-address codes.
1111 1101 — —

1111 1110 0000 — } → 31 1-address codes.
1111 1101 1110 —

1111 1111 1111 0000 } → 16 0-address codes.
1111 1111 1111 1111

Escape Code.

example 3:- Is it possible to design an expanding opcode to allow the following to encode with a 12-bit instruction? Assume a register operand requires 3 bits.

— 4 instructions with 3 registers
— 255 instructions with 1 register.
— 16 instructions with 0 registers.

→ $4 \times 2^{9} + 255 \times 2^{3} + 16 \times 1 = 4104$.

$2^{12} = 4096$.

Q. Assume a CPU having 16 registers and instruction length is 16 bit. It should be having:

14 type 3 address instructions.
31 type 2 address instructions.
12 type 1 address instructions.

Address size = $\log_2(16)$ (no. of registers)
= 4

Calculate max. no. type 0 address instructions it should have? Show appropriate encoding.

→ $14 \times 2^{12} + 31 \times 2^{8} + 12 \times 2^{4} + x = 2^{16}$

$\therefore \boxed{x = 64.}$

0000 — — — } 14 → 3 Address instruction.
1101 — — — }

1110 0000 — — }
1110 1110 — — } 31 → 2 Address instruction.

1111 1110 0000 — }
1111 1111 1011 — } 12 → 1 Address instruction.

1111 1111 1100 0000 }
1111 1111 1111 1111 } 64 → 0 Address instruction.

Escape Code.

Q. Is it possible to design an expanding opcode to allow the following to be encoded with a 12-bit instruction? Assume a register operand requires 3 bits.

- 7 instructions: 2 15-bit addr and 1 3-bit reg.
- 500 instructions: 1 15-bit addr and 1 3-bit reg.
- 50 instructions: with no addresses or regs.

→

$7 \times 2^{15} \times 2^{15} \times 2^{3} = 7 \times 2^{33}$

$500 \times 2^{15} \times 2^{3} = 500 \times 2^{18}$

50 patters.

Sum = 6.026×10^{10} bit patterns.

It is not possible with 12-bits, but possible with 36-bits.

With 36 bit patterns,

$2^{36} = 6.87 \times 10^{7}$.

So we need 36 bit patterns to create opcodes.

Q. A processor has 16-bit instruction length, 8-registers, and memory address of 8-bit. It is possible to have following types of instructions?

1. 3 no. of type that have 2 register fields 1 memory addr.

→ Yes! Possible. 8 registers → 3 bits.

$3 \times 2 = 6$ bits.

∴ remaining 2 bits. $2^2 = 4$ possible.

8 bits for memory addr.

∴ 3 are also possible.

∴ 00 — — —
01 — — —
10 — — —

2. 6 no. of instructions of type 1 reg. field and 1 memory addr.

→ 11 000 — —
11 101 — —

3. 6 no. of instructions of type that have all 3 reg. fields.

→ 11, 11, 0008 — — —

11, 11, 101 — — —

4. 3 no. of instructions with 1 memory address.

→ 11 11 11 00 ——

11 11 11 10 ——

5. 7 no. of instructions of type that have 2 reg. fields.

→ 11 11 11 11 00 10 → 3. — —

8 + Not possible + 3×2

14 → Can't represent 7 instructions using Only 3 are possible.

6. 7 no. of instructions of type that have only 1 reg. field.

→ 11 11 11 11 11 00 0 ——

11 11 11 11 11 110 ——

7 can be represented.

7. 8 no. of instructions of no. addr. no. reg. type.

→

11 11 11 11 11 111 000

11 11 11 11 11 111 111

8 can be represented.

Control :-

Hardware based Control

Micro-program based Control (Firmware).

- Pipeline Implimentation :-

IF	ID	IF	M	WB.

- There are 4 particular subcycles :-
  1. Instruction Cycle.
  2. Indirect cycle.
  3. Exectution cycle.
  4. Interrupt cycle.

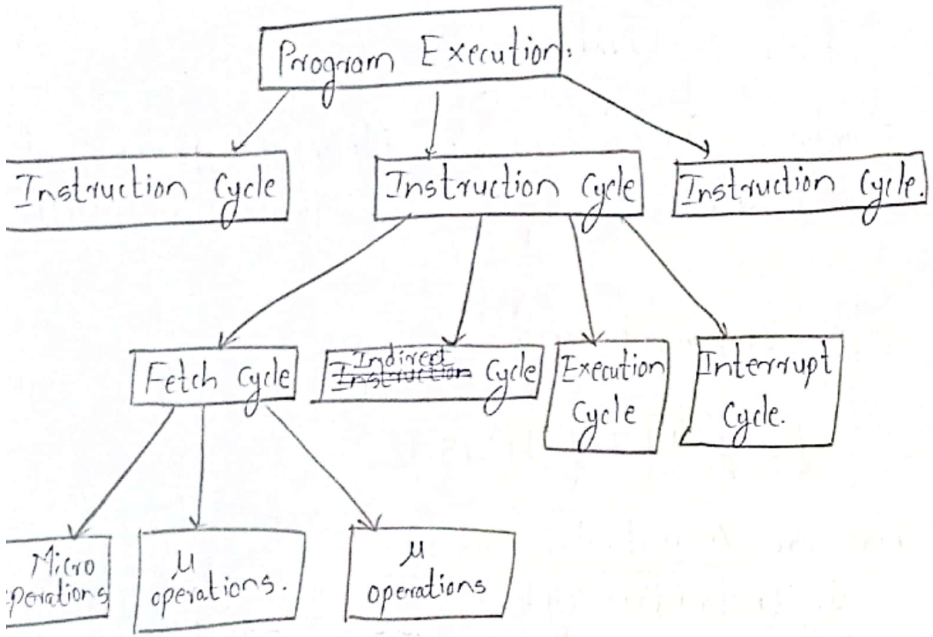
Control Unit :

Data path ← → Control Path.

- Control Signals trigger events in the processor.
- Control Unit generates Control signals.

- Micro-operations :-

It will follow series of instructions.
Micro → simple and will do a small task.



- **Instruction Fetch (IF Cycle):-**

It occurs at the beginning of each Instruction cycle, and causes execution of fetch micro-operations.

