Tutorial-2 (Pradip Dhoj... 201751

Q) 1) Describe the working principle of a very simple CPU that can execute the following instructions.

Instruction	Instruction code	Operation
ADD	00AAAAAA	AC ← AC + M[AAA
AND	01AAAAAA	AC ← AC ∧ M[AAAAAA
JMP	10AAAAAA	GO TO AAAAAA
INC	11xxxxxx	AC ← AC + 1

Draw the state diagram and explain how each instruction is executed.

⇒ Answer.

In addition to AC, this CPU needs several additional registers to perform the internal operations necessary to fetch, decode, and execute instructions. The registers in this CPU are fairly standard and are found in many CPUs; their sizes vary depending on the CPU in which they are used. This CPU contains the following registers:

↳ A 6-bit address register, AR, which supplies an address to memory via A[5...0]

↳ A 6-bit program counter, PC, which contains the address of the next instruction to be executed.

↳ An 8-bit data register, DR, which receives instructions and data from memory via D[7...0].

↳ A 2-bit instruction register, IR, which stores the opcode portion of the instruction code fetched from memory.

→ Fetching instruction from memory:
↳ The address of instruction to be fetched is stored in the PC. Since, address register (AR) supplies an address to memory through A[5.....0] pins, firstly copy the content of PC to AR.
Fetch 1: AR ← PC

↳ The CPU must read the instruction from the memory and store in DR.
Fetch 2: DR ← M, PC ← PC+1

↳ Finally, two higher order bits of DR are copied into IR: these two bit indicate which instruction is to be executed, and the CPU copies the lower order 6-bit of DR into AR.
Fetch 3: IR ← DR[7, 6], AR ← DR[5.....0]

→ Decoding Instruction:
↳ The CPU must determine which instruction it has fetched so as to invoke the current routine.
↳ The value of IR i.e, 00, 01, 10 and 11 determine which routine is to be invoked.

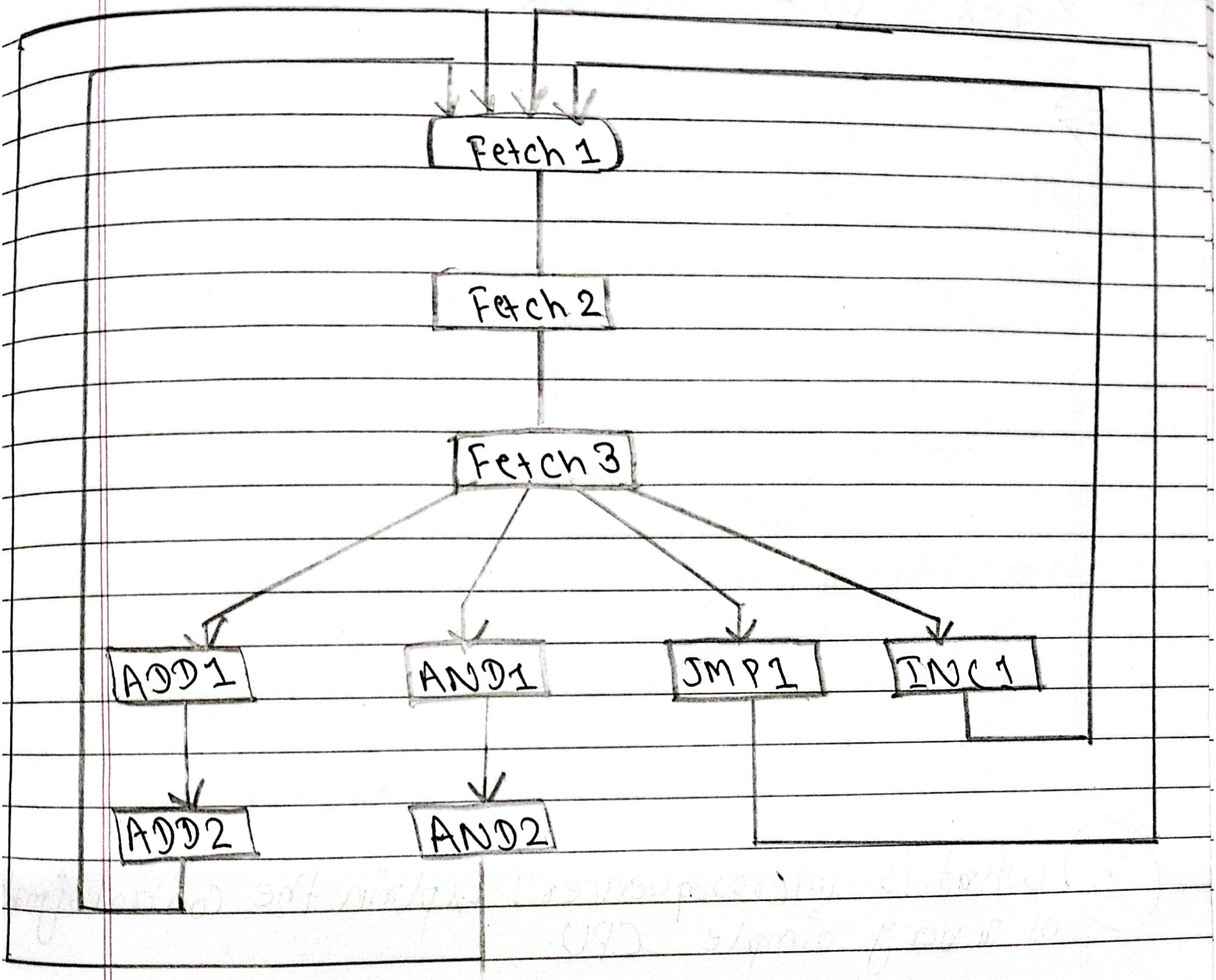
→ Executing Instructions:

↳ ADD instruction:
⇒ First it must fetch one operand from the memory.
⇒ Then, it must add this operand to the current content of accumulator and store the result back into the accumula
ADD 1: DR ← M
ADD 2: AC ← AC + DR

↳ AND instruction:
AND 1: DR ← M
AND 2: AC ← AC ^ DR

↳ JMP instruction:
⇒ The address to which the CPU must jump is copied into the PC.
⇒ Since, the result is already stored in DR[5....0], we simply copy this value to PC.
JMP 1: PC ← DR[5....0]
⇒ INC instruction:
INC 1: AC ← AC + 1

State diagram:

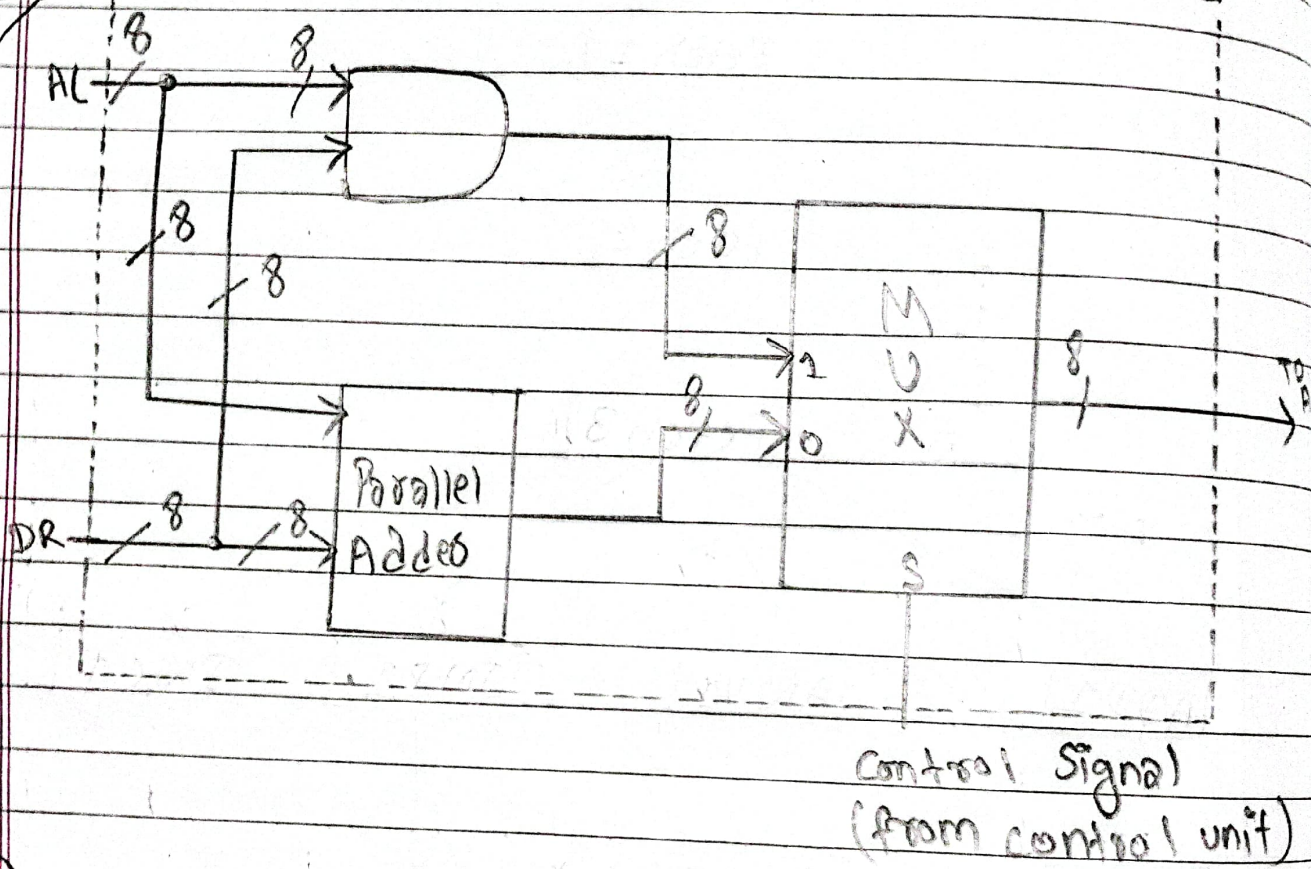


→ Establishing required data path:

Fetch 1: AR ← PC
Fetch 2: DR ← M, PC ← PC + 1
Fetch 3: IR ← DR[7, 6], AR ← DR[5....0]
ADD 1: DR ← M
ADD 2: AC ← AC + DR
AND 1: DR ← M
AND 2: AC ← AC ^ DR
JMP 1: PC ← DR[5....0]
INC 1: AC ← AC + 1

