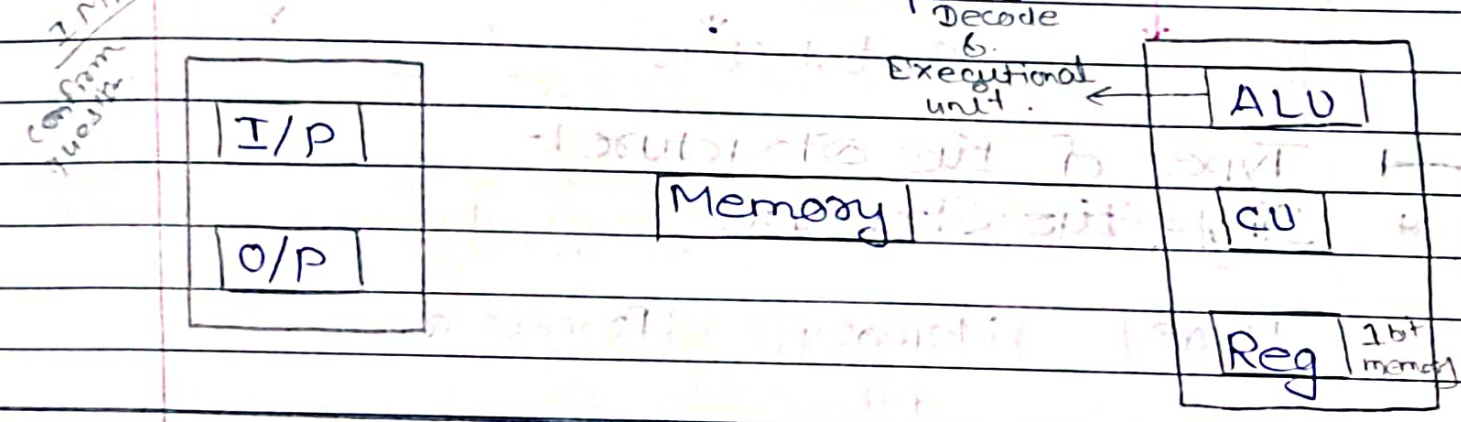
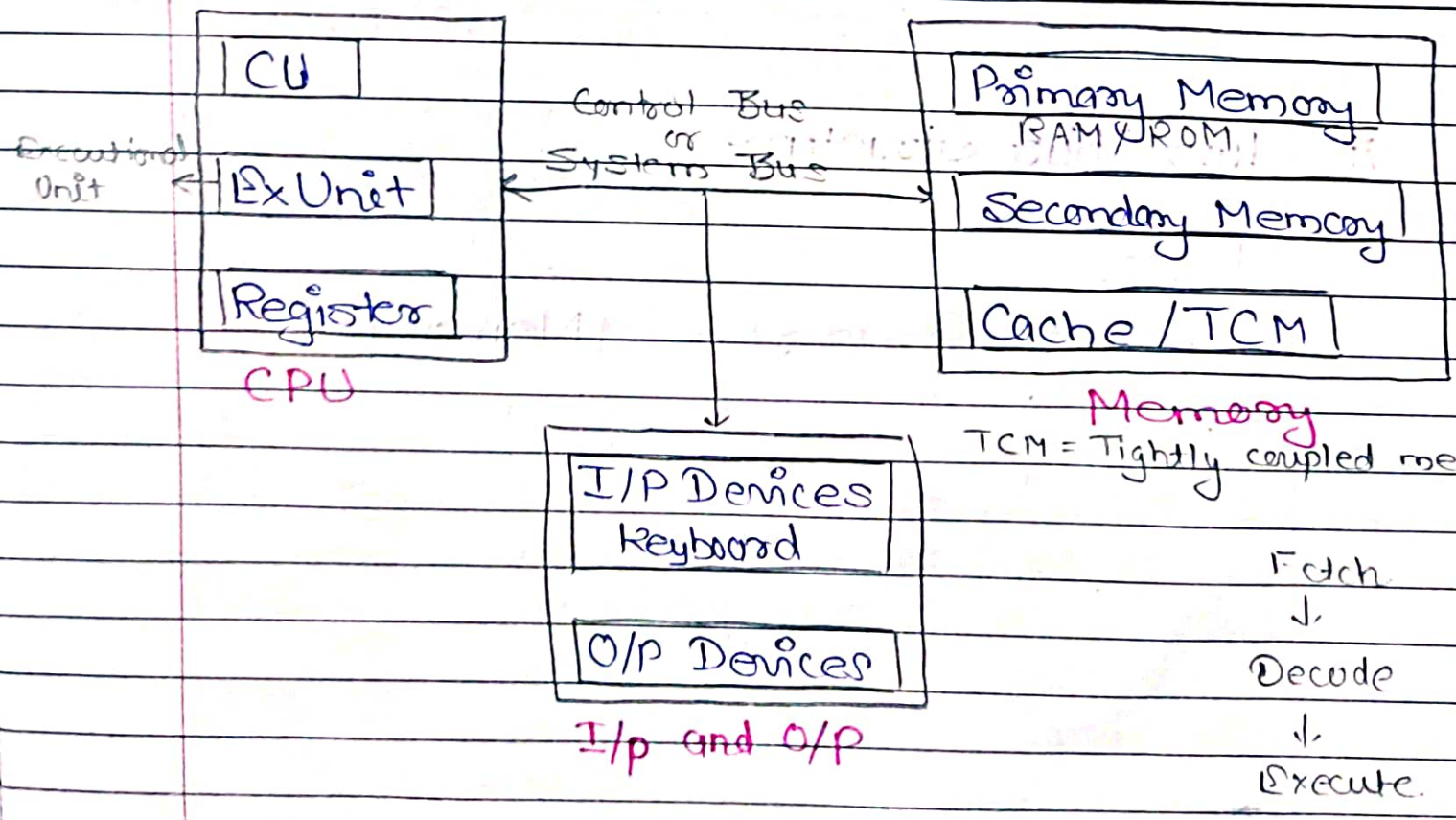
Date 13/7/24
Page