4 Registers are involved in Fetch:-
- MAR (Memory Address Register)
- MBR (Memory Buffer Register)
- PC (Program Counter)
- IR (Instruction Register).

◇ Grouping of Micro-operations:-

Can't refer to same memory/register in same cycle.

- **Indirect Instruction:-**

Fetch source operands.

This ~~cycle~~ is used to fetch the values of operands, when using indirect addressing.

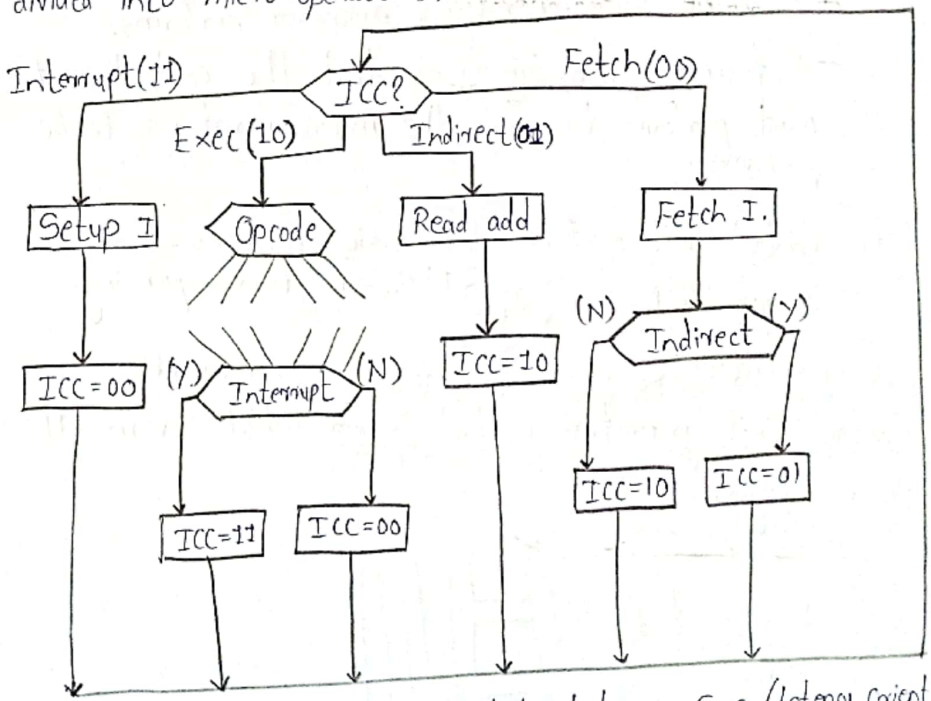
If direct addressing is used, this is skipped.

□ **Interrupt Cycle:-**

The PC is stored in MBR. Interrupt address is loaded in MAR. PC is changed.

ISR is executed.

The original PC is restored. These tasks are also divided into micro-operations.



- Cores ~~are~~ in CPU are called Latency Core. (Latency oriented) They deal with Latency
- Cores in GPU are called Throughput oriented.

CPU
Control Unit ↑
Compute unit ↓

GPU.
Control Unit ↓
Compute Unit ↑

Cuda Core

Tensor (Google's TPU) Core.

Control Unit Functional Requirements:-

Three step process to lead to a characterization of the Control Unit:-

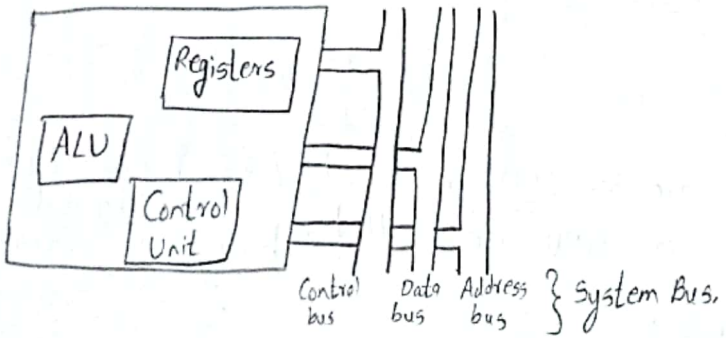
- Define basic elements of processor.
- Describe micro-operations processor performs.
- Determine the functions that the control unit must perform to cause the micro-operations to be performed.

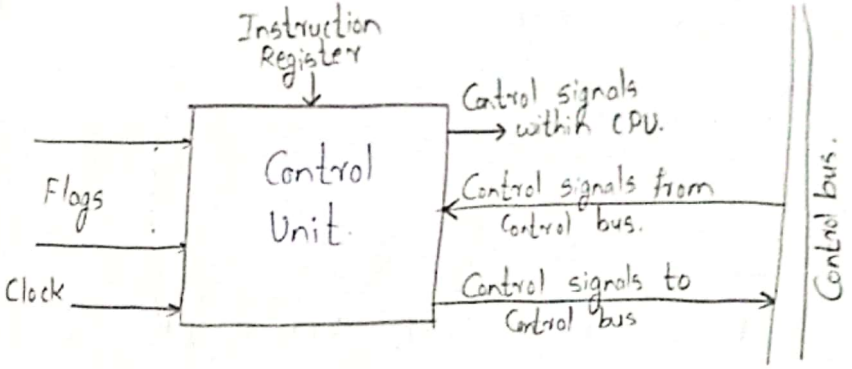
The Control unit performs two basic operations:-

- Sequencing → Micro-operations sequencing.
- execution → Micro-operations execution.

} IMP.

Control Unit generates Control Signals which govern all these things.





Control Signals Governs:-

- Processor (ALU)
- Memory / IO / External things
- Datapath.

Fetch Cycle (From previous):-

PC ⟶ MAR.
Then read from Memory to MBR.
PC ++.
To do this, different control signals are used.

⊛ Control Signals governs the execution of sequence of micro-operations. ⊛

Building a Data path:-

- Elements that process data and addresses in CPU Registers, ALU, Multiplexer, etc.
- We will be using RISC-V

PC = PC+4 ⟶ Because each instruction is of 4 byte, same length RISC Archi.

(L_1D and L_1I Cache is split). (L_2 and L_3 is unified).

- R-Format Instructions:-

Load-Store Instructions:

Branch Instructions:-

- Compare Operands.
  ALU subtracts and check Zero Output.