(2) Design the ALU that performs AC ← AC+DR and AC ← AC ^ DR

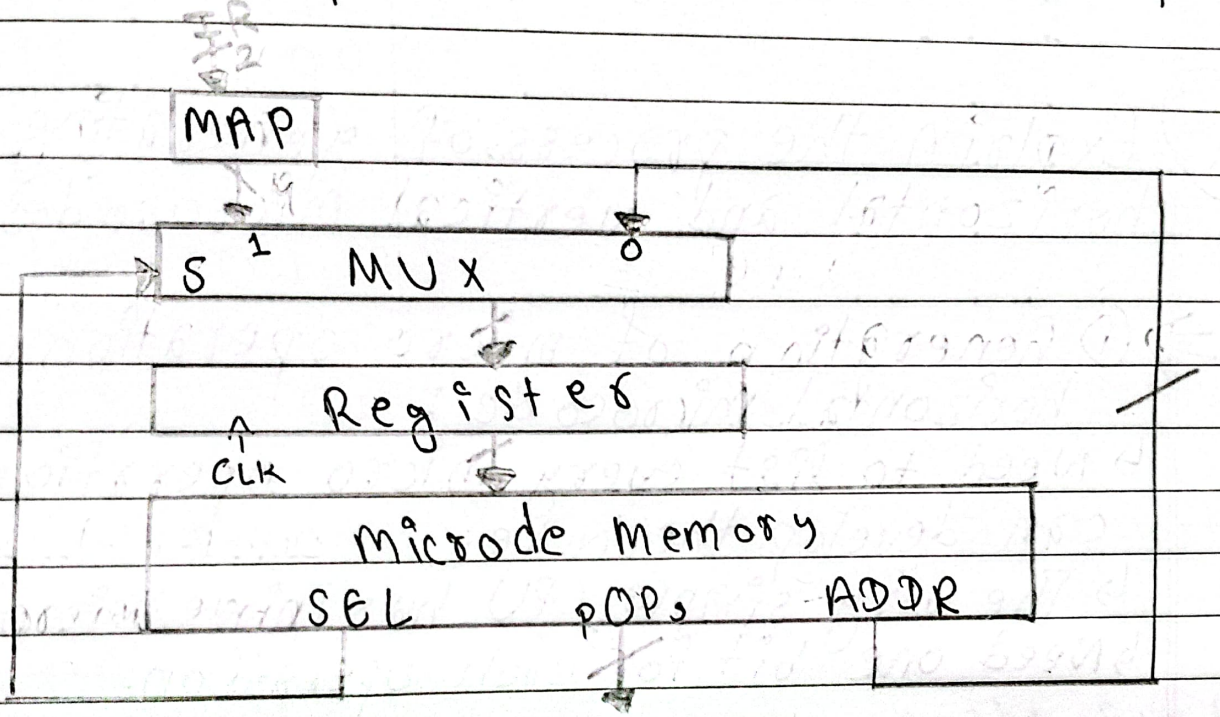
⇒



(3) What is microsequencer? Explain the control signa of a very simple CPU.

⇒ A microsequencer is a component or circuitry within a computer's central processing unit (CPU) that controls the execution of microinstructions. It retrieves, decodes, and sequences the microinstructions, which are low-level instructions that specify the individual operations performed by the CPU. The microsequencer generates the necessary control signals to execute the microinstructions in the correct order, ensuring proper operation of the CPU.

→ Consider the state diagram in the CPU design.
↳ In this only two possible next addresses are used; the opcode mapping and an absolute jump.
↳ fetch 3 goes to one of the four execute routines; implemented via the mapping input.
↳ Remaining states must go to next specific address; implemented via absolute jump.



↳ MUX is used to select one of the two possible address
↳ Select bit generated by microsequencer is used to choose correct next address.
↳ A total of 9 states in the state diagram, and hence 4-bits needed to represent them.

→ Control Signal values for the very simple CPU:

ARLOAD	ARPC V AIDR
PCLOAD	PCDR
PCINC	PCIN
DRLOAD	DRM
	PLUSVAND

ACINC	ACIN
IRLOAD	AIDR
ALUSEL	AND
MEMBUS	DRM
PCBUS	ARPC
DRBUS	AIDR V PCDR V PLUS V AND
READ	DRM

(4) Explain the process of generating horizontal and vertical microcodes.

⇒ (1) Generation of micro operation using horizontal microcode:

↳ Need to list every micro operation so we can develop the code.

↳ The very simple CPU has nine micro operation

↳ Need one bit for each operation.

