COMPUTER ORGANIZATION

Thread :- It is a sequence of instructions that a computer's CPU executes.

Threads are the smallest unit of execution within a process, which can contain one or more threads.

Threads shares resources like memory and computing resources with other threads in the same process which can allow for faster communication and context switching.

Pthreads :- (POSIX threads) : are a standardized multi-thread programming interface for UNIX system

Software Threads are threads of execution managed by the operating system.

Hardware threads are a feature of some processors that allow better utilisation of the Processor under some circumstances.

- Architecture Principles :-

8 Great ideas in computer architecture :-

1. Design for Moore's law
2. Use abstraction to simplify design.
3. Make the common case fast
4. Performance via parallelism
5. Performance via Pipelining
6. Performance via Prediction
7. Hierarchy of memories
8. Dependability via redundancy.

COMPUTER ORGANIZATION

* **Response Time:-** It is the time it takes for a computer to responsd to a user's request or inquiry.

  Response time is frequently used as a measure of the performance of an interactive system.

* **Throughput:-** Throughput is a computer term that measures the volume of data or work that passes through a system or network in a given amount of time.

* Measuring Execution Time
  - Elapsed Time
  - CPU Time.

* **CPU clocking:-**
  Also known as clock rate.
  constant cycle time = τ
  $\therefore$ clock rate $= \frac{1}{\text{cycle time}} = \frac{1}{\tau} = f$

- Size of program determined by its **Instruction Count (IC)**

- Different machine instructions may require different numbers of clock cycles to execute.
  $\therefore$ Cycles per Instruction (CPI) becomes an important parameter for measuring the time needed to execute each instruction.

CPU Time = CPU clock cycles X Clock cycle Time = CPU clock cycles / Clock Rate

Execution Time (ET) = $N \times CPI \times \tau$

$= IC \times CPI \times \tau$

$= \frac{N \times CPI}{f}$

Memory cycle :- Time needed to complete one memory reference.

- CPI example :-

Computer A : cycle Time = 250 ps , CPI = 2.0
Computer B : Cycle Time = 500 ps , CPI = 1.2

Same ISA
which is faster and by how much ?

~~CPU Time A = 250~~ clock cycles

CPU Time A = Instruction count $\times$ CPI_A $\times$ Cycle $Time_A$

$= I \times 2.0 \times 250 = 500 \times I$ / $I \times 500$ ps

CPU Time B = Instruction count $\times$ CPI_B $\times$ Cycle $Time_B$

$= I \times 1.2 \times 500 = 600 \times I$ / $I \times 600$ ps

$\frac{\text{CPU Time B}}{\text{CPU Time A}} = \frac{I \times 600 \text{ ps}}{I \times 500 \text{ ps}} = 1.2$

If different instruction classes take different number of cycles

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

- Weighted average CPI

$$CPI = \frac{\text{clock cycles}}{\text{Instruction count}} = \sum_{i=1}^{n} \left(CPI_i \times \underbrace{\frac{\text{Instruction count}_i}{\text{Instruction count}}}_{\text{relative freq.}} \right)$$

* CPI example.

Alternative compiled code sequences using instructions in 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

Sequence 1: IC = 5

Clock cycles = 2×1 + 1×2 + 2×3
= 10

Avg CPI = $\frac{10}{5}$ = 2

Sequence 2: IC = 6

Clock cycles = 4×1 + 1×2 + 1×3
= 9

∴ Avg CPI = $\frac{9}{6}$ = 1.5

Rainbow

Problem 1.1 : A 400-MHz processor was used to execute a benchmark program with the following instruction mix and clock cycle counts.

Instruction type	Instruction count	Clock cycle count
Integer arithmetic	4 50 000	1
Data Transfer	3 20 000	2
Floating point	1 50 000	2
Control Transfer	80 000	2

Determine the effective CPI, MIPS rate and execution time for this program.

① Avg CPI = $\frac{4,50,000\times1 + 3,20,000\times2 + 1,50,000\times2 + 80,000\times2}{1000000}$

= 1.55

② MIPS rate = $\frac{400\times10^6}{1.55\times10^6}$ = 258.06 MIPS

③ Execution Time = $\frac{IC \times CPI}{f}$

= $\frac{10^6 \times 1.55}{400\times10^6}$ = 3.875 ms

Problem 1.4 : Consider the execution of an object code with 2×10^6 instructions on a 400-MHz processor. The program consists of four major, the program consists of four major type of instructions. The instruction mix and the number of cycles (CPI) needed for each instruction type are given below based on the result of a program trace experiment.

Rainbow

Instruction type	CPI	Instruction mix
Arithmetic and logic	1	60%
load/store with cache hit	2	18%
Branch	4	12%
Memory Reference with cache miss	8	10%

① Avg CPI = $1 \times 0.6 + 2 \times 0.18 + 4 \times 0.12 + 8 \times 0.1$
$= 2.24$

② MIPS rate = $\frac{f}{CPI \times 10^6} = \frac{400 \times 10^6}{2.24 \times 10^6} = 178.57$ MIPS

* MIPS rate :-

$= \frac{IC}{T \times 10^6} = \frac{f}{CPI \times 10^6} = \frac{f \times IC}{C \times 10^6}$

Problem 1.6 : The execution time (in seconds) of four programs on three computers are given below:
Assume that 10^9 instructions were executed in each of the four programs. Calculate the MIPS rating of each program on each of the three machines. Based on these ratings, can you draw a clear conclusion regarding the relative performance of the three computers?

Program	Execution Time (in seconds)		
	Computer A	Computer B	Computer C
Program 1	1	10	20
Program 2	1000	100	20
Program 3	500	1000	50
Program 4	100	600	100

Benchmarks :-

- kernels (e.g. matrix multiply)
- Toy programs (e.g. sorting)
- Synthetic Benchmark (Whrystone)
- Bench mark suites (e.g. SPEC 06 Fp, TPC-C)

- Amdahl's law:-

$$\text{Execution time}_{new} = \text{Execution time}_{old} \times \left(1 - \text{fraction}_{enhance} + \frac{\text{Fraction}_{enhance}}{\text{speedup}_{enhance}}\right)$$

$$\text{Speedup Overall} = \frac{\text{Execution time}_{old}}{\text{Execution time}_{new}} = \frac{1}{\left(1 - \text{fraction}_{enhanced}\right) + \left(\frac{\text{Fraction}_{enhanced}}{\text{speedup}_{enhanced}}\right)}$$

$$S_o = \frac{1}{(1-F) + \frac{f}{s}} \qquad F < 1 \qquad S > 1$$

Rainbow

- Amdahl's law

fraction enhanced : The fraction of the computation time in the original computer that can be converted to take advantage of enhancement.

$f < 1$ always

Speedup enhanced : The improvement gained by the enhanced execution mode, that is, how much faster the task would run if the enhanced mode were

$S > 1$ always.

$$\therefore \text{Speedup}_{\text{overall}} = \frac{1}{(1-f) + \frac{f}{S}}$$

Problem 1:

Suppose that we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement.

$$S_o = \frac{1}{(1-f) + f/s} = \frac{1}{(1-0.4) + \frac{0.4}{10}}$$

$$= \frac{10}{0.6 \times 10 + 0.4}$$

$$= \frac{10}{6.06 + 0.4} \approx 1.56$$

$$= \frac{10}{6.4} \approx 1.5625$$

ainbow

Problem 2 :

A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal to enhance the FPSQR hardware and speedup this operation by a factor of 10. The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Compare these two design alternatives.

given: FP case : $S = 1.6$, $f = 0.5$

$$\therefore S_o = \frac{1}{(1-0.5)+\frac{0.5}{1.6}} = \frac{1}{0.5\times1.6+0.5} = \frac{1.6}{0.8+0.5}$$

$$= \frac{1.6}{1.3} = \underline{\underline{1.23}}$$

FPSQR : $f = 0.2$, $S = 10$

$$\therefore S_o = \frac{1}{(1-0.2)+\frac{0.2}{10}} = \frac{10}{0.8\times10+0.2} = \frac{10}{8.2} = 1.2195$$

S_o of FP > S_o of FPSQR $\therefore$ second design is preferable.