PERF. (linux utility).
- what it allows to do/perform.
- cache misses are more in column major. (in intel)
  are less in row major
  For Ryzen, it is opposite.

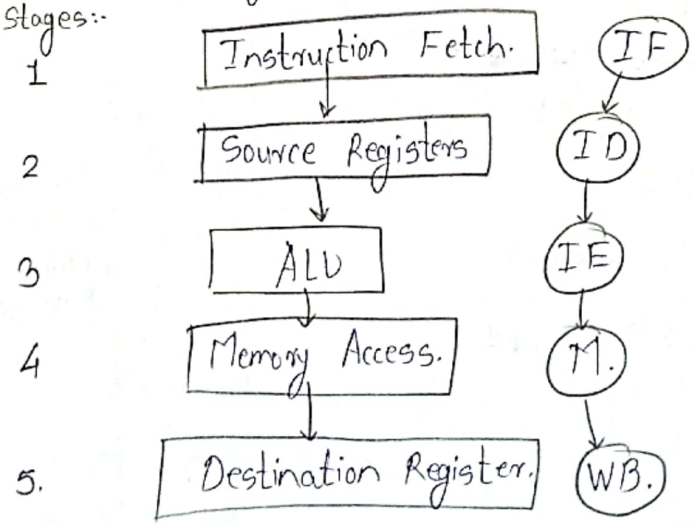
- Full Data Path Diagram:-

- Five-step sequence of actions to fetch and execute an instruction:-

1. Fetch an instruction and increment the program counter.
2. Decode the instruction and read registers from the register file.
3. Perform ALU operation.
4. Read and write memory data if the instruction involves a memory operand.
5. Write the result into the destination register, if needed.

- Basic Processing Unit:-

Stages:-



Five Active temporary resistors, which are always Active.
eg. RA, RB, RM, RY, Rz.

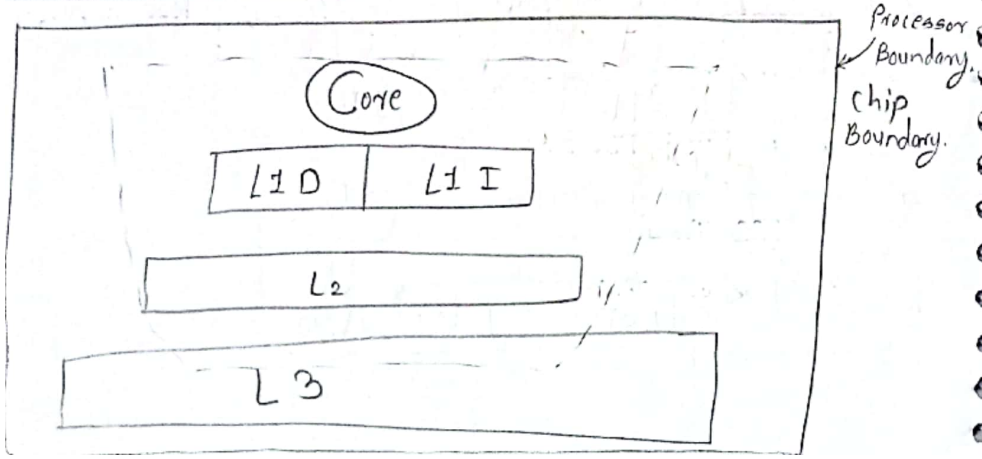
Each Physical Core Supports 2 Hardware threads by Implementation.

L1 Cache is Split into I and D, which is private to each physical Cored.

L2 Cache is private to each physical Core, Unified (not split).

L3 Cache is shared Cache. Unified.

What is block size, and who decides it?



If Cache Miss Occurs at L1, then it will check in TLB, and then will chek in L2 Cache, then in L3 and then the request will go off-chip, to RAM.

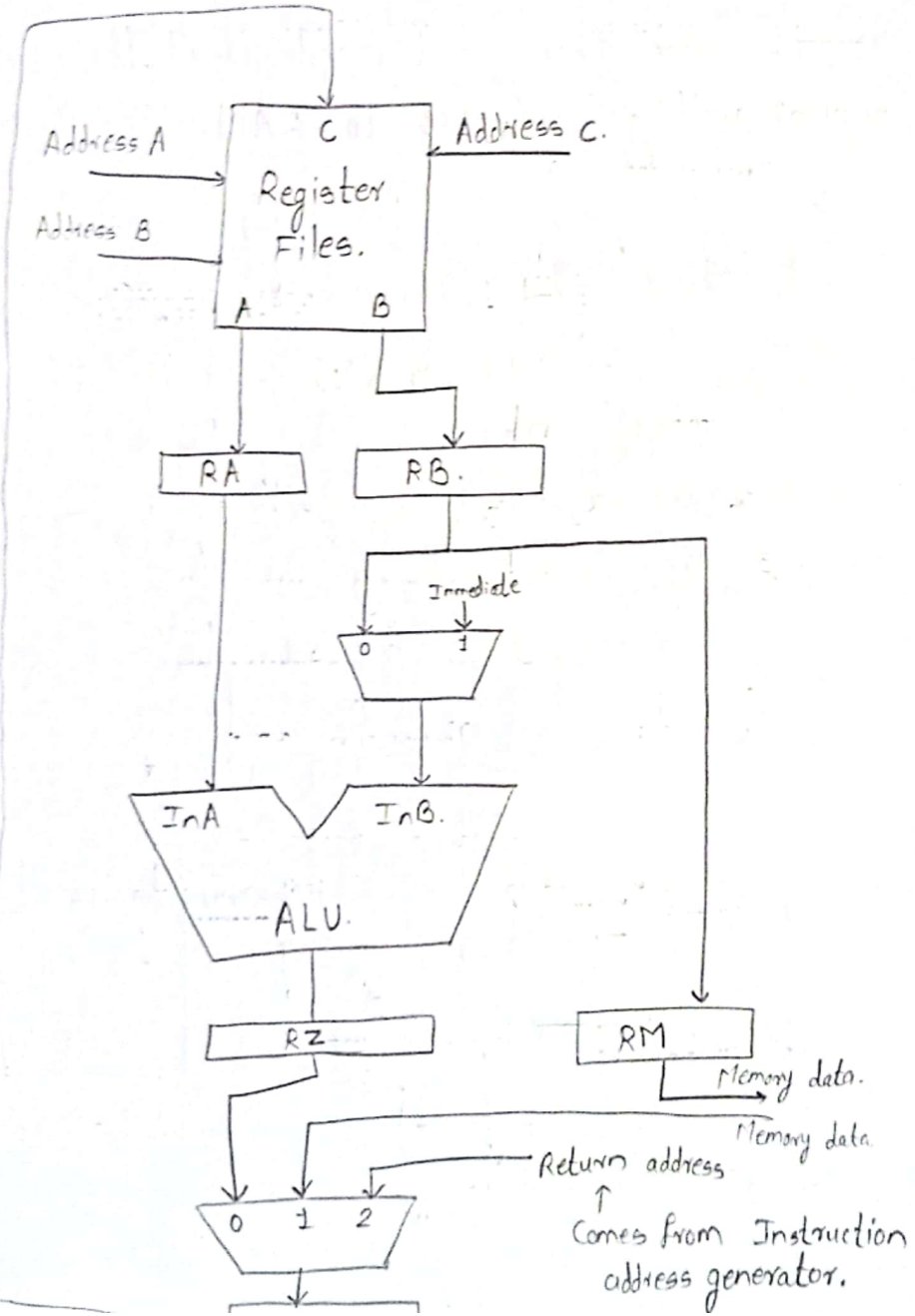
Branch_if_[R5]=[R6] LOOP.