Mnemonics	Micro Operation
ARPC	AR ← PC
ARDR	AR ← DR[5...0]
PCIN	PC ← PC+1
PRDR	PC ← DR[5...0]
DRM	DR ← M
IRDR	IR ← DR[7...6]
PLUS	AC ← AC + DR
AND	AC ← AC ^ DR
ACIN	AC ← AC+1

→ Optimized horizontal microcode for the very simple microsequencer

State	Address	SEL	ARPC	AIDR	PCIN	PCDR	DRM	PLUS	AND	ACIN	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	0001
FETCH2	0001 (1)	0	0	0	1	0	1	0	0	0	0010
FETCH3	0010 (2)	1	0	1	0	0	0	0	0	0	XXXX
ADD1	1000 (8)	0	0	0	0	0	1	0	0	0	1001
ADD2	1001 (9)	0	0	0	0	0	0	1	0	0	0000
ADD1	1010 (10)	0	0	0	0	0	1	0	0	0	1011
AND2	1011 (11)	0	0	0	0	0	0	0	1	0	0000
JMP1	1100 (12)	0	0	0	0	1	0	0	0	0	0000
INC1	1110 (14)	0	0	0	0	0	0	0	0	1	0000

(11) Generation of microoperation using vertical MicooCode:

→ Micro-operation field assignments & values

M1	
Value	Micro-operation
000	NOP
001	DRM
010	ARPC
011	AIDR
100	PCDR
101	PLUS
110	AND
111	ACIN

M2	
Value	Micro-Operation
0	NOP
1	PCIN

Vertical microcode for the very simple microsequencer

State	Address	SEL	M1	M2	ADDR
FETCH1	0000 (0)	0	010	0	0001
FETCH2	0001 (1)	0	001	1	0010
FETCH3	0010 (2)	1	011	0	XXXX
ADD1	1000 (8)	0	001	0	1001
ADD2	1001 (9)	0	101	0	0000
AND1	1010 (10)	0	001	0	1011
AND2	1011 (11)	0	110	0	0000
JMP1	1100 (12)	0	100	0	0000
INC1	1110 (14	0	111	0	0000

6) Compare Hardwired control unit & microprogrammed

=>

Hardwired Control Unit	Microprogrammed Control Unit
The control unit whose control signals are generated by the hardware through a sequence of instruction is called a hardwired control unit.	(i) The control unit whose control signals are generated by the data stored in control memory and constituted program on the small scale is called a microprogrammed control unit.
(ii) The control logic of a hardwired control unit is implemented with gate, flip flops, decoders etc.	(ii) The control logic of a micro-program control is the instructions that are stored in control memory to initiate the required sequence of microopentions
(iii) Wiring changes are made in the hardwired control unit if there are any changes required in the design.	(iii) Changes in a microprogrammed control unit are done by updating the microprogram in control memory.
(iv) It is faster and known to be complex structure	(iv) Slow compared but are simple

~~5. Design a CPU that meets~~ :

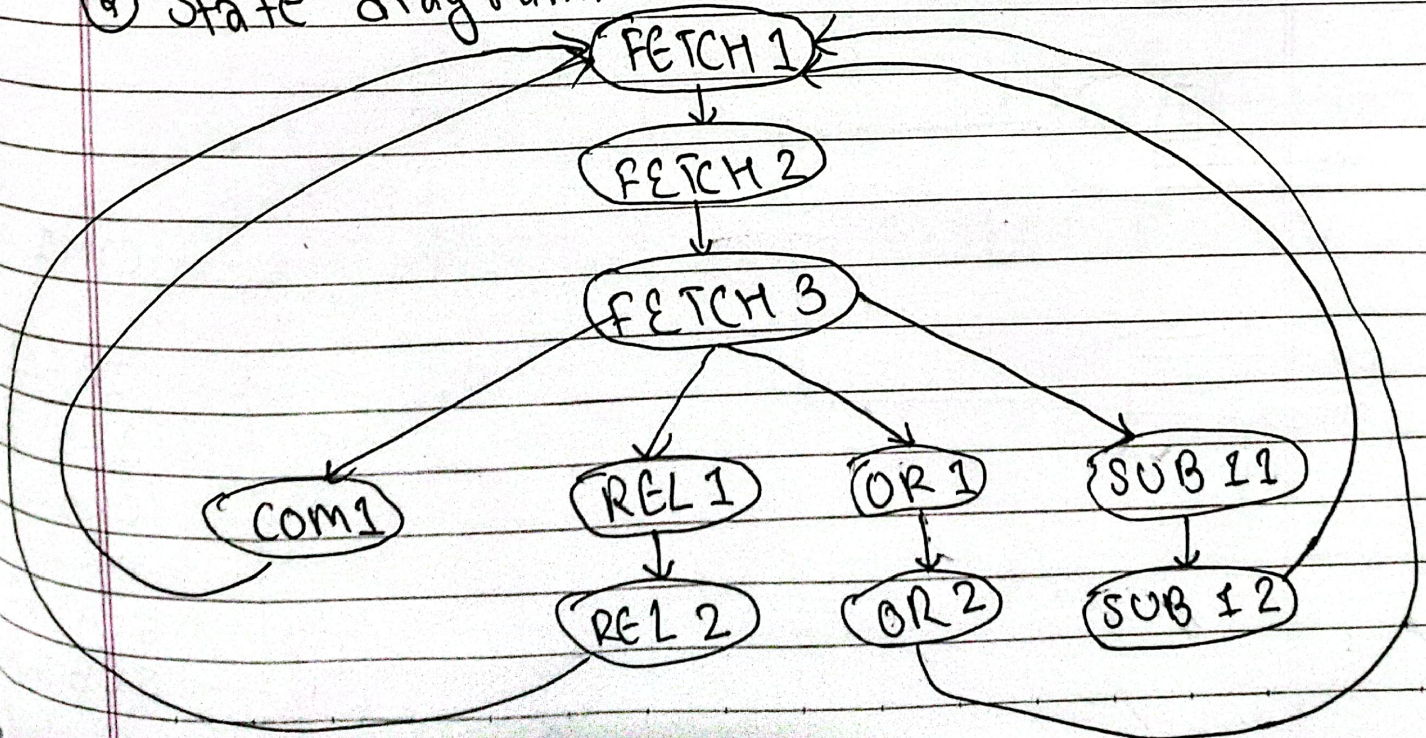
5. The instruction set of a very simple CPU is given as:

Instruction	Instruction Code	Operation
COMP	00xxxxxx	AC ← AC'
JREL	01AAAAAA	PC ← PC + AAAAAA
OR	10AAAAAA	AC ← AC v m[AAAAAA]
SUB	11AAAAAA	AC ← AC - m[AAAAAA] - 1

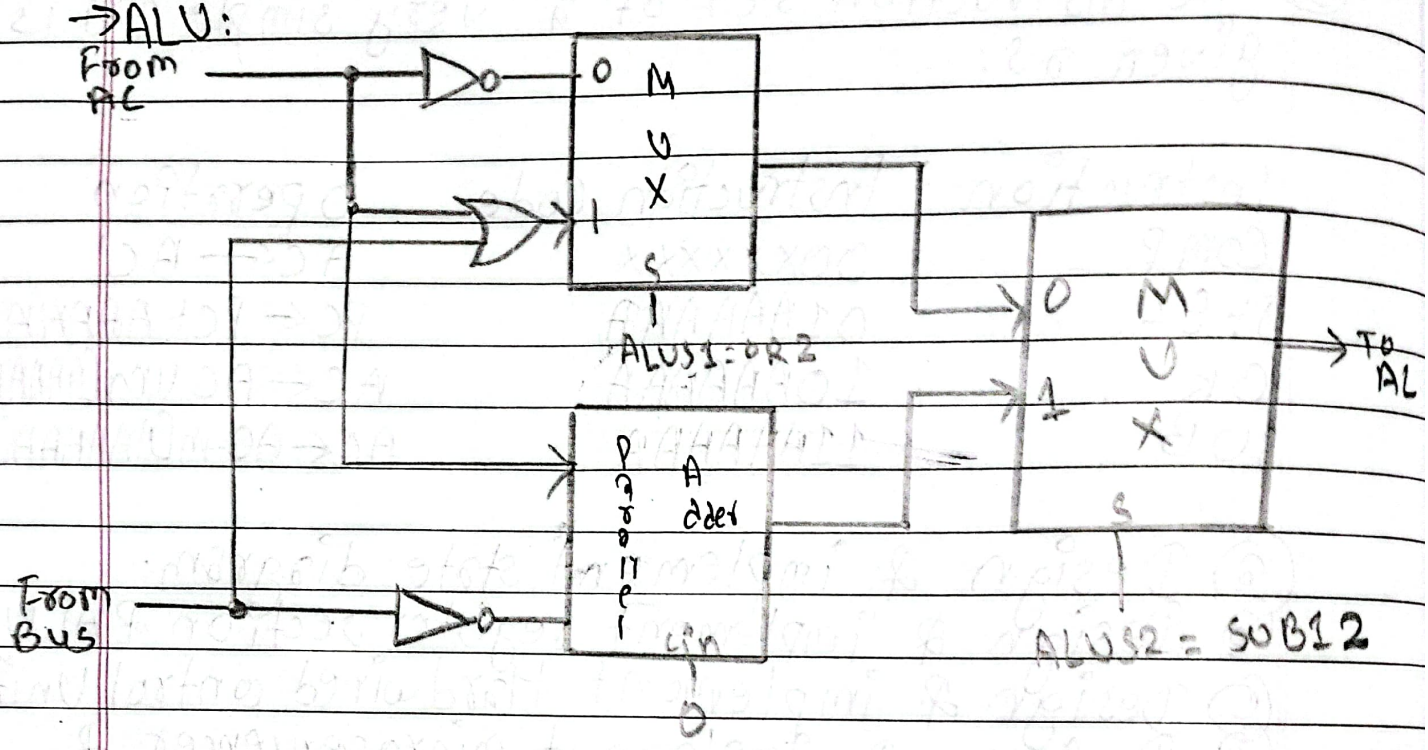
(a) Design & implement state diagram.
(b) Design & implement register section & ALU.
(c) Design & implement Hardwired control unit
(d) Design & implement microsequencer & control unit.