CA	CO
• What	• How
• Functional behaviour.	• Structural relationship
• Instruction set, Addressing Mode	• CKT design, CPU, ALU, Memory
• Low level design issue	• High level design issue

* Functional Units of Computer

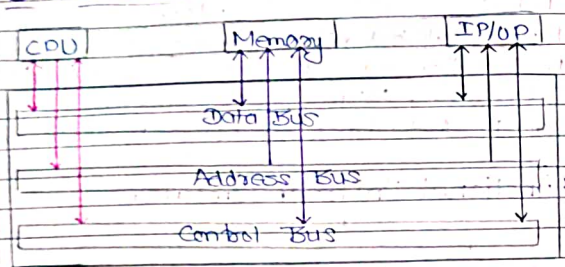




Fetch
↓
Decode
↓
Execute

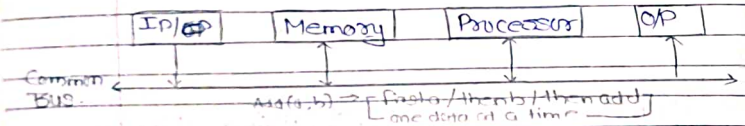
Date 19/7/24
Page

* Bus Structure

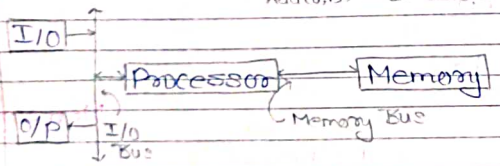


→ Types of Bus Structure :-

* Single Bus Structure
- capable of doing all operations
- also operations like read & write
- one instruction at a time

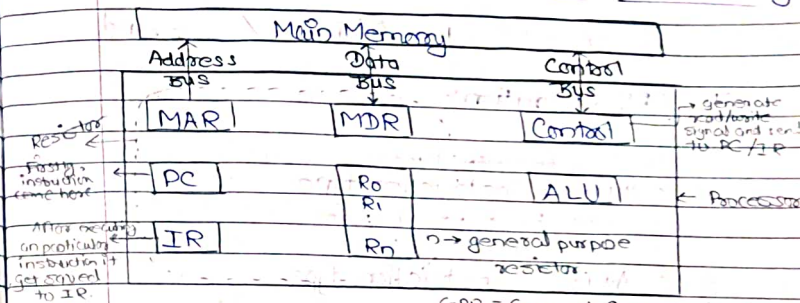


* Two Bus Structure
- Multiple instructions at the same time
Add(a,b) → all data at same time



MAR = Memory Address Register
IR = Instruction Register
Program counter = PC
Memory Data Register = MDR

* Connection between Processor & Main Memory



GPR = General Purpose Register

Example :-

Add ; R0, LOCA (Location A)
// R0 ← R0 + M(LOCA)

OR — Load the content R1 at location A and then add content with register R1

Load R1, LOCA // R1 ← M(LOCA)
Add : R0, R1 // R0 ← R0 + R1

MAR – carries the address of the location where data is going to be stored.
MDR – raw data present in location
PC – points to address of next instructions which need to be executed.
GPR – intermediate results are stored here temporarily.
IR – recieves and executes current working instructions.