1. Memory address ← [PC], Read Memory, IR ← Memory data.
   PC ← [PC] + 4.
2. Decode Instruction. RA ← [R5], RB ← [R6].
3. Compare [RA] to [RB]. If [RA] = [RB], then
   PC ← [PC] + Branch offset.
4. No Action.
5. No action.

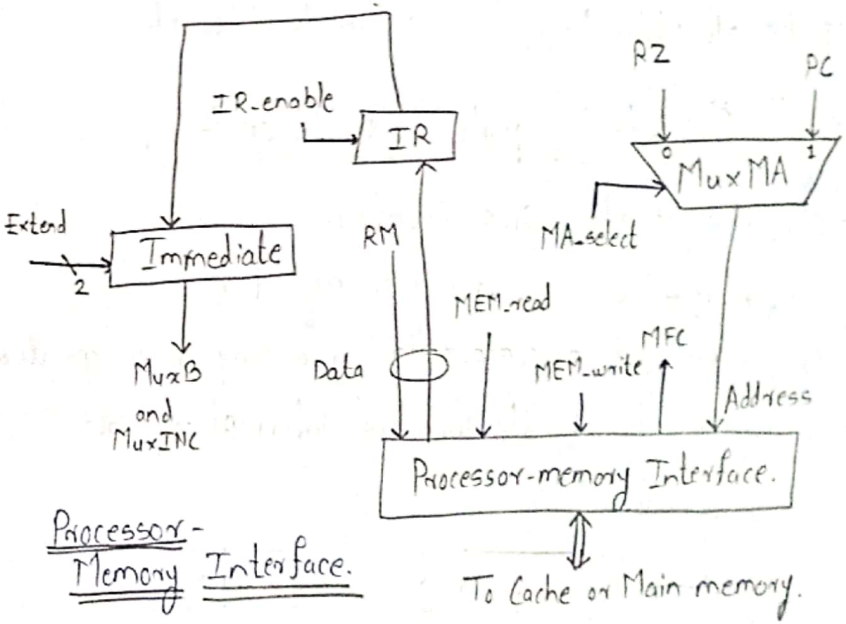
Call_Register Rg.

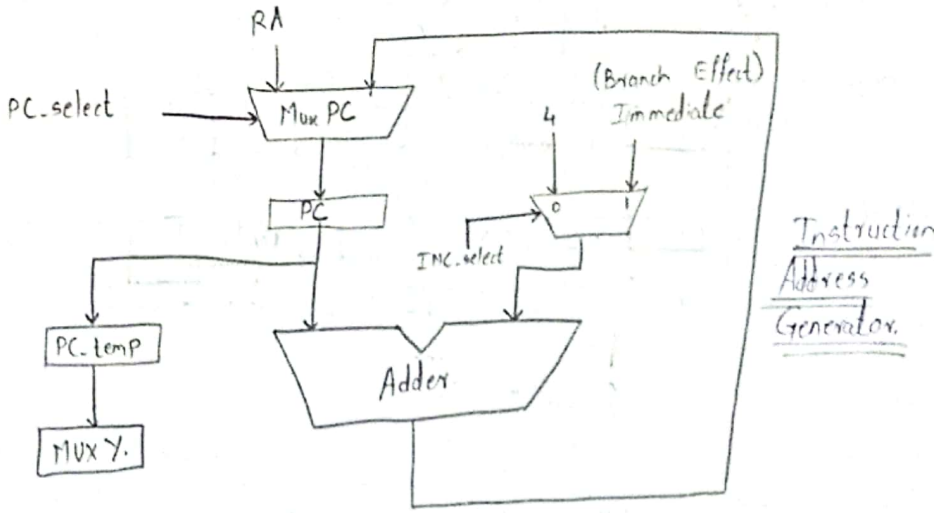
1. Memory address ← [PC], Read Memory. IR ← Memory data, PC ← [PC] + 4.
2. Decode instruction, RA ← [Rg].
3. PC-Temp ← [PC], PC ← [RA]
4. RY ← [PC-Temp].
5. Register Link ← [RY].

MFC — Memory Function Control.



- Processor - Memory Interface
- Control signals for Instruction address generator.