⇒ Solution:

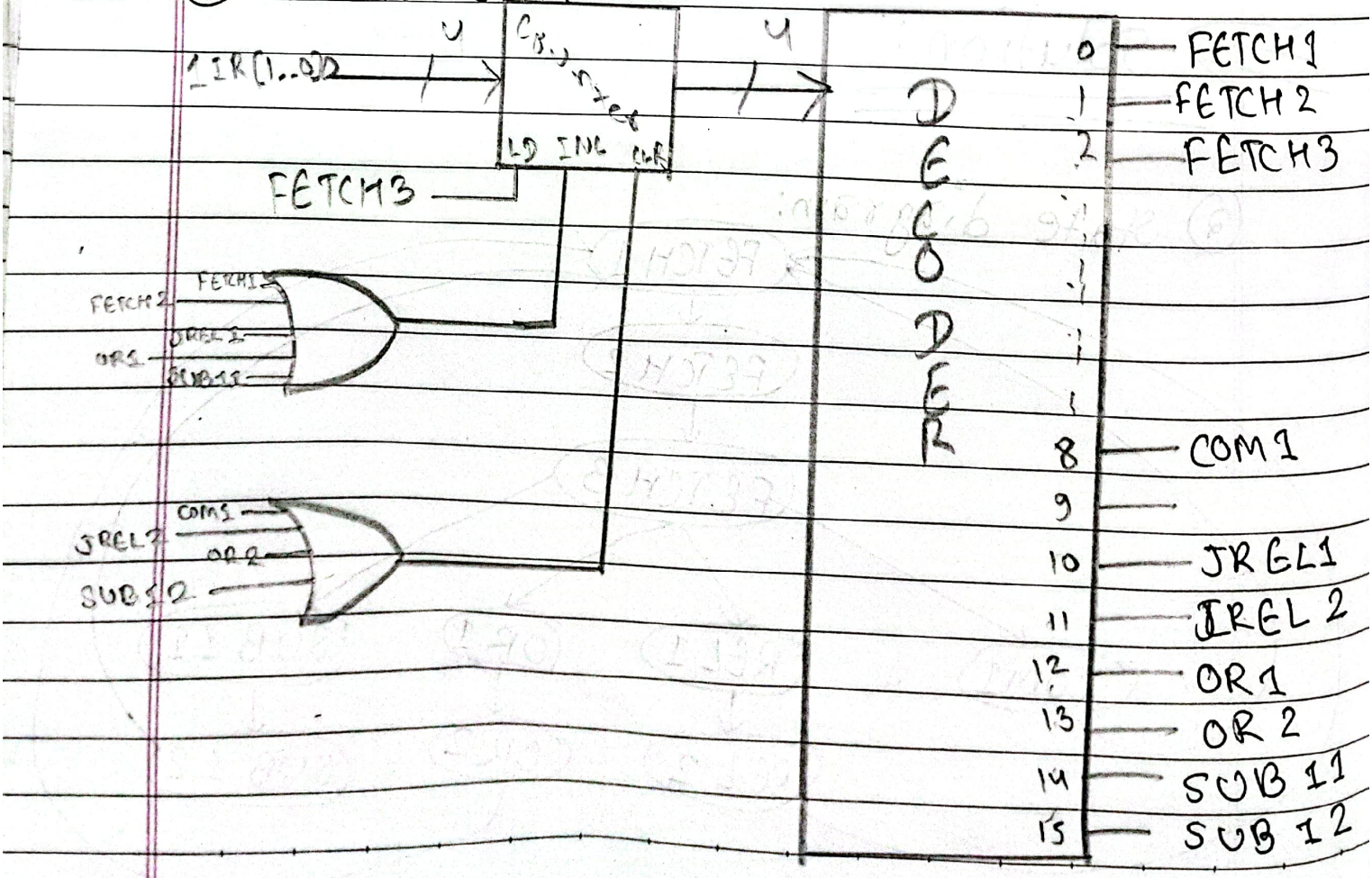
(a) State diagram:



(b): Register section & ALU:



(c) Control Unit:



(12) Explain the Booth's algorithm for division with an example.

⇒ Here,
let's take an example,
For 112 ÷ 7 with n=4, UU = 0111 0000 and X = 0111. Since, U ≥ X, the algorithm exists immediately. Had it proceeded, it should have produced a result of 16 (10000) and a remainder of 0. The value 10000 (cannot be stored in a 4-bit register), this is the reason for the overflow;

Now,

G1: FINISH ← 1, OVERFLOW ← 1
2: Y ← 0, C ← 0, OVERFLOW ← 0, i ← n
3: shl (CUU), shl (Y), i ← i-1
(C+G) 4: Y_0 ← 1, U ← U + X' + 1
Z'4: GOTO 3
Z4: FINISH ← 1

→ Trace of the RTL code for the operation 147 ÷ 13.
Here, U = 1001, V = 0011, X = 1101 & n = 4