* Instruction Format

Mode	OP code	Address / Operand
	operation	

* CPU Operations

- Stack based organisation
- Single accumulator organisation
- General resister organisation

} Types of CPU operation

→ Instruction type :-

1] 3 Address instruction type

OP code ← (Add) | (R1, R2, R3) | → operand or Address

R1 ← R2 + R3

} General resister organisation.

2] 2 Address instruction type

Add (R1), (R2) → Source

Destination R1 ← R1 + R2

3] 1 Address instruction type <Single accumulator organi>

4] 0 Address instruction type. <Stack based organisation>

Add ⇒ Load x — Here accumulator is used rather than resistors.

Add x

A ← M[x]

Temporary storage provided by accumulator ← (Ac) ← Ac + M[x]

→ Zero Address

Push A

Add B

Pop C

1] 3 Address

Destination address → Source address

OP Code	DA	SA1	SA2

e.g- C = A + B // M[C] ← M[A] + M[B]

Add C, A, B

2] 2 Address

OP Code	DA	SA

e.g- C = A + B // M[C] ← M[A]

Mov C, A → M[C] ← M[C] + M[B]

Add C, B

↳ Moving A to C as we have only two string address

3] 1 Address

OP Code	Operand Address <SA>

e.g- C = A + B // A[C] ← M[A] → Accumulator

load A (Ac) ← (Ac) + M[B]

add B M[C] ← Ac

store C

4] 0 Address

e.g- C = A + B → Top of the stack

Push A (Tos) ← A

Push B Tos ← B

add Tos ← A + B

Pop C M[C] ← T.O.S

(A + B) × C

add R1, A, B // R1 ← M[A] + M[B]

mul R1, C. // R1 ← R1 * M[C]

Q. Write a program to evaluate arithmetic statement.

$X = (A+B) * (C+D)$

Soln Three address instruction

```
ADD R1, A, B    // R1 ← M[A] + M[B]
ADD R2, C, D    // R2 ← M[C] + M[D]
MUL X, R1, R2   // M[X] ← R1 * R2
```

Two address instruction

```
MOV R1, A       // R1 ← M[A]
ADD R1, B       // R1 ← R1 + M[B]
MOV R2, C       // R2 ← M[C]
ADD R2, D       // R2 ← R2 + M[D]
MUL R1, R2      // R1 ← R1 * R2
MOV X, R1       // M[X] ← R1
```

One address instruction

```
load A          // Ac ← M[A]
ADD B           // Ac ← Ac + M[B]
Store T         // M[T] ← Ac
load C          // Ac ← M[C]
ADD D           // Ac ← Ac + M[D]
MUL T           // Ac ← Ac * M[T]
Store X         // M[X] ← Ac
```

Zero address instruction

```
Push A          // TOS ← A
Push B          // TOS ← B
add             // TOS ← (A+B)
Push C          // TOS ← C
Push D          // TOS ← D
add             // TOS ← (C+D)
Mul             // TOS ← (C+D) * (A+B)
Pop X           // M[X] ← TOS
```

Q. $Y = \dfrac{(A-B)}{[C+(D*E)]}$

Soln Three address instruction

```
SUB R1, A, B    // R1 ← M[A] - M[B]
Mul R2, D, E    // R2 ← M[D] * M[E]
Add R2, C, R2   // R2 ← R2 + M[C]
DIV Y, R1, R2   // M[Y] ← R1 / R2
```

Two address instruction

```
MOV R1, A       // R1 ← M[A]
SUB R1, B       // R1 ← R1 - M[B]
MOV R2, D       // R2 ← M[D]
MUL R2, E       // R2 ← R2 * M[E]
Add R2, C       // R2 ← R2 + M[C]
MOV Y, R1       // M[Y] ← R1
DIV Y, R2       // M[Y] ← M[Y] / R2
```

One address instruction

```
load A          // Ac ← M[A]
Sub B           // Ac ← Ac - M[B]
Store T         // M[T] ← Ac
load D          // Ac ← M[D]
Mul E           // Ac ← Ac * M[E]
Add C           // Ac ← Ac + M[C]
```