Problem 3 :

Suppose we have made the following measurements:

Frequency of FP operations = 25%

Avg. CPI of FP operations = 4.0

Avg. CPI of other instructions = 1.33

Frequency of FPSQR = 2%

CPI of FPSQR = 20

Assume that the two design alternatives are to decrease the CPI of FPSQR to 2 or to decrease the average CPI of all FP operations to 2.5. Compare these two design alternatives using the processor performance equation.

$ET = N * CPI * \tau$

$= \frac{N * CPI}{f}$

FP operations :-

$CPI\ original = \sum_{i=1}^{n} CPI_i \times \left[\frac{IE}{Instruction\ count}\right]$

$= (4 \times 25\%) + (1.33 \times 75\%)$

$= 1.9975 \approx 2.0$

CPI with new FPSQR = CPI original − 2% × (CPI old FPSQR − CPI of new FPSQR only)

$= 2.0 - 2\% \times (20 - 2) = 1.64$

CPI new FP = (75% × 1.33) + (25% + 2.5) = 1.625

Speedup new FP = $\frac{CPU\ Time\ Original}{CPU\ Time\ New\ FP}$

$= \frac{IC \times Clock\ cycle \times CPI\ original}{IC \times clock\ cycle \times CPI\ new\ FP}$

$= \frac{CPI\ Original}{CPI\ new\ FP}$

$= \frac{2.0}{1.625} = 1.23$

Problem 4 :

Assume that we make an enhancement to a computer that improves some mode of execution by a factor of 10. Enhanced mode is used 50% of the time, measured as a percentage of the execution time when the enhanced mode is in use. Recall that Amdahl's law depends on the fraction of the original, unenhanced execution time that could make use of enhanced mode. Thus we cannot directly use this 50% measurement to compute speedup with Amdahl's law

(a) What percentage is the speed up we have obtained from fast mode?

(b) What percentage of the original execution time has been converted to fast mode?

Rainbow

(a) Old ET = 0.5new + 0.5 x 10 new = 5.5new

(b) In orginal code, unenhanced part = enhanced part speedup by 10
$(1-x) = \frac{x}{10}$ = $10 - 10x = x$ = $11x = 10$
$\therefore x = 10/11$

Problem 5:

Your company has just bought a new Intel Core i5 dual-core processor, and you have been tasked with optimizing your software for this processor. You will run two applications on this dual core, but the resource requirements are not equal. The first application requires 80% of the resources, and the other only 20% of the resources. Assume that when you parallelize a portion of the program, the speedup for that portion is 2.

(a) Given that 40% of the first application is parallelizable, how much speedup would you achieve with that application if run in isolation?

(b) Given that 99% of the second application is parallelizable, how much speedup would this application observe if run in isolation?

(c) Given that 40% of the first application is parallelizable, how much overall system speedup would you observe if you parallelized it?

(d) Given that 99% of the second application is parallelizable, how much overall system speedup would you observe if you parallelized it?

(a) $S_o = \frac{1}{(1-0.4) + 0.4/2} = \frac{1}{0.8} = 1.25$

(b) $S_o = \frac{1}{(1-0.99) + \frac{0.99}{2}} = 1.98$

(c) $S_o = \frac{1}{0.8 + 0.2 \times 0.01 + 0.2 \times \frac{0.9}{2}} = 1.11$

Problem 6 :

When parallelizing an application, the ideal speedup is speeding up by the number of processors. This is limited by two things: percentage of the application that can be parallelized and the cost of communication.

Amdahl's law takes into account the former but not the latter.

(a) ~~What~~

(b)

(c)

→ (a) $S_o = \frac{1}{(1-F) + \frac{F}{N}} = \frac{1}{(1-0.8) + \frac{0.8}{N}}$

(b) $S_o = \frac{1}{0.2 + (8 \times 0.005) + \frac{0.8}{8}} =$

(c) $S_o = \frac{1}{0.2 + (\log_2 8 \times 0.005) + \frac{0.8}{8}} = \frac{1}{0.2 + (3 \times 0.005) + \frac{0.8}{8}} =$

(d) $S_o = \frac{1}{0.2 + (\log_2 N \times 0.005) + \frac{0.8}{N}}$

Rain

④ $\left(\frac{1}{(1-0.0P)+(\log_2 N \times 0.005)+\frac{0.0P}{N}}\right)' = 0$

$\therefore \frac{d}{dN}\left(\frac{1}{(1-0.0P)+(\log_2 N \times 0.005+0.0P/N)}\right) = 0$

Problem 7:

Consider the unpipelined processor in the previous section. Assume that it has a 1ns clock cycle and that it uses 4 cycles for ALU operations and branches and 5 cycles for memory operations. Assume that the relative frequencies of these operations are 40%, 20% & 40% respectively. Suppose that due to clock skew and setup, pipelining the processor adds 0.2 ns of overhead to the clock. Ignoring any latency impact, how much speedup in the instruction execution rate will we gain from a pipeline?

→ Average instruction execution time = Clock cycle x Average CPI
= 1ns x [(40% + 20%) x 4 + 40% x 5]
= 4.4 ns.

Speedup from pipelining = Average instruction time unpipelined / Average instruction time pipelined
= 4.4 ns / 1.2 ns = 3.7 times

Problem 8:

We begin with a computer implemented in single-cycle implementation. When the stages are split by functionality, the stages do not require exactly the same amount of time. The original machine had a clock cycle time of 7ns. After the stages were split, the measured times were IF, 1ns; ID, 1.5ns; EX, 1ns; MEM, 2ns; & WB, 1.5ns. The pipeline register delay is 0.1 ns.

⑤

(b)

(c)

(d)

(e)

→ 2ns + 0.1ns = 2.1ns.

(b)

→ 5 cycles / 4 instructions = 1.25

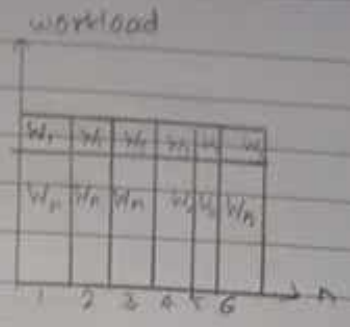
(f)

→ $\frac{I \times CPI \times t}{I \times CPI \times t} = \frac{4 \times 1.25 \times t}{4 \times} \quad \frac{1 \times 7}{1.25 \times 2.1} = 2.67$

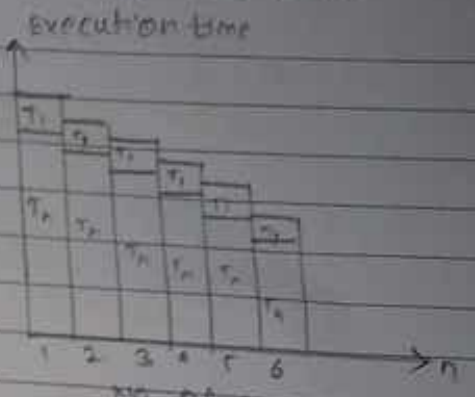
(g)