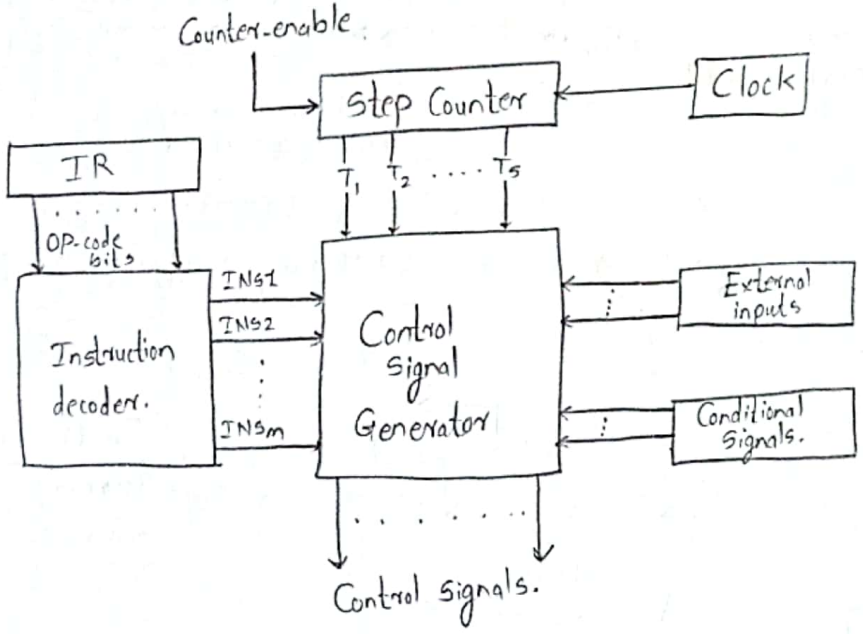




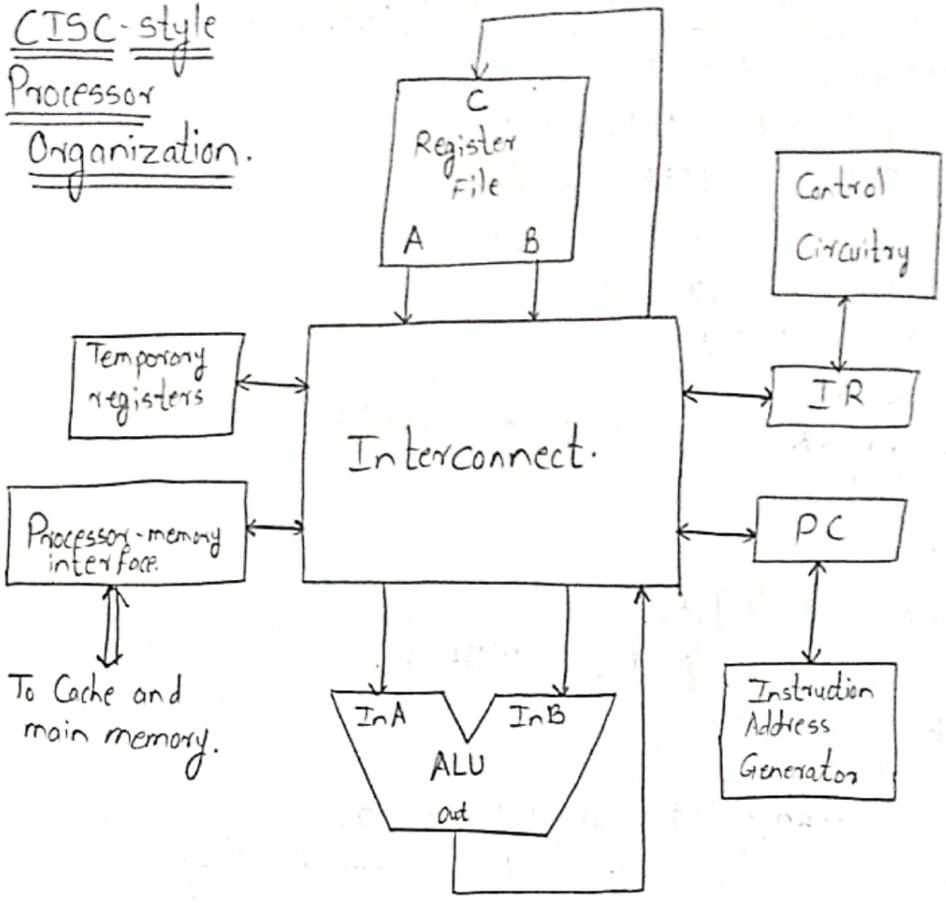
Two Methods to Generate Control Signals:-

Hardwired Control	Micro-program Control.
- Circuit is printed/fixed.	- More Generalized.
- Can't be altered.	- Can be altered.

- **Setting of Control signals depends on:-**
  - Contents of the step Counter.
  - Contents of the instruction register.
  - The result of a computation or a comparison operation.
  - External input signals, such as interrupt requests.



CISC-style Processor Organization.

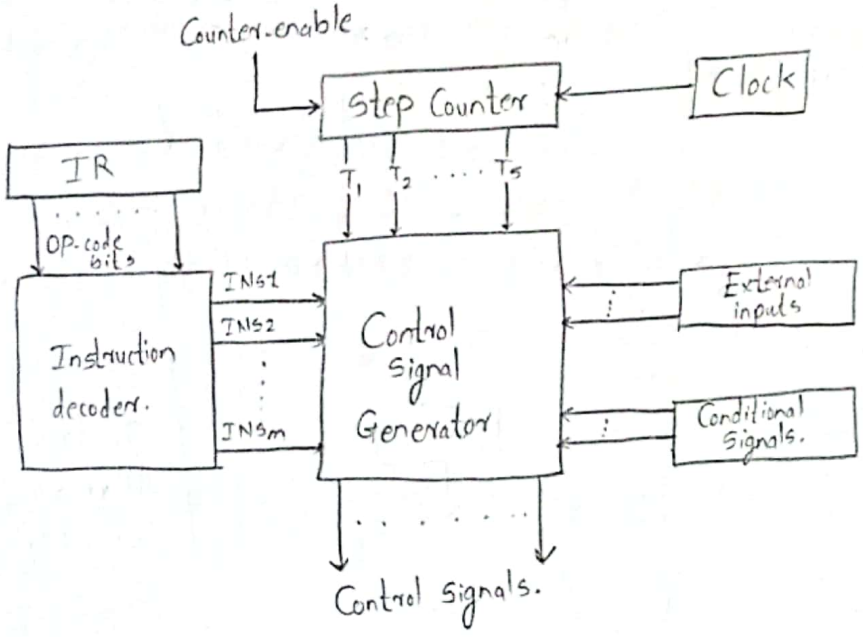


As per instruction in IR, there is a need of getting sequencing of appropriate signals. It can be done by two approaches: (i) Hardware Control.
(ii) Micro-program Control.