DIV T	// AC ← M[T] / AC
Store Y	// M[Y] ← AC

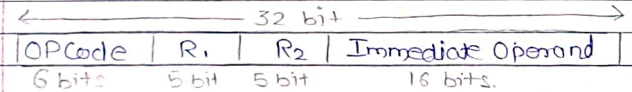
Zero address instruction

Push A	Tos ← A
Push B	Tos ← B
Sub	Tos ← (A-B)
Push C	Tos ← C
Push D	Tos ← D
Push E	Tos ← E
MUL	Tos ← (D * E)
Add	Tos ← C + (D*E)
Div	Tos ← (A-B)/(C+(D*E))
Pop Y	Y ← Tos

26/7/24

Q Processor has 40 distinct instruction and 24 general purpose resistor. A 32 bit instruction word has an op-code, two resistor operands and an immediate operand. Calculate the number of bits available for immediate operand?

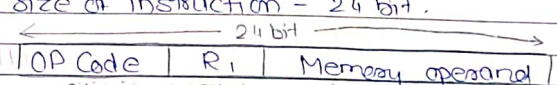
Soln Instruction = 40 = OPcode = (2^6) 6 bits
Resistors (length) = 24 = 5 bits (2^5)
Size of instruction = 32 bit



∴ Size of immediate operand
= 32 − (6 + 5 + 5)
= 16 bits

Q A processor has 512 distinct instruction and 80 general purpose resistor. A 24 bit instruction word has an OP-code, one resistor operand and a memory operand. How many bits are reserved for memory operand field?

Soln Instruction = 512 = OP-code = 2^9 = 9 bits.
Resitor length = 80 = (2^7) 7 bits.
Size of instruction = 24 bit.



∴ Size of memory operand
= 24 − (9+7)
= 8 bits

Q Consider a processor with 64 resistors and an instruction set of size 12. Each instruction has 5 distinct field namely OPcode, two source resistors, one destination resistor and 12 bit immediate value. Each instruction must be stored in byte aligned fashion. If a program has 100 instruction calculate the amount of memory consumed by program?

Soln

Immediate value = 12 bit

Register length = 64 = 2^6 bit

Size of instruction = 12 bit = 2^4

OP Code	R_1	R_2	R_3	Immediate operand.
4 bit	6 bit	6 bit	6 bit	12 bit

For instruction memory required

= 4 + 6 + 6 + 6 + 12

= 34 = 5 byte (4+1)

= 32 + 2

For 100 such instruction

= 100 × 5

= 500 byte.

29/7/24

* Addressing Mode — Unit-2 Topic

Mode	OP Code	Address / operand

↳ is always of 1 bit size

The way in which operand are chosen (fetch) during the program execution depending on addressing mode of instruction.

Mode Field –

Mode field is related to address / operand field. So we name it as a Addressing Mode. It specifies how to interpret the information within the address field.

It specifies how to compute actual and effective address of operand.

31/7/24

→ Types of Addressing Mode

1] Implied Addressing Mode / Implicit } No address is required.
2] Immediate Addressing Mode }
3] Direct Addressing Mode. } Address is specified (Memory address)
4] Indirect Addressing Mode }
R0 5] Resistor – Direct Mode (Memory address specifies resistors)
(R0) 6] Resistor – Indirect Addressing Mode }
7] Autoincreament – Autodecreament AM (then find direct/indirect)
8] Relative Addressing Mode } Address field + content of register
9] Index Addressing Mode }
10] Base Resistor Addressing Mode }

Φ > Also known as Displacement Addressing Mode (3M)

31/7/24

1] Implied Mode (Implicit Mode)

e.g. PUSH, POP

2] Immediate Mode

– Direct value is given.

OP Code	Operand

e.g. load #7 .... AC ← 7

Effective address is given. The address at which we find operand directly is effective address.

3] Direct Mode

Instruction.

OP Code	Add A

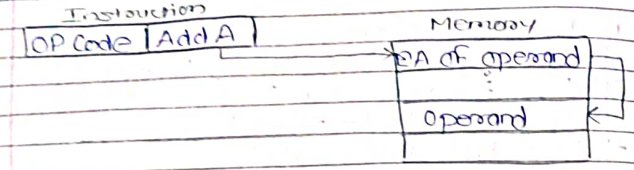
→ Memory: Operand

e.g. ADDA .... AC ← AC + M[A]

LOAD B .... AC ← M[B]

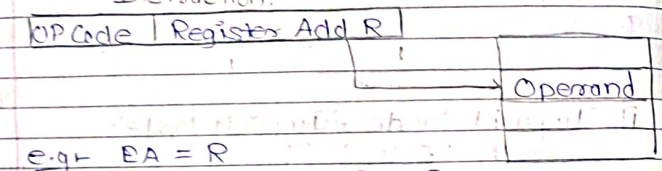
↳ Accumulator is used because it is one add instruction