→ $\frac{I \times 1 \times 4}{I \times 1 \times 0.1} = 40$

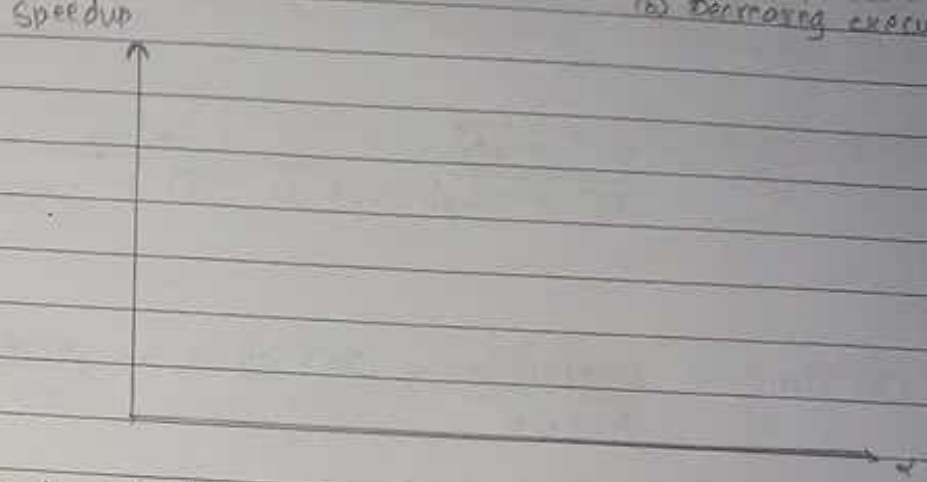
Amdahl's law :- $S_n = \frac{n}{1 + \alpha(n-1)}$ implication is that $S \to \frac{1}{\alpha}$ as $n \to \infty$



No. of processors (a) Fixed workload

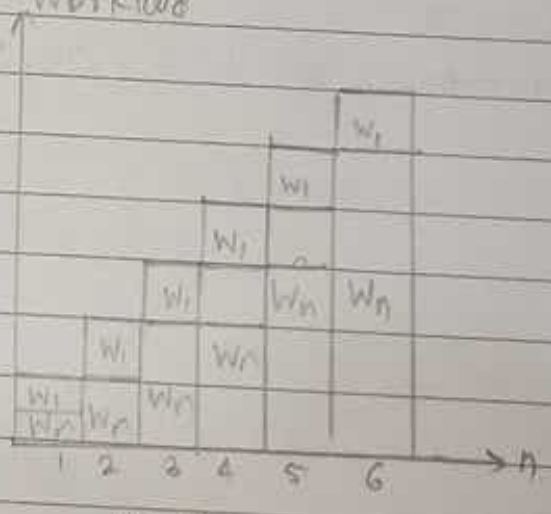


No of processors (b) Decreasing execution time

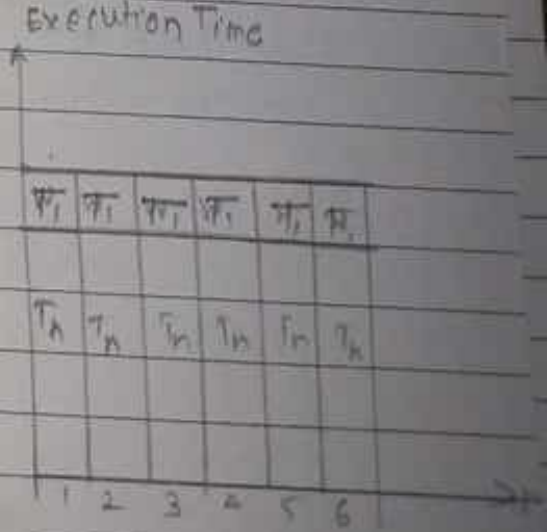


α is called as sequential bottleneck in a program.

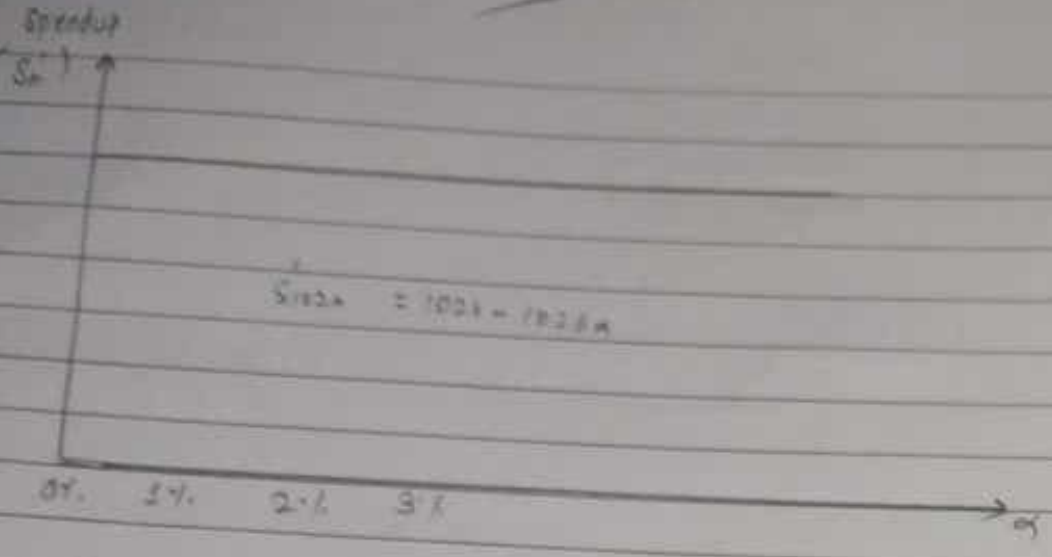
Gustafson's law :-



No. of processors (a) Scaled workload



No. of processors (b) Fixed execution time.



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

* CPU Time = $\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$

- Performance depends on
  - Algorithm : affects IC, possibly CPI
  - Programming language : affects IC, CPI
  - Compiler : affects IC, CPI
  - Instruction set architecture : affects IC, CPI, T_c.

Programs to Execute $Y = \frac{A - B}{C + (D \times E)}$

(a) Three-address instructions

Instruction		Comment	
SUB	Y, A, B	Y ← A-B	4
MPY	T, D, E	T ← D×E	
ADD	T, T, C	T ← T+C	
DIV	Y, Y, T	Y ← Y÷T	

(b) Two address instructions

Instruction		Comment	
MOVE	Y, A	Y ← A	6
SUB	Y, B	Y ← Y-B	
MOVE	T, D	T ← D	
MPY	T, E	T ← T×E	
ADD	T, C	T ← T+C	
DIV	Y, T	Y ← Y÷T	

(c) One-address instructions:

LOAD D	AC ← D	8
MPY E	AC ← AC×E	
ADD C	AC ← AC+C	
STOR Y	Y ← AC	
LOAD A	AC ← A	
SUB B	AC ← AC-B	
DIV Y	AC ← AC÷Y	
STOR Y	Y ← AC	

Now, $Z = K + B * C$

Write all for this

Two address instruction:

MOV R1, C

MUL R1, B

ADD R1, K

MOV Z, R1

One address instruction.

LOAD C		4
MUL B		
ADD K		
STR Z		

Three address instructions

OP d, i, operands

Two address instructions

OP d, S

One address instructions

OP S

three address instruction.

MPY R1, B, C } 2

ADD Z, K, R1 }

zero address instruction (use stack)

Z = K + B * C → post-fix → KBC*+

```
PUSH K
PUSH B
PUSH C
MUL
ADD
POP Z
```

6

- zero-address instruction:-

Postfix expression of $Y = \frac{A-B}{C+(D*E)}$

i.e. Y = (A - B) / C + (D * E)

AB - CDE * + /

```
PUSH A
PUSH B
SUB
PUSH C
PUSH D
PUSH E
MUL
ADD
DIV
POP Y
```

10

Eg.

① Z = P - Q Z = (((P - (Q + S) + T) / (U / (V - W))))

Three address instruction:-

```
ADD  R1, Q, S     R1 ← Q + S
MUL  R2, R1, T    R2 ← R1 * T
SUB  R3, P, R2    R3 ← P - R2
SUB  R4, V, W     R4 ← V - W
DIV  R5, U, R4    R5 ← U / R4
DIV  R6, R3, R5   R6 ← R3 / R5
```

6

Two-address instructions.

