Chapter 05:microprogramed control unit

Hari K.C.

Department of Software Engineering

Gandaki College of Engineering and Science

Control unit
- Sequencing
- Execution