4] Indirect Mode - 1 add instruction.



e.g:- ADD (A)
or ADD @A .... $AC \leftarrow AC + M[M[A]]$

5] Register Direct Mode



e.g:- EA = R
MOV R_1, R_2 ... $R_1 \leftarrow R_2$

6] Register Indirect Mode
EA = (Ri)

e.g:- ADD $(R_1), R_2$ ... $M[R_1] \leftarrow M[R_1] + R_2$

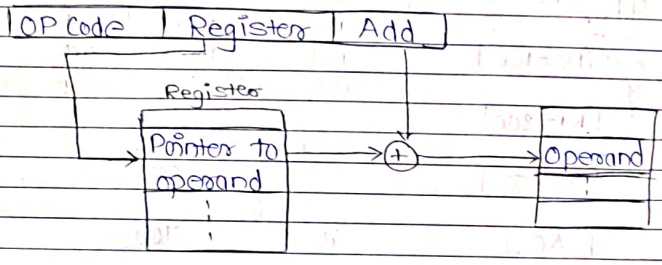
7] Auto Increament / Decreament

In Auto Increament
EA (Ri)+
e.g:- ADD R_1, (R_2)+ ....
— $R_1 \leftarrow R_1 + M[R_2]$,
$R_2 \leftarrow R_2 + 1$

In Auto Decreament
EA = −(Ri)
e.g:-
ADD R_1, −(R_2) .... $R_2 \leftarrow R_2 - 1$
$R_1 \leftarrow R_1 + M[R_2]$

8] Displacement Addressing Mode

Types :-
1] Relative Addressing (EA = (PC) + A) → Relative Register
2] Indexed — 11 — (EA = A + (XR)) → Index Register
3] Base Register — 11 — (EA = Ri + A)



8] Relative Addressing Mode
EA = (PC) + A
e.g:-
load A(PC) .... $AC \leftarrow M[PC + A]$

9] Indexed Addressing Mode

EA = A + (XR) - index resistor

e.g:-

ADD R_1, 20(R_2)

.... $R_1 \leftarrow R_1 + M[20+R_1]$

10] Base Addressing Mode

EA = (R_i + A)

e.g:-

load A(R_i) .... $R_1 \leftarrow M[R_1 + A]$

ADD R_1, 20(R_2) ... $R_1 \leftarrow M[R_2 + 20]$

1/8/24

Q]

[PC=100]

[R1=300]

[XR=50]

[AC]

Add	Memory	
100	load to AC | Mode	2 one word
101	Address = 400	
102	Next instruction	
299	250	
300	600	
400	700	
450	900	
502	375	
700	200	

1] Implicit mode

2] Immediate Mode

EA = —

Operand = 400

3] Direct Mode

EA = 400

Operand = 700

4] Indirect Mode

Add = 400

EA = 700 Operand = 200

5] Register Direct Mode

EA = —

Operand = 300

6] Register Indirect Mode

EA = 300

Operand = 600

7] Auto Increament / Decreament

EA = 300 Operand = 600

EA = 299 Operand = 250

8] Relative Addressing Mode

EA = 400 + 102 = 502

400 = PC

Add = 102

Operand = 375

iii) Index Addressing Mode
EA = 400 + 50 = 450
Operand = 900

iv) Base Addressing Mode
EA = 400 + 300 = 700
Operand = 200

Q Given the following memory values and one address machine with an AC. What values do the following instruction load into AC.

Word 20 contain 70
-11- 40 contain 80
-11- 70 -11- 100
-11- 80 -11- 120

Instructions are
① load immediat 20 20
② load direct 20 100 70
③ load indirect 20 100

Q.

Word 20 contain 40
30 contain 50
40 contain 60
50 contain 70

Instructions are
① load immediate 20 Operand = 20
② Direct 20 EA = 20 Operand = 40
③ Indirect 20 EA = 40 Operand = 60

Q ① load direct 60 EA → X & operand → X
② load indirect 60 EA = X & operand = X

Q Word 20 contain 40
30 contain 50
40 contain 60
50 contain 70

Instructions are.
1] Immediate 20 Operand = 20
2] Direct 20 EA = 20 Operand = 40
3] Indirect 20 EA = 40 Operand = 60
4] Direct 60 EA = X Operand = X
5] Indirect 60 EA = X Operand = X

Q An instruction is stred at location 500 with its address field at loc 501. Address field has value 600. A process or register R = 400 evaluate the EA and operand if addressing mode of instruction are
(a) Direct AM (b) Immediate AM
(c) Relative AM (d) Register Indirect AM
(e) Index with R.