MOVE, R_1, Q	$R_1 \leftarrow Q$
ADD, R_1, S	$R_1 \leftarrow R_1 + S$
MUL, R_1, T	$R_1 \leftarrow R_1 * T$
MOVE R_2, P	$R_2 \leftarrow P$
SUB R_2, R_1	$R_2 \leftarrow R_2 - R_1$
~~SUB~~	
MOVE R_3, V	$R_3 \leftarrow V$
SUB R_3, W	$R_3 \leftarrow R_3 - W$
~~DIV~~	
MOVE R_4, U	$R_4 \leftarrow U$
DIV R_4, R_3	$R_4 \leftarrow R_4 / R_3$
DIV R_2, R_4	$R_1 \leftarrow R_2 / R_4$

(something might be wrong) it should be 12

One-address instructions.

LOAD Q	AC ← Q
ADD S	AC ← AC + S
~~MUT T~~	
STOR T	~~LOAD Q~~
LD P	ADD S
SUB T	MUL T
MUL T	STOR Temp
LD V	~~LOAD P~~
SUB W	~~SUB Temp~~
STOR T1	STOR Temp
LD U	LOD V
DIV T1	SUB W
STOR T1	STOR Temp1
LD T	LOAD V
DIV T1	DIV Temp1
STOR Z	STOR Temp2
	LOAD Temp
	DIV Temp2
	STOR Z

inbow

zero address instruction → 14
postfix : QS+T*P-

* Expanded Opcode Technique:-

I have set of 16R with, and i'm represent each instruction with 16 bit, then 14 no of type 3 address instruction, 30 of type 2 address instruction, 28 of type 1 address instructions, 64 of type 0 addres instruction.

type 3 :

4 X 3 = 12 bits for operands.	0000 ----, ----, ---- ^{01} ^{02} ^{03}
4 bits are for opcodes	1101 ----, ----, ----

14×2^{12}

for 2 address instructions.

8 bits for opcodes & 8 bits for oprands.

In oprode first 3 bits are mask type & remaining for oprode.

range 111 | 00000 ----, ----

111 | 11101 ----, ---- 30×2^8.

escape code

for type 1 address instructions.

escape code 1111111100000000

1111111 01 (Binary of 27) 28×2^4

for zero address instruction.

all 16 bit represents oprode.

1111111111000000 64/£

1111111111011111

∴ By solving this we can calculate total

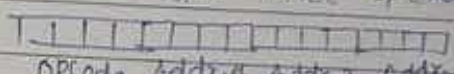
$14 \times 2^{12} + 30 \times 2^8 + 28 \times 2^4 + 1 \times 2^6 =$

Example 1: Consider a machine with 16-bit instructions and 16 registers.

The instruction format can have several structures:

- opcode + Memory address (such as MARIE):
  - → If we have 4kB byte addressable memory we need 12 bits to specify an address location.
  - → The remaining 4 bits are used for the opcode: 16 instructions are hence available

- Oprode + Registers Addresses
  - → We need 4 bits to select one of the 16 available registers
  - → Suppose we have 4 bits oprode, we could encode 16 different instructions with three operands each (3x 4 bits = 12 bits)



Example 2: Consider a machine with 16-bit instructions & 16 registers. And we wish to encode the following instructions.

- 15 instructions with 3 addresses
- 14 instructions with 2 addresses
- 31 instructions with 1 address
- 16 instructions with 0 addresses.

Can we encode this instruction set in 16 bits?

⟹ Yes, if we use expanding opcodes

→ 15 instructions with 3 addresses.

0000, ----, ----, ----

:

1110, ----, ----, ----

14 instructions with 2 addresses.

1111 0000 ----, ----

:

1111 1101 ----, ----

31 instructions with 1 address

1111 1110 0000 ----

:

1111 1111 1110 ----

16 instructions with 0 addr

1111 1111 1111 0000

:

1111 1111 1111 1111

$15 \times 2^{12} + 14 \times 2^{8} + 28 \times 2^{4} + 64 = 65536$

$15 \times 2^4 \times 2^4 \times 2^4 = 61440$ bit patterns 4 — 3 instruction addr.
$14 \times 2^4 \times 2^4 = 3584$ bit patterns — 2 instruction addr.
$31 \times 2^4 = 496$ bit patterns — 1 instruction addr.
$16 \times 2^0 = 16$

add all = 65536

Example: 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.

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

$4 \times 2^3 \times 2^3 \times 2^3 + 255 \times 2^3 + 16 \times 2^0 = 4104$

we have 4096.
new requirement is 4104

Q. Assume a CPU having 16 registers and instruction length is 16 bits, it should be having
14 of type 3 addr. instruction
31 of type 2 addr. instruction
12 of type 1 addr. instruction
calculate, at the most how many no. of type 0 address instruction have.
Show appropriate encoding

$14 \times 2^4 \times 2^4 \times 2^4 + 31 \times 2^4 \times 2^4 + 12 \times 2^4 + x = 65536$

∴ 64 0 address instructions

0000, ----, ----, ----
⋮
1101, ----, ----, ----

1111 0000, ----, ----
1111 1110, ----, ----

1111 1110 0000, ----
1111 1111 1110, ----

1111 1111 1111 0000
:
1111 1111 1111 1111

Example 4:
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: two 15 bit addr & 1 3bit reg
500 instructions one 15 bit addr & 1 3 bit reg
50 instructions with no addresses or regs.

→ 7 × 2
the first 7 instructions account for
~~2×~~ $7 \times 2^{15} \times 2^{15} \times 2^3 = 7 \times 2^{33} = 6.013 \times 10^{10}$

the next 500 instructions
$500 \times 2^{15} \times 2^3 = 1.31 \times 10^8$

The next 50 instructions = 50 bit patterns
~~$50 \times 2^{15} \times 2^3$~~
In total we need 6.026×10^{10} bit patterns.
With 36 bit instructions we can have $2^{36} = 6.87 \times 10^{10}$ bit patterns.

∴ We have enough bit patterns so can create opcodes

Rainbow

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

(a) Three no. of instructions of type that have 2 register fields and one memory address.

→ there are two registers each of 3 bit = 2X3 = 6 bits.
and one memory address = 8 ∴ 8+6 = 14 bits
2 are remaining, we need 2 ∴ 00 ----------
10 ----------

It is possible.

(b) six no. of instructions of type that have 1 register field and one memory address.

→ one register field contains = 3 bits } total = 11 bits
one memory address contains = 8 bits }
now out of 16 bits, 5 bits are remaining because 11 bits are gone.
∴ 11 000 ----------
11 101 ----------

It is possible

(c) six no. of instructions of type that have all 3 register field.

→ one register field contain = 3 bits = 3X3 = 9 bits.
now remaining are 7 bits
It requires six
escape code
∴ 1111 | 000 | ----------
:
1111 | 101 | ----------

It is possible

(d) Three no. of instructions that have single memory address field.

∴ one memory address = 8 bits.
∴ 11111 | 000 | ----------
11111 | 101 | ----------
∴ It is possible.

Rai

(e) seven no. of instructions of type that have 2 register fields

→ for registers ⇒ 2x3 = 6bits
10 are remaining.

1111111 000 - - - - - - -
:
1111111 110 - - - - - - -

If we use 8 bits code as escape then it is not possible

(f) seven no. of instructions of type that have only one register field

∴ one registers = 3 bits
remaining = 13 bits.

1111 111 1 1111000 - - -
:
1111111111 910 - - -

It is possible.

(g) eight no. of instructions of type that have 0 address.

1111111111111000
:
1111111111111111

Pipeline Implementation | IF | ID | IE | M | WB |

There are four particular subcycles :

① Instruction cycle
② Indirect cycle
③ Execution cycle
④ Interrupt cycle

control

Micro-operation: That are the Functional it will follow it is a very simple and it will complete very small or little amount of task.