→

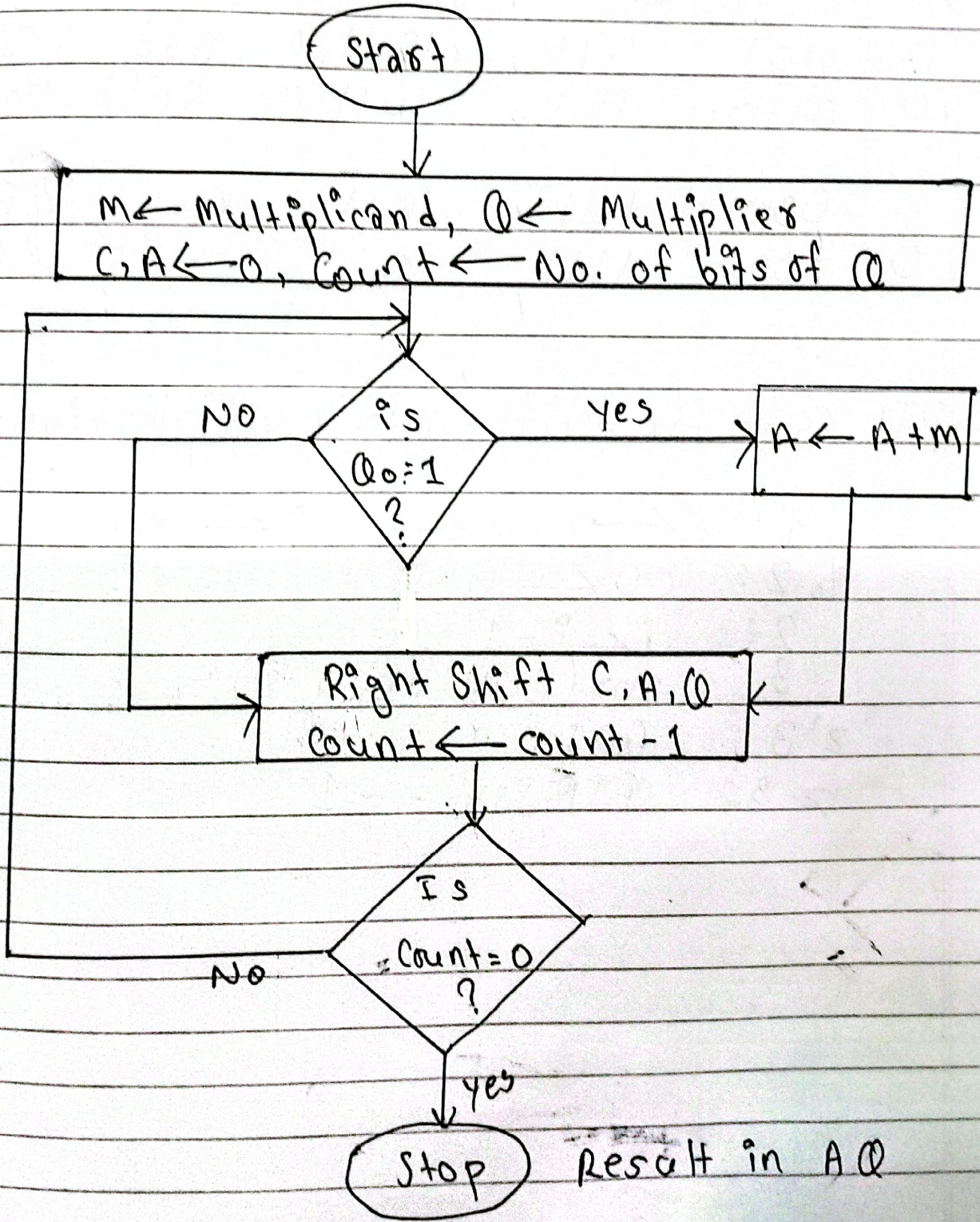
Conditions	Micro-operations	i	C	U	V	Y	Z	Finish
START		x	x	1001	0011	xxxx		
1	NONE							
2	Y ← 0; C ← 0, OVERFLOW ← 0, i ← 4	4	0			0000	0	
3	shl(CUV), shl(Y) i ← i - 1	3	1	0010	0110	0000	0	
(C+G)4,2'4	Y0 ← 1, U ← U + x' + 1, GOTO 3			0101		0001		
3	shl(CUV), shl(Y) i ← i - 1	2	0	1010	1100	0010	0	
z'4	GOTO 3							
3	shl(CUV), shl(Y) i ← i - 1	1	1	0101	1000	0100	0	
(C+G)4,2'4	Y0 ← 1, U ← U + x' + 1, GOTO 3			1000		0101	1	
3	shl(CUV), shl(Y) i ← i - 1	0	1	0001	0000	1010	1	
(C+G)4,2'4	Y0 ← 1 U ← U + x' + 1, FINISH ← 1			0100		1011		1

(7) Explain the shift add multiplication algorithm with an example. Also write down the RTL.

⇒ Shift add mutiplication:

⇒ multiplication of unsigned numbers.

⇒ multiplication can be performed by repeated mutiplicand and shifting.



C	A	Q	M		
0	0000	1101	1011	Initial values	
0	1011	1101	1011	Add	First cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second cycle
0	1101	1111	1011	Add	Third cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

→ RTL code for shift-add multiplication:

```
1:  C ← 0, U ← 0, i ← n
2:  CU ← U + x
2:  i ← i - 1
3:  shr(CUV), cir(Y)
3:  GOTO 2
3:  FINISH ← 1
```

Example: $CU \leftarrow X+Y$; X=1101 & Y=1011

Conditions	Micro-operations	i	C	U	U	Y	Z	FINISH
START		x	x	xxxx	xxxx	1011		0
1	$C \leftarrow 0, U \leftarrow 0, i \leftarrow 4$	4	0	0000			0	
Y_0 2,2	$CU \leftarrow U + X$ $i \leftarrow i-1$	3	0	1101			0	
3, z'3	shr(CUU), cir(Y) GO TO 2		0	0110	1xxx	1101		
Y_0 2,2	$CU \leftarrow U + X$, $i \leftarrow i-1$	2	1	0011			0	
3, z'3	shr(CUU), cir(Y) GO TO 2		0	1001	11xx	1110		
2	$i \leftarrow i-1$	1					0	
3, z'3	shr(CUU) cir(Y) GOTO 2		0	0100	111x	0111		
Y_0 2,2	$CU \leftarrow U + X$, $i \leftarrow i-1$	0	1	0001			1	
3, z3	shr(CUU), cir(Y) FINISH $\leftarrow$ 1		0	1000	1111	1011		1

(8) Trace the shift add algorithm for 10X6.