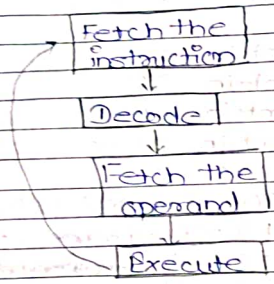
500	load to AC	Mode
501	Address = 600	
502	Next instruction.	

R = 400

(a) EA = 500 O = 60 (b) O = 500 (c) 500 + 502 O = X = 1102
(d) EA = X O = X (e) EA = 600 + XR = 600 + 400 = 1000

7/8/24

Instruction Cycle



* RISC and CISC

These are instructions set architecture.
Provides interface betn hardware & software.
(e.g. mobile)

RISC → Reduce instruction set computer.
reduce large inst. to smaller ones.

CISC → Complex instruction set computer.
require more no. of instruction for single operations (10-20 just for one operation)

It provides timing n number of micro instruction using Time Generator. ← Clock

Hardware used :-
1) Combinational ckt
2) Encoders.

To → Time generated for micro instruction.
T_1

ID – Instruction decoder.

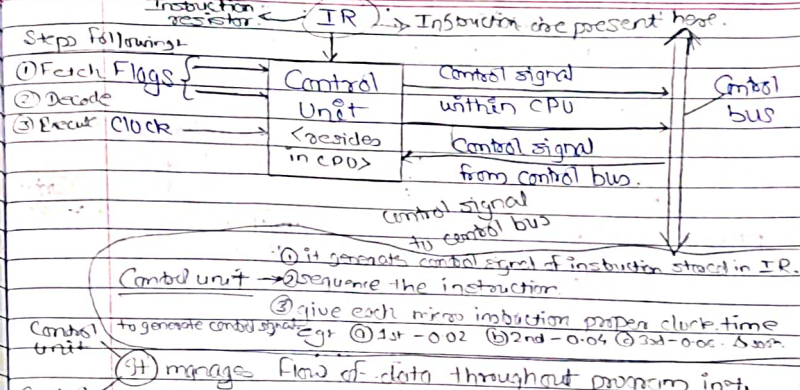
* All instruction generates into control signal. It once using single clock in Hardwired Control Unit. All signals are active at a time.

Unit-5

(1) Hardwired Control Unit
(2) Bus Organization
(3) Micro-programs Control Unit
(4) Numericals

Size short questions for exam.

9/8/24



Flags – are basically status resistors, that provides status of a particular resistor.

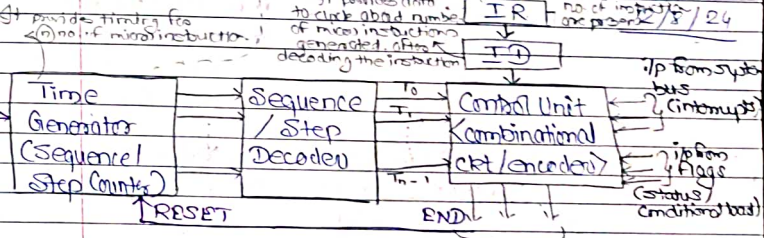


Fig. Hardwired Control Unit <Block symmetry>

16 bit instruction.

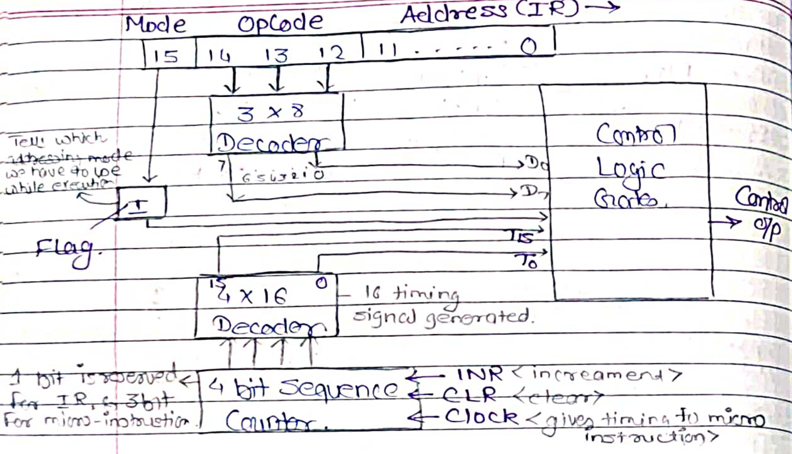


Fig: Hardwired Control Unit <Executional diagram>.