Fetch cycle:- The fetch cycle, it occurs at beginning of the instruction cycle and causes to be instruction fetch from memory

Indirect cycle:- Once the instruction is fetch, then next step is to fetch source operands

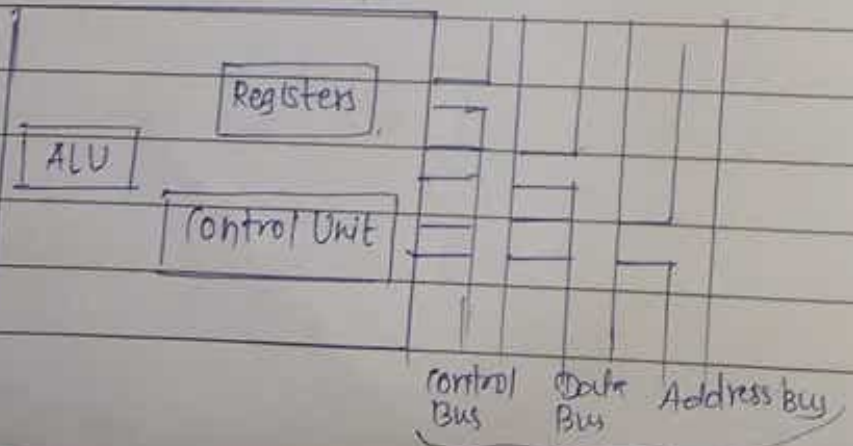
Interrupt cycle:

Control Unit Functional Requirements :

~~By reducing the operation~~

Task of control unit → to generate control signals.

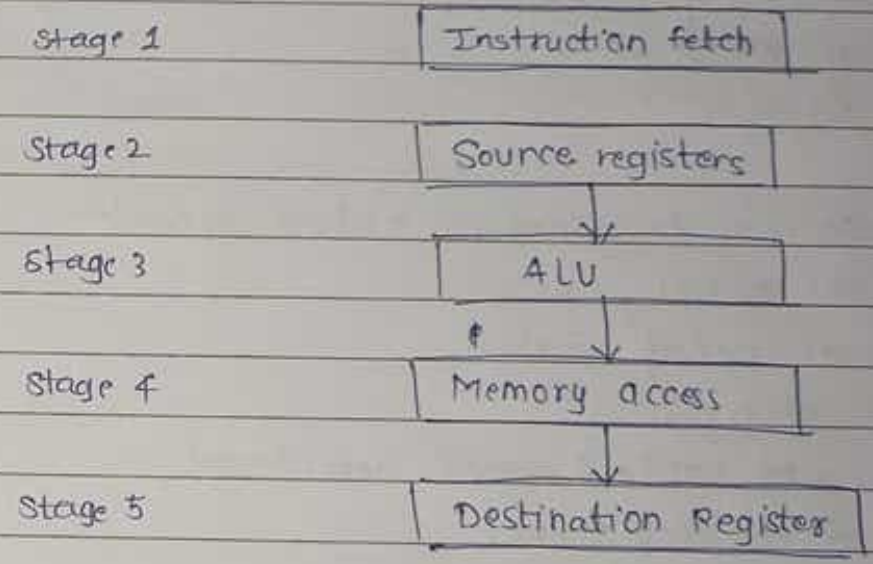
Diagram:- The CPU with System Bus



- The control unit performs two basic tasks
  (a) Sequention
  (b) Execution

A five step sequence of actions to fetch and execute an instruction.

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



Branch_if_[R5]=[R6] LOOP

Sequence of actions needed to fetch and execute the instruction Branch_if_[R5]=[R6] LOOP

Step	Action.
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.

Two approaches to generate control signals:-

- Hardwired control
- Microprogrammed Control.

• Hardwired control:

The 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.

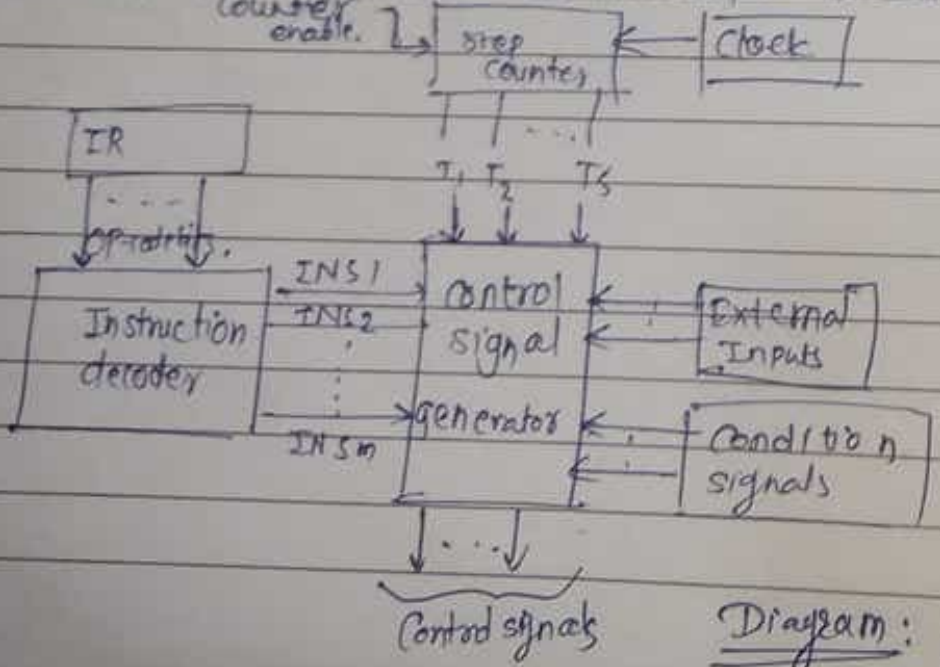
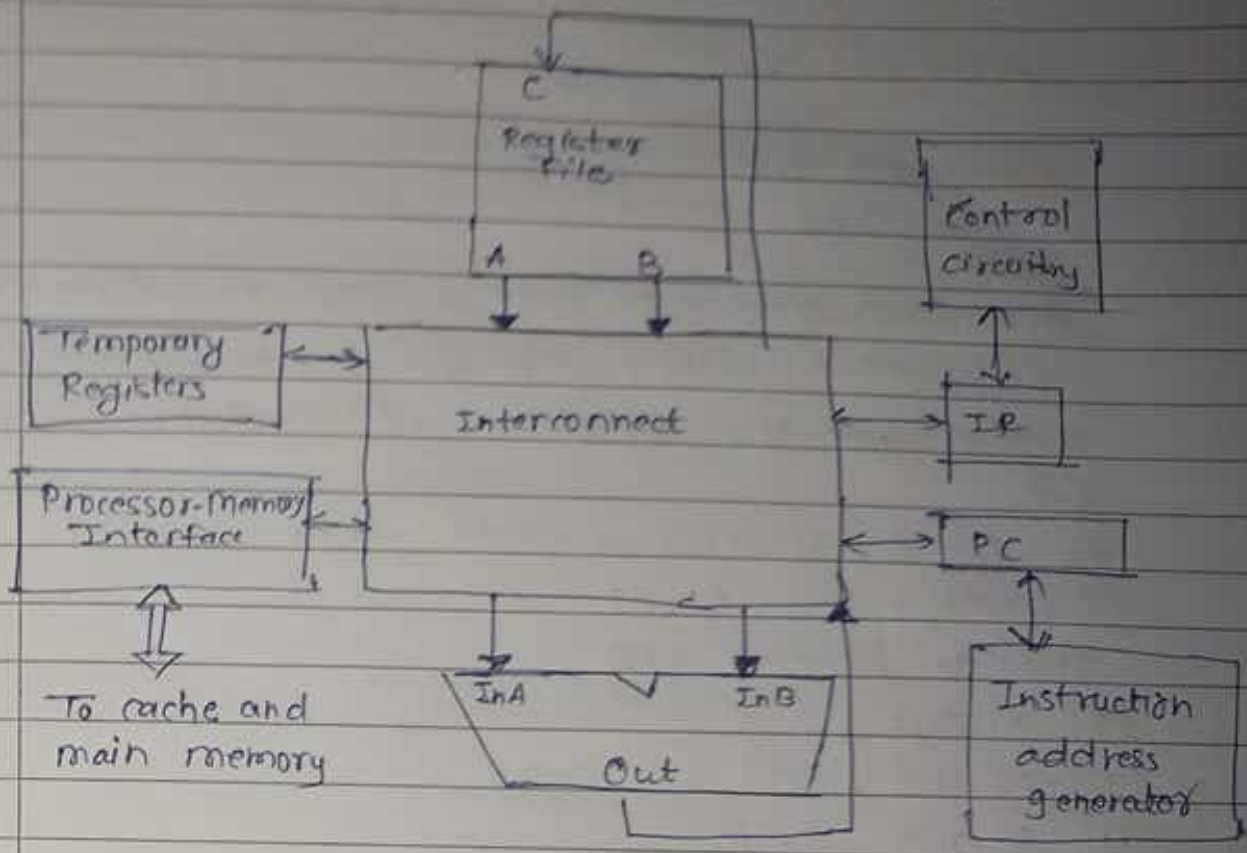


Diagram: Generation of the control signals

CISC style Processor

Diagram: Organization of a CISC-style processor.