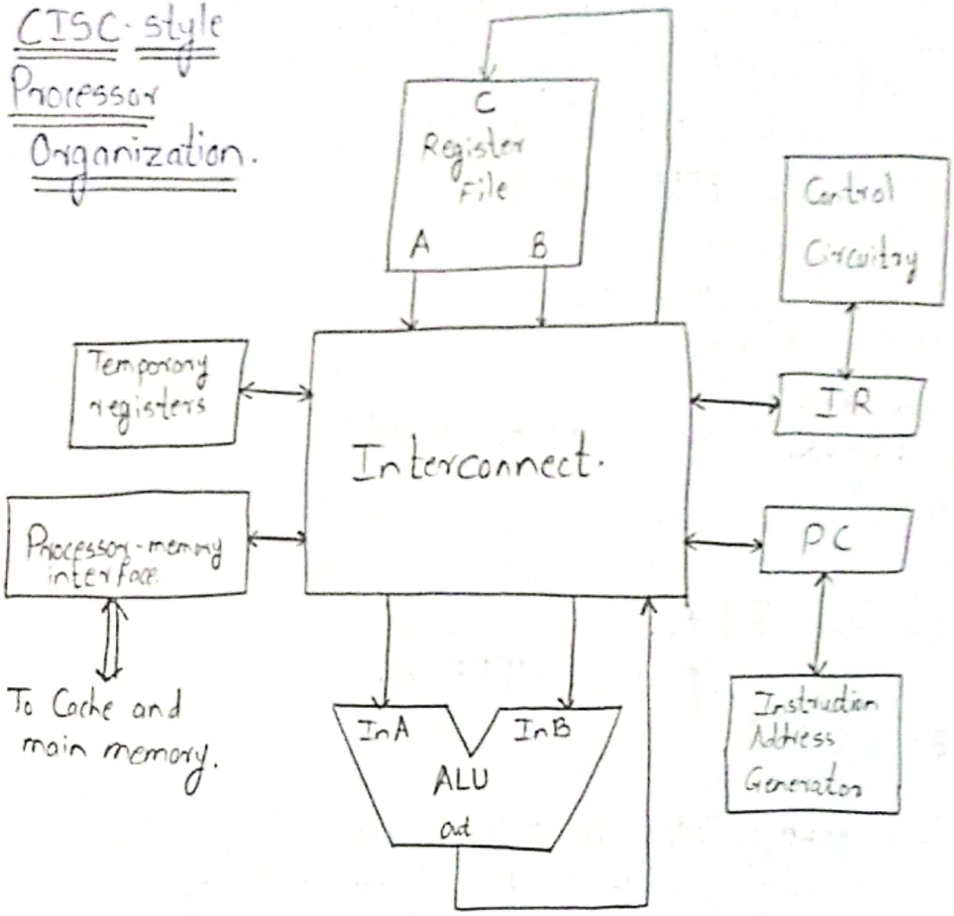
Two Methods to Generate Control Signals:-

Hardwired Control	Micro-program Control.
- Circuit is printed/fixed.	- More Generalized.
- Can't be altered.	- Can be altered.

- **Setting of Control Signals depends on:-**
  - Contents of the Step Counter.
  - Contents of the instruction register.
  - The result of a computation or a comparison operation.
  - External input signals, such as interrupt requests.



CISC-style Processor Organization.



As per instruction in IR, there is a need of getting sequencing of appropriate signals. It can be done by two approaches: (i) Hardware Control.
(ii) Micro-program Control.

Let, us consider, 3 instruction steps:

INSTR 1: Add R_1, R_0.

Steps to Execute: [I] R_1 out, X in.
[II] R_0 out, Add.
[III] Z out, R_1 in
[IV] End.

INSTR 2: XOR R_0R_5

[I] R_0 out, X in
[II] R_5 out, XOR
[III] Z out, R_0 in
[IV] End.

INSTR 3: Add $[R_0], R_6$.

[I] R_0 out, MAR in, MAR out, MBR in.
[II] MBR out, X in.
[III] R_6 out, ADD
[IV] Z out, MBR, in, MBR out, MAR out, RD.
[V] End.

R_0 out = ((INSTR1 AND T2)) OR

When INSTR1 and T2 means R_0 out should be made 1 if there is INSTR1 and second clock cycle.

Add = (INSTR1 and T2) OR (INSTR3 and T3) OR (...)

⋮

end = (INSTR1 and T4) OR (INSTR2 and T4) OR ...

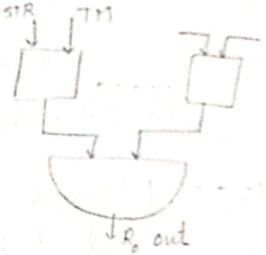
Timing Marker (TM):-

The AND combination is for activation of particular signal in context of signal instruction.

Here, for every instruction, there is one particular gate and giving to each AND gate one combination timing marker and instruction.

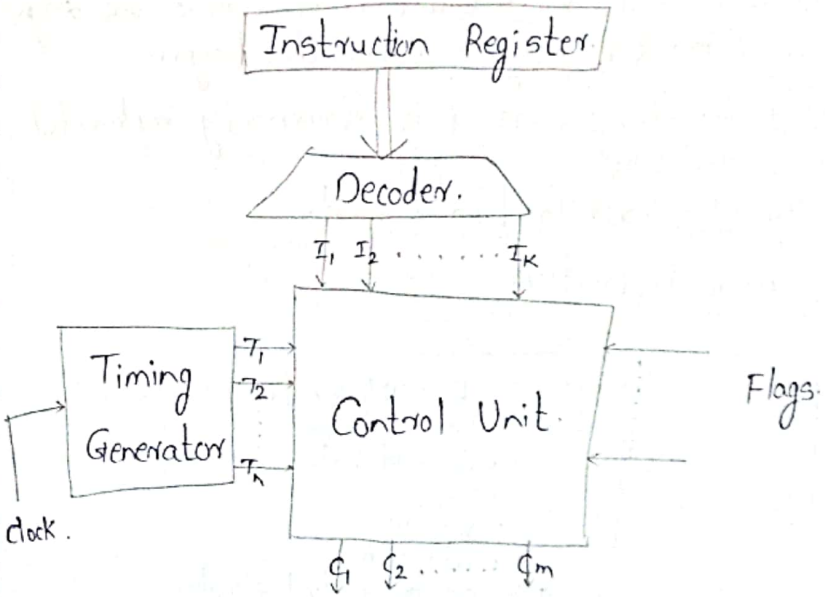
And all the AND gates are ORed.