As we have 3 instruction, then the decoder will convert that instruction into 2^{3} micro instruction.

$2^3 = 8$ signals generated.

As the signals are generated it is denoted as $D_0 - D_7$

Microprogram

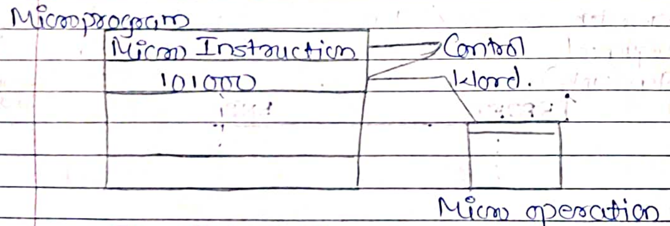


Fig. Control Memory

CAR = Control Address Resistor.
↳ same as MAR <Memory Address Resistor>

14/2/24

Micro instruction are present here

μPC = micro program counter
↳ it holds address of next instruction to be executed and points the next micro instruction.

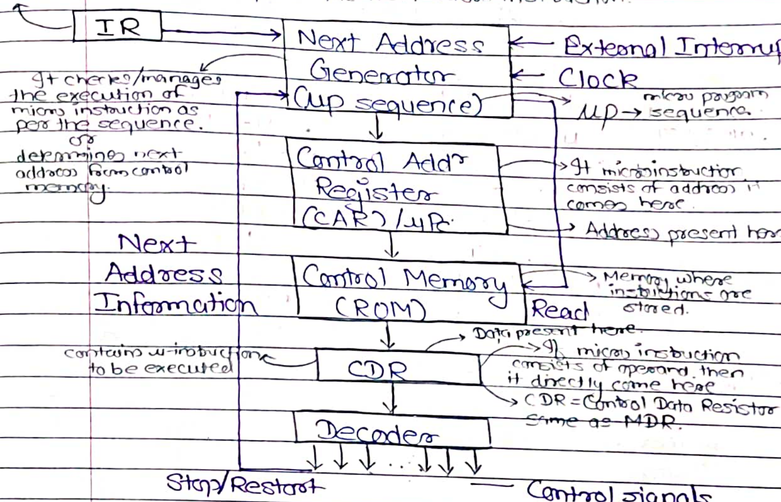


Fig. Micro-program Control Unit.

Q Hardwire Control Unit Vs μ-processor Control Unit

1]	Control signals	-	Hardware	μ-programming
2]	Modification	-	Difficult	Easier
3]	Instruction set	-	works on smaller instruction	works on larger/large instruction set
4]	Debugging	-	Difficult	Easier
5]	Execution speed.	-	Faster	Low as compared.
6]	Memory	-	Less	More memory
7]	Cost	-	High	Low
8]	Processor	-	RISC processor	CISC processor
9]	Structure.	-	Hardware approach.	Software approach

Date
Page

μ-programmed control unit

horizontal μ-CU

- μ-instruction
- C_0 | C_1 | C_2 | C_3
- Decoder (n×2^n)
- Control signal.
- Memory issue occurs after certain limit i.e above 64 bit
- Memory increase/expand horizontal.

vertical μ-CU

- Memory expand vertically
- No memory restriction.

	Horizontal	Vertical
CS	Less	More
Decoder	No decoder	Present
Memory	More	Less
Execution sp	Fast	Slow as compared

2 4 8 16 32 64 128 256 512 1024

Date
Page

1] Consider a hypothetical control unit which has total 233 control signal. How many bits are required in control unit for horizontal micro programming and vertical μ-programming?

→ For vertical μ-programming,
Control signal = 233 (128 < 233 < 256)
∴ No. of bits required = 2^n
$233 = 2^8$
8 bits required.

For horizontal μ-programming,
Control signal = 233 <input>
No. of bits required = 233 control signals.

2] Consider a μ-program control unit design which support 7 group of mutually exclusive control signals. How many more control bit are required using horizontal μ-programming over vertical μ-programming?

Group	G_1	G_2	G_3	G_4	G_5	G_6	G_7
CS	21	10	4	1	18	23	6

→ For horizontal μ-programming
Control signal = 64
∴ No. of bits required = 64 control signal.