Control unit Inputs

Control Unit logic

Note:
As per Instruction in IR there is need of getting sequencing of appropriate signals. It can be done by ~~that means~~ control signals two approaches : ① Hardware control
②

Now let us consider 3 instr. steps:

① INSTR1 add R1, R0
Steps to execute this
(I) R1 out, X in
(II) R0 out, add
(III) Z out, R1 in
(IV) End

② INST2 XOR R0, R5
(I) R0 out, X in
(II) R5 out, XOR
(III) Z out, R0 in
(IV) END

③ INST3 ADD [R0, R6]
(I) R0 out, MAR in, MAR* out, MDR* in
(II) MDR out, X in
(III) R6 out, ADD
(IV) Z out, MDR in, MDR into out, MAR into out, RD
(V) END

Rainbow

Ro out = ((INSTR AND T2) OR → means Ro out should be made 1 if there is instruction 1 & second clock cycle.

(___ AND T1) OR

(___ AND T1) OR ---)

ADD = ~~FINST~~ ((INSTR1 AND T2) OR

(INSTR AND T2) OR ----)

⋮

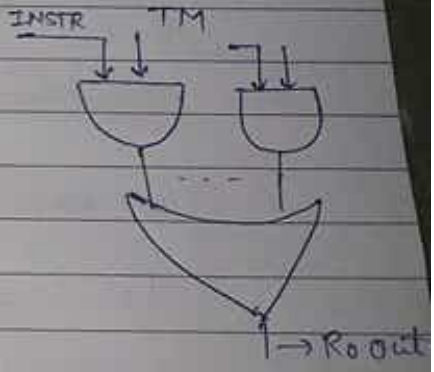
END

END = ((INSTR1 AND T4) OR

(INSTR2 AND T4) OR. - - -)

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

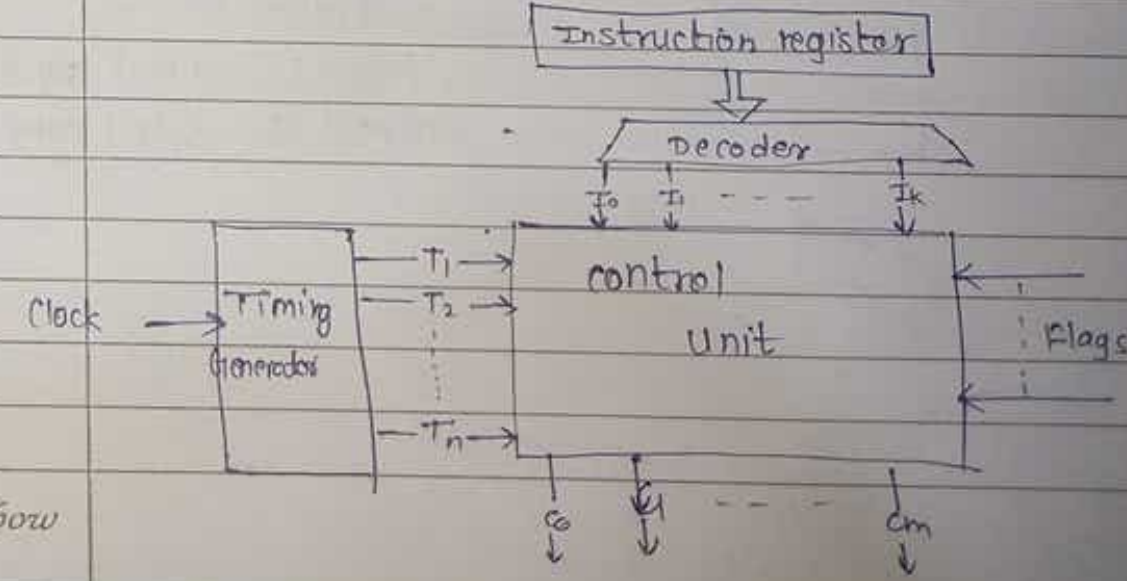
Here, for every instruction there is one particular "and" get and giving to each AND gate on combination timing maker and Instruction.



So, we required two types of signals:

① Time marker ② Instruction marker.

Diagram: Control unit with decoded inputs.



Rainbow

Different Instructions with different steps means naturally different no. of clock cycles.

Ex: 8 steps - 3 bit counter
16 steps - 4 bit counter.

End signal connected to n-bit counter that indicates one instruction is completed and set n bit count to 0000.

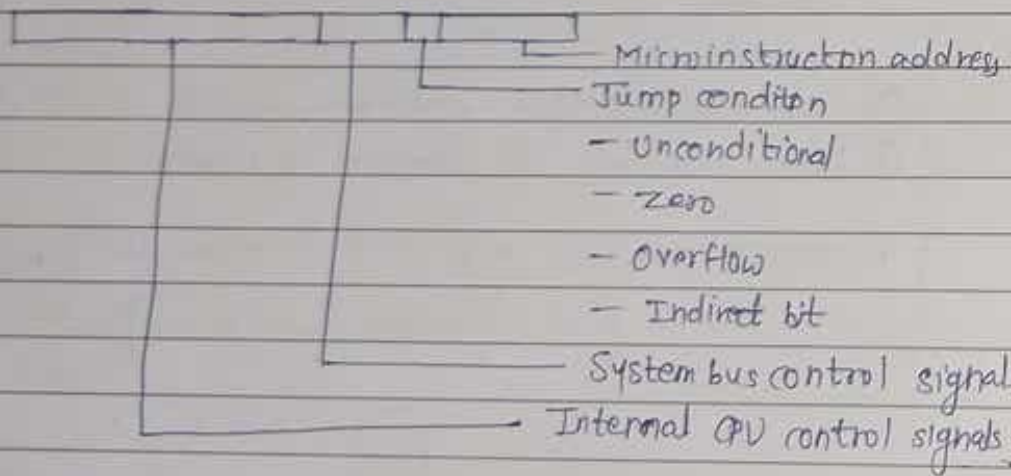
After the previous step, now IR is loaded with new instruction and counter starts counting the clock, again from start.

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

No flexibility for change as it is permanently implemented in hardware architecture.