It gives R_0 out.



We require two types of signal for such an activity:

(i) Time Marker.
(ii) Instruction Marker.



Control Unit with Decoded Inputs:

- Different instruction with different steps to execute means naturally different no. of clock cycles.

So if almost 8 bits instruction, we can choose 3 bit counter.

If 8 steps ⟶ 3 bit counter.
16 steps ⟶ 4 bit counter.

00	T_1
01	T_2
10	T_3
11	T_4.

Like wise, we can have similar pattern for 3 bits, 4 bits, and so on.

End signal connected to n bit counter that indicates 1 instr. is completed and set n bit counter to 0000.

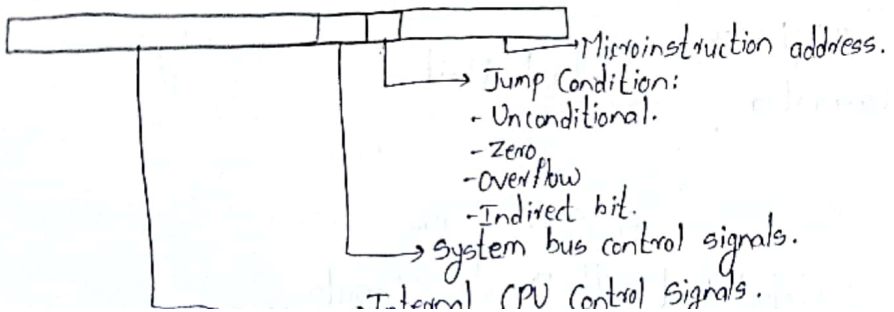
After the prev. step, now IR is loaded with new instr. and counter starts counting the clock again from start.

The approach discussed in hardware based control unit design lasts till we throw the design or scrap the design.

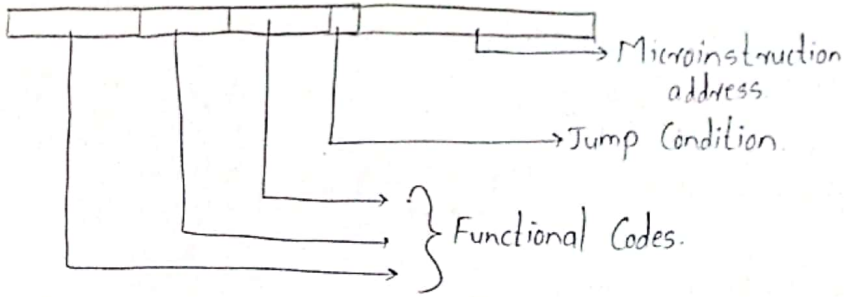
No flexibility for changes as it is permenantly implemented in hardware architecture.

3) Diagram: Typical Microinstruction Formats.

1) Horizontal micro-instruction:



2) Vertical Micro-instruction:



- Control Memory.
- Control Unit Micro-architecture.

Arithmetic for Computers:-

- Integer Addition.
- Integer Subtraction.

ALU (Arithmetic and Logical Unit):-

- Integer Representation.
  - Sign-magnitude Representation. (two 0's).
  - Two's Complement Representation. (range: -2^{n-1} to $2^{n-1}-1$)
    To expand the bits, repeat the signed bit to the left. (only one representation of 0).
    To subtract, A-B, Add two's complement of B to A.

Alternative Representations:-

Decimal	Sign-magnitude	Two's Complement	Biased
+8	–	–	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
-0	1000	–	0110
-1	1001	1111	0101
-2	1010	1110	0100
-3	1011	1101	0011
-4	1100	1100	0010
-5	1101	1011	0001
-6	1110	1010	0000
-7	1111	1001	–
-8	–	1000	–

eg.

-128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

$-128 + 8 = -120$.

New trick.

Range Extension:

- For sign-magnitude,
  → shift the ~~sign~~ sign-bit leftmost, add 0's.
- For two's complement
  → Copy the sign-bit to all the left digits.

- Negation:-
  Negate all bits (1's complement). [Don't change the first 1 from right, change all the leftmost bits].
  then add '1'.

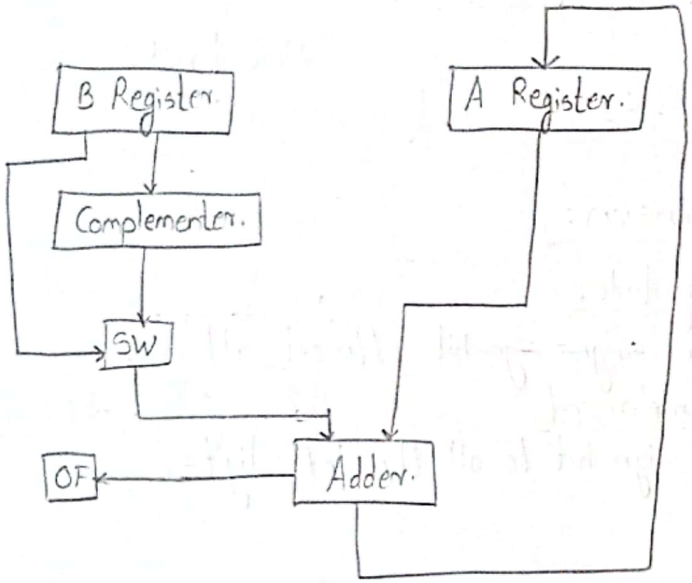
- Special Case:-(i) (-128 for 8 bits).

  $-(-128) \approx -128$.

  (ii) 00000000 → 1's Complement 1111 1111 +1
  1 0000 0000
  ie $-0 \equiv 0$.

Overflow rule:-

If two nos have same sign, and their result has opposite sign, then overflow occurs.