For vertical μ-programming
Control signal = 21 + 4 + 2 + 1 + 5 + 5 + 3
∴ No. of bits required = 21 bits
= bits.

16 bit = 1 word

More Control bits required
= Horizontal − Vertical
= 64 − 21
= 43 bits

26/8/24

* Micro-Instruction Format

Control Field	Address Field	Flags	Branch Condn

Horizontal = C bits — no. of control word — no. of bits = $\log_2 B$
Vertical = $\log_2 C$ — no. of bits = $\log_2 f$
- size of control memory
$\geq 2^n \times$ size of control word

Q Consider a control memory unit in which 22 bits u-instruction is stored. The u-instruction having 3-fields — address field, branch field having size of 8 bit and contain 16 flags. What is the size of control memory?

Ans u-instruction size = 22 bits.
Branch field = 8 bits.
No. of Flags = 16 = 4 bits (2^4)

← 22 bits →

Address Field	Flags	Branch field
10 bit	4 bit	8 bits.

$2^{10} = 1k$
$4k = 4 \times 2^{10}$

Address Field = 22 − 8 + 4
= 10 bits

∴ Size of control memory = $2^n \times$ size of control word
$= 2^{10} \times 22$
= 22528 bits

Q Consider a control unit that support that support 4k words a hardwire contain 64 control signal and 16 flags what is the size of control word/ memory used in bits and bytes? using horizontal and vertical programming?

→

Control signal	Flags	Address
6 bits.	4 bits.	12 bits

Address = 4k = $4 \times 2^{10} = 2^2 \times 2^{10} = 2^{12}$
= 12 bits.
Control signal = 64 = 2^6 = 6 bits.
Flags = 16 = 2^4 = 4 bits.

Horizontal Control signal = 64 & Vertical Control signal = 6

∴ Control word = 6 + 4 + 12
(Vertical) = 22 bits.
Control word = 4 + 64 + 12
(Horizontal = 80 bits

Size of control memory (Horizontal) $= 2^n \times 80$
$= 2^{12} \times 80$
$= 327680$ bits.
$= 40960$ bytes.

Size of control memory (Vertical) $= 2^n \times 22$
$= 2^{12} \times 22$
$= 90112$ bits
$= 11264$ bytes.

① A hardwired uses 10 control signals S_1 to S_{10} in various time clock T_1 to T_5 to implement instruction I_1 to I_4 as shown below. Find the boolean expression to generate control signal S_5.

	T_1	T_2	T_3	T_4	T_5
I_1	$S_1, S_3, (S_5)$	S_2, S_4, S_6	S_1, S_7	S_{10}	S_8, S_8
I_2	$S_1, S_3, (S_5)$	S_2, S_9, S_{10}	$(S_5), S_6, S_7$	S_6	S_{10}
I_3	$S_1, S_3, (S_5)$	S_7, S_8, S_{10}	S_2, S_8, S_9	S_{10}	S_1, S_3
I_4	$S_1, S_3, (S_5)$	S_2, S_6, S_7	$(S_5), S_{10}$	S_6, S_9	S_{10}

Soln $T_1 = S_5(I_1, I_2, I_3, I_4) = T_1$ is generate in all instruction.
$T_2 = S_5(I_2, I_4)$

∴ Boolean expression
$= T_1 + T_3(I_2, I_4)$

29/8/24

① Computer support 5 instruction I_1 to I_5 and four time cycles T_1 to T_4. Control signals for computer 'Z' is $Z = (I_1 + I_2)T_1 + T_2 + T_3 + T_4(I_2 + I_3 + I_4 + I_5)$ During the execution I_5 what are the time cycles for execution.

Soln $Z = T_1(I_1 + I_2) + T_2 + T_3 + T_4(I_2 + I_3 + I_4 + I_5)$

$T_1 = I_1, I_2$
$T_2 = I_1, I_2, I_3, I_4, (I_5)$
$T_3 = I_1, I_2, I_3, I_4, (I_5)$
$T_4 = I_2, I_3 + I_4, (I_5)$
∴ ~~Boolean expression~~
∴ Time cycle for I_5 is T_2, T_3 and T_4

$I_5 = ?!$

	~~I_1~~	~~I_2~~	~~I_3~~	~~I_4~~	~~I_5~~
	T_1	T_2	T_3	T_4	
→	I_1	I_1	I_1		
	I_2	I_2	I_2	I_2	
		I_3	I_3	I_3	
		I_4	I_4	I_4	
		(I_5)	(I_5)	(I_5)	

Time cycle $= \{T_2, T_3, T_4\}$