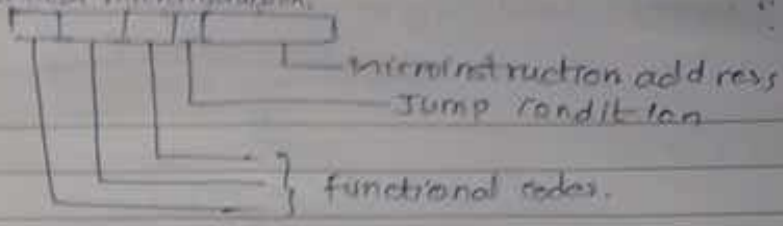
Diagram: Typical Microinstruction Formats.

a) Horizontal microinstruction



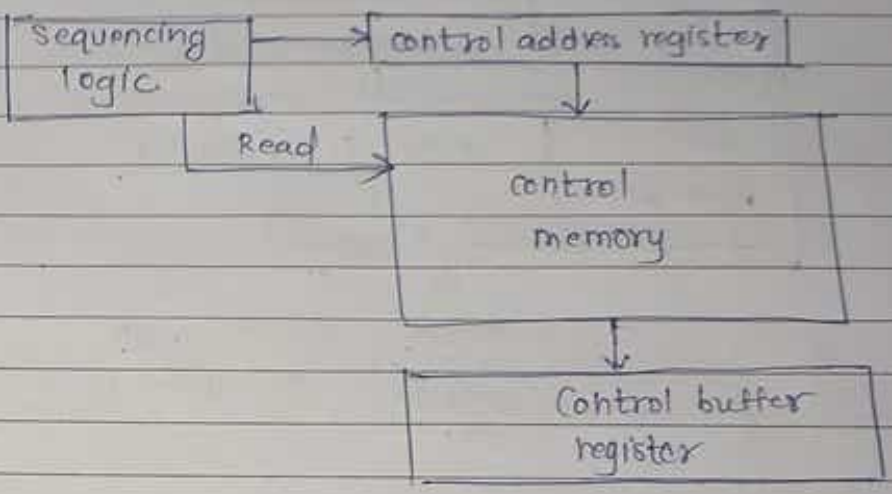
Rainbow

(b) Vertical microinstructions.



- control-memory: control words or microinstructions could be arranged in a control memory.

Diagram: Control unit microarchitecture.



- The control unit functions as follows:-

① To execute an instruction, the sequencing logic unit issues a READ command to the control memory.

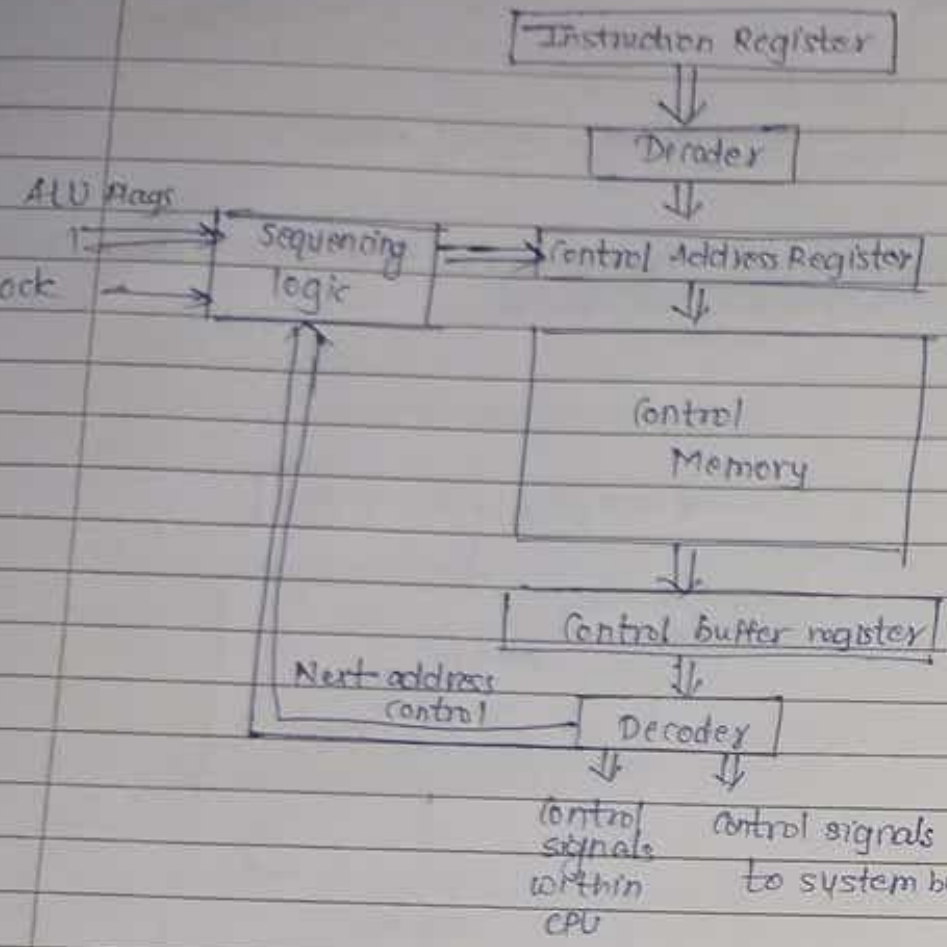
② The word whose address is specified in the control address register is read into the control buffer register.

③ The contents of the control buffer register generates control signals and next-address information for the sequencing logic unit.

④

Rainbow

Diagram: Functioning of Microprogrammed control Unit.



RISC - H/w based approach
CISC - microprograms.

Attributes	Hardware based CU	Microprogram based CU
① Speed	Faster	Slower
② Cost of implementation	More costly	Cheaper.
③ Flexibility	Not at all flexible	Flexible to accomodate new system
④ Ability to handle complex instructions	Difficult to handle complex instructions	Difficult to handle complex instructions
⑤ decoders	Complex decoder and sequencing logic	Easier decoder and sequencing logic.
⑥ Applicability.	Used by RISC kind of processors	Used by CISC kind of processors.

Rainbow

① Instruction set size	Small	large.
⑧ Control Memory	Absent	Present.
⑨ Chip area requirement	less	More.
⑩ Occurrence of Error	Is more	Is less
	Horizontal Microinstruction.	Vertical Microinstructions.
①	It support longer control word	Supports shorter control word.
②	It supports higher degree of parallelism if degree is n → n control signals are enabled at time	It allows low degree of parallelism either 0 or 1
③	No additional hardware is required	Additional hardware is required in form of decoder.
④	Faster as compared to vertical	Slower as compared to horizontal.
⑤	less Flexible	More Flexible
⑦	Makes less use of ROM encoding compared to vertical	More use of ROM encoding to reduce length of control word

Rainbow

. ARITHMETIC FOR COMPUTERS

. Characteristics of 2's complement Representation and Arithmetic

Range	-2^{n-1} through $2^{n-1}-1$
Number of representations of zero	One
Negation	Take the boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of bit length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both +ve or both -ve) are added, then overflow occurs if and only if the result has the opposite sign.
Substraction Rule	To substract B from A, take the 2's complement of B and add it A

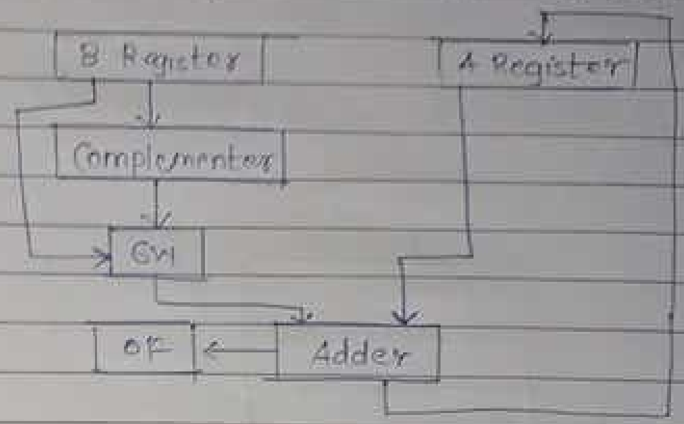
- Range Extension:

- Range of numbers that can be expressed is extended by increasing the bit length.

- In sign-magnitude notation this is accomplished by moving the sign bit to the new leftmost position, and fill with zeroes.

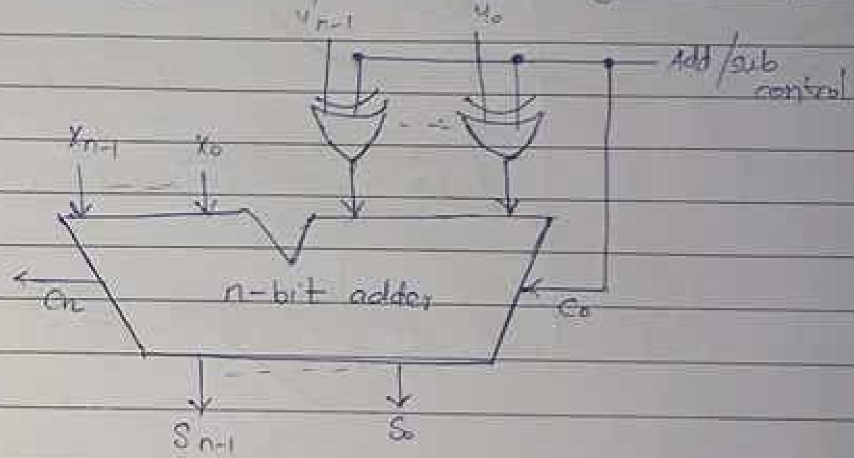
Rainbow

Diagram: Block diagram of Hardware for Addition and Subtraction

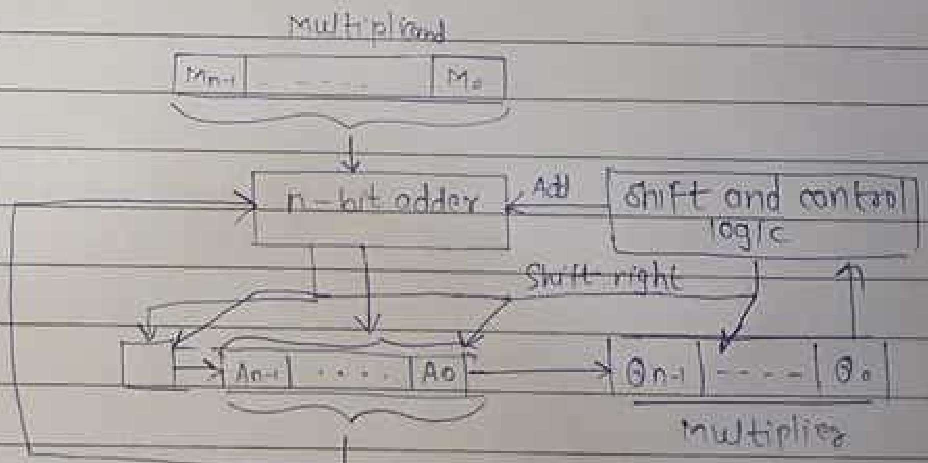


OF = Overflow bit
SW = switch

Diagram: Binary Addition/Subtraction logic circuit



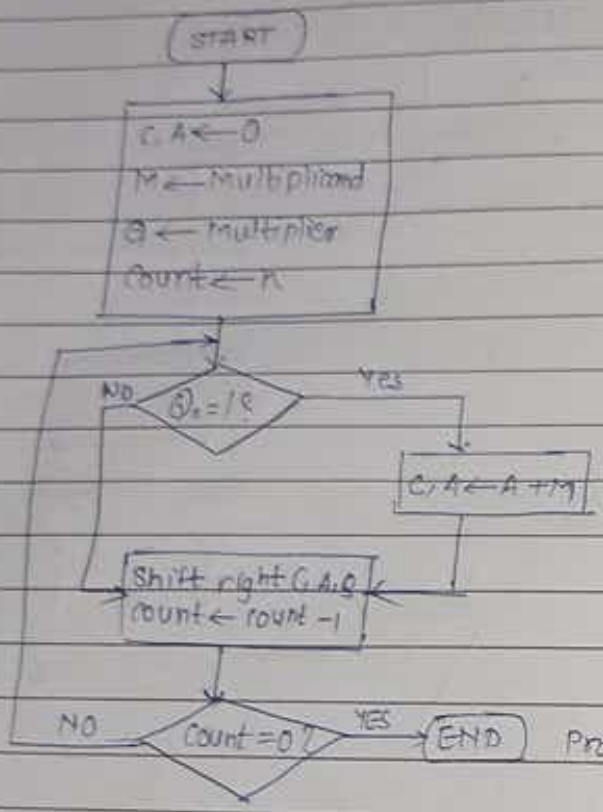
Diagram



M Register → Multiplicant
Q → Multiplier
Two additional register intialize to zero.

Register A initially initialise to zero.
One ~~register~~ bit C registe for carryout

Diagram: Flowchart for Unsigned Binary Multiplication



Su

-7 & 3

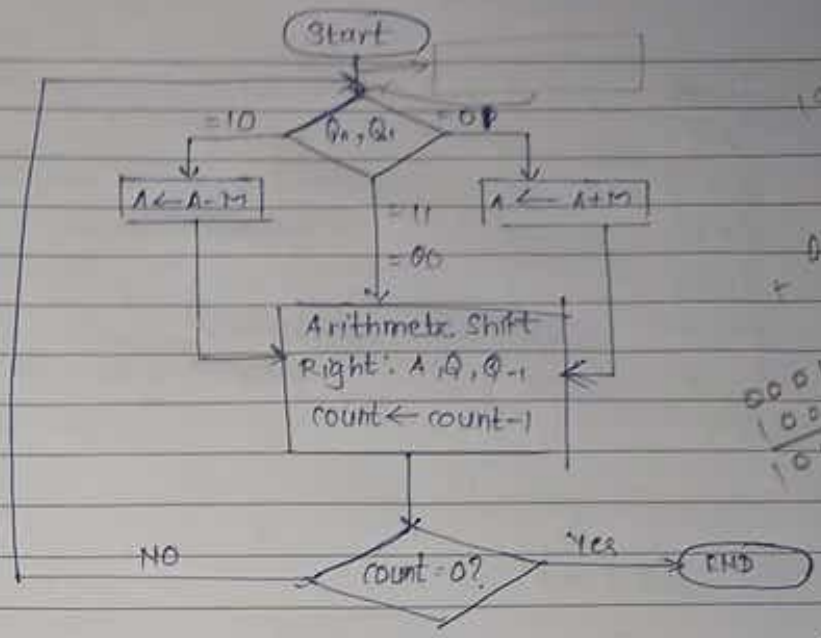
7 = 0111 ∴ -7 = 1001

C	A	Q	M
0	0000	0011	1001
0	1001	0011	1001 } First cycle
0	0100	1001	1100 }
1	0010	1001	1100 } 2nd cycle
0	1000	0100	1110 }
0	0100	0010	0111 } 3rd cycle
0	0010	0001	0011

(27)

Rainbow

Booth's Algorithm for 2's complement Multiplication.



1001 0111

0000
+

0001
1001
1010
0111
1001

count = 4

A	Q	Q-1	M		
0000	0011	0	1001		
0111	0011	0	1001	A ← A-m	} First cycle
0011	1001	1	1001	Shift	}
0001	1100	1	1001	Shift	} second cycle
1010	1100	1	1001	A ← A+m	} third cycle
1101	0110	0	1001	Shift	}
1110	1011	0	1001	Shift	} 4th cycle

↓
Take 1's complement

0001 0100
1
0001 0101
16 4 1 = 21

Rainbow

Q = 7 m = -3

0011
1101
0

count = 4

A	Q	Q-1	M	
0000	0111	0	1101	
0011	0111	0	1101	A ← A-m } first cycle
0001	1011	1	1101	shift
0000	1101	1	1101	shift } second cycle
0000	0110	1	1101	shift } 3rd cycle
1101	0110	1	1101	A ← A+m } 4th cycle
1110	1011	0	1101	shift

1111
+ 0011
0010

0001 0101
16 4 1
= 21

0001
1101
1110

⇒ indicates -ve sign.

Q = -7 , m = -3

0011

C = 1

A	Q	Q-1	M	
0000	1001	0	1101	
0011	1001	0	1101	A ← A-m } first cycle
0001	1100	1	1101	shift
1110	1100	1	1101	A ← A+m } 2nd cycle
1111	0110	0	1101	shift
1111	1011	0	1101	shift } 3rd cycle
0010	1011	0	1101	A ← A-m } 4th cycle
0001	0101	1	1101	shift

21

Rainbow