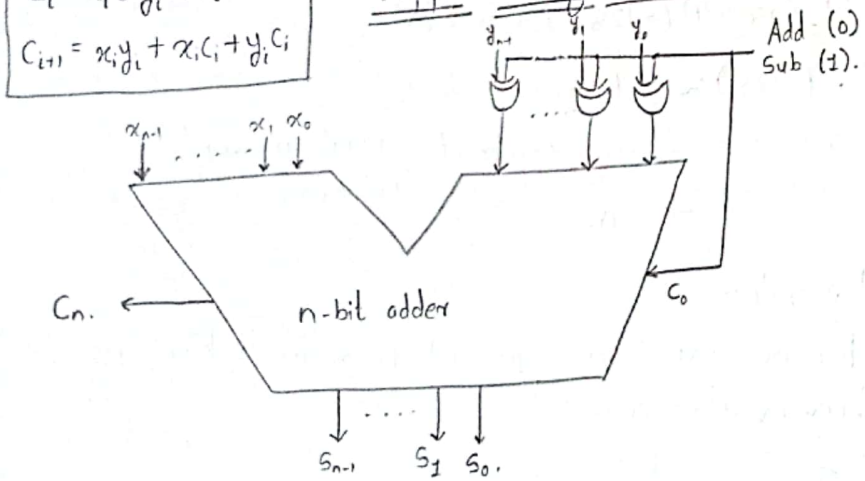


SW = Switch (addition or subtraction)
OF = Overflow bit

$S_i = x_i \oplus y_i \oplus C_i$

$C_{i+1} = x_i y_i + x_i C_i + y_i C_i$

Ripple Carry Adder.



Binary addition / subtraction logic.

Carry look ahead:-

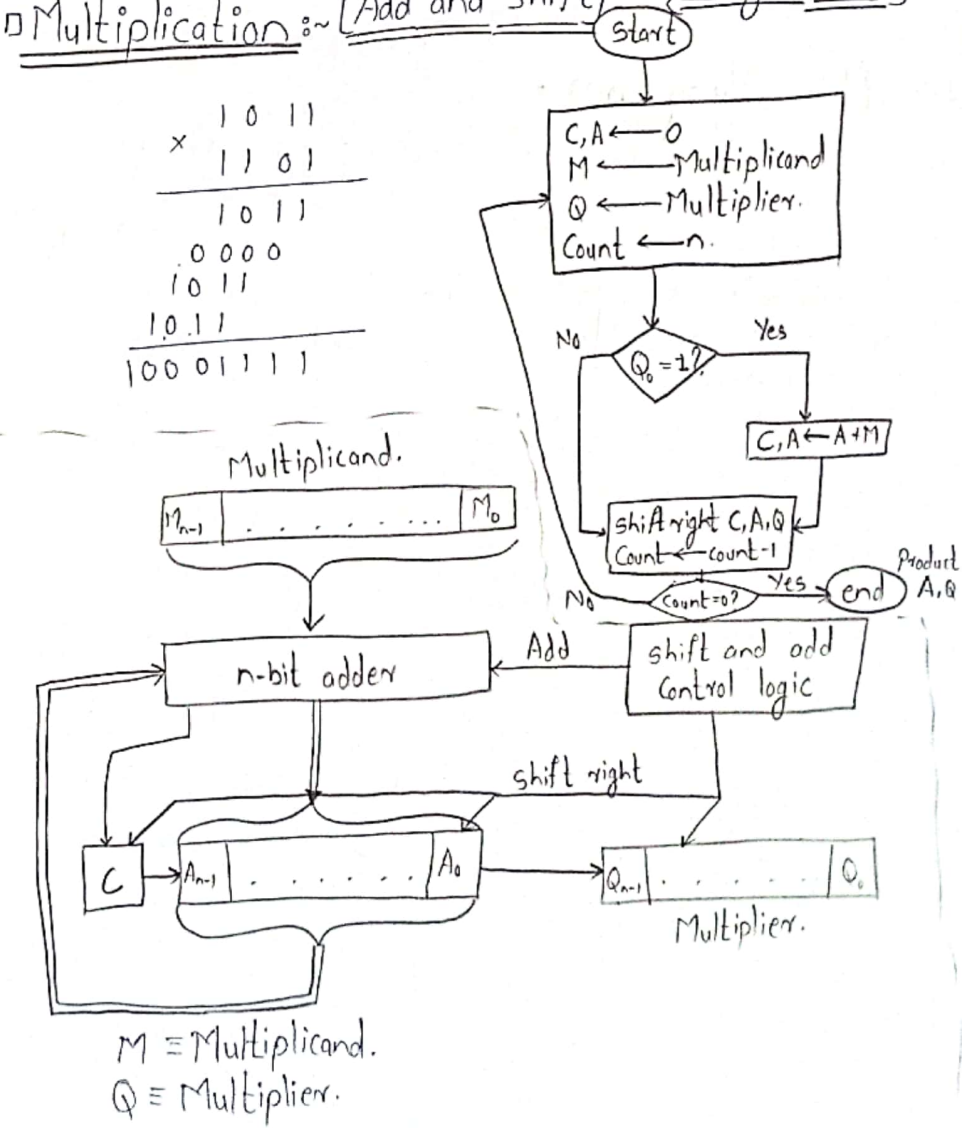
$S_i = (x_i \oplus y_i) \oplus C_i$

$C_{i+1} = x_i y_i + (x_i + y_i) C_i \approx x_i y_i + (x_i \oplus y_i) C_i$

$G_i = x_i y_i \qquad P = x_i \oplus y_i$

Multiplication :- [Add and shift]. {Unsigned int}

```
   1 0 1 1
 x 1 1 0 1
-----------
   1 0 1 1
 0 0 0 0
 1 0 1 1
 1 0 . 1 1
-----------
1 0 0 0 1 1 1 1
```



M ≡ Multiplicand.
Q ≡ Multiplier.

$(-7) \times (+3)$.

$7 = 0111$

$3 = 0011$

$$
\begin{array}{r}
1001 \\
\times\ 0011 \\
\hline
1001 \\
1001 \\
0000 \\
0000 \\
\hline
00011011
\end{array}
$$

$\therefore$ Add and shift will not work for signed integer.

- Booth's Algorithm :-

Division.