⇒ Here,
for shift add algorithm for 10X6;
we have,
X = 1010 , Y = 0110
Now,

Function	i	C	U	V	Y	Comments
U=0			0000	0000	0110	
if Y_0=1	1	0	1010			Y_0=0
Shr(CUV)		0	0101	0000	0011	
Cir(Y)						
if Y_0=1	2	0	1111			Y_0=1, Add
Shr(CUV)						
[illegible]		0	0111	1000		
Cir(Y)					1001	
if Y_0=1	3	1	1001			Y_0=1, Add
Shr(CUV)		0	1100	1100		
Cir(Y)					1100	
if Y_0=1	4					Y_0=0
Shr(CUV)		0	0110	0110		
Cir(Y)					0110	original

RTL:

1: $U \leftarrow 0, i \leftarrow x$

$Y_0$2: $CU \leftarrow U + x$

2: $i \leftarrow i - 1$

3: Shr(CUV), Cir(Y)

2'3: GOTO 2

23: FINISH $\leftarrow$ 1

Under,

Z=1 when i=0 & 1, 2, & 3 are consecutive states

(*) (9) Trace the RTL code for shift add multiplication of 5×5.

⇒

Conditions	Micro-operations	i	C	U	V	Y	Z	Finish
START		x	x	xxxx	xxxx	0101		0
1	U ← 0, i ← 4	4		0000			0	
2	i ← i - 1	3					0	
3, z'3	shr(CUV), cir(Y), goto2		0	0000	0xxx	1010		
Yo 2, 2	CU ← U+X, i ← i-1	2	0	0101			0	
3, z'3	shr(CUV), cir(Y), goto2		0	0010	10xx	0101		
Yo 2, 2	CU ← U+X, i ← i-1	1	0	0111			0	
3, z'3	shr(CUV), cir(Y), goto2		0	0011	110x	1010		
Yo 2, 2	CU ← U+X, i ← i-1	0	0	1000			1	
3, z'3	shr(CUV), cir(Y), goto2		0	0100	0110	0101		1

Result: 5 * 5 = 25, or, 0101 * 0101 = 0100 0110

(10) Explain Booth's algorithm for multiplication of unsigned two's complement number's with example.

⇒ Booth Algorithm:

- It reduces the number of addition.
- Negative multiplier, positive multiplier are treated same.
- For negative numbers multiplication, the operation is performed by representing the negative numbers in it's 2's complement form.

ADVANTAGES:

⇒ When there are lot of ones, all will be cancelled and replaced with zeros.

⇒ More Zeros means, it reduces the addition (no operation is done with zeros)
⇒ But shift remains the same, can't be reduced.

(*) Example: -5×4
Perform multiplication using Booth algorithm.
Here,

$X = -5 = 01011$
$Y = 4 = 00100$
$\therefore X = 1011 \;, Y = 0100$

$x' = 0100$

$$\begin{array}{r} 01011 \\ 10100 \\ +\quad\;\; 1 \\ \hline 10101 \end{array}$$

Now,

Conditions	Micro-operations	i	U	U	Y	Y-1	Z	Finish
START		X	XXXX	XXXX	0100	X		0
1	U ← 0, Y-1 ← 0, i ← n	5	00000			0	0	
$Y_0 Y_{-1}'$ 2,2	U ← U + X' + 1	4	01101				0	
	i ← i - 1		11010					
3, Z'3	ashr(UY), cir(Y), Y_{-1} ← Y0		11010	1XXX	0010	1		
	goto 2							
$Y_0 Y_{-1}'$ 2,2	U ← U + X	3	00101				0	
	i ← i - 1							
3, Z'3	ashr(UY), cir(Y), Y_{-1} ← Y0		10010	11XX	0001	0		
	goto 2							
$Y_0 Y_{-1}'$ 2,2	U ← U + X' + 1	2	00111				0	
	i ← i - 1							
3, Z'3	ashr(UY), cir(Y), Y_{-1} ← Y0		10011	111X	1000	1		
	goto 2							
2	i ← i - 1	1						
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

→

3, z'3	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$		11001	1111X	01000	1	
	goto 2						
2	$i \leftarrow i-1$	0					1
3, z3	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$		11100	11111	00100		

Here UV = 11100 11111
There is end around carry so, we find 2's complement and place 've' sign before the equivalent decimal.

→ The RTL of Booth's algorithm for multiplication:

1: $U \leftarrow 0, Y_{-1} \leftarrow 0, I \leftarrow n$

$Y_0 \bar{Y}_{-1}$ 2: $U \leftarrow U + X' + 1$

$\bar{Y}_0 Y_{-1}$ 2: $U \leftarrow U + X$

2: $i \leftarrow i - 1$

3: ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$

z'3: goto 2

z3: FINISH $\leftarrow 1$

(11) ⇒ Done in 10 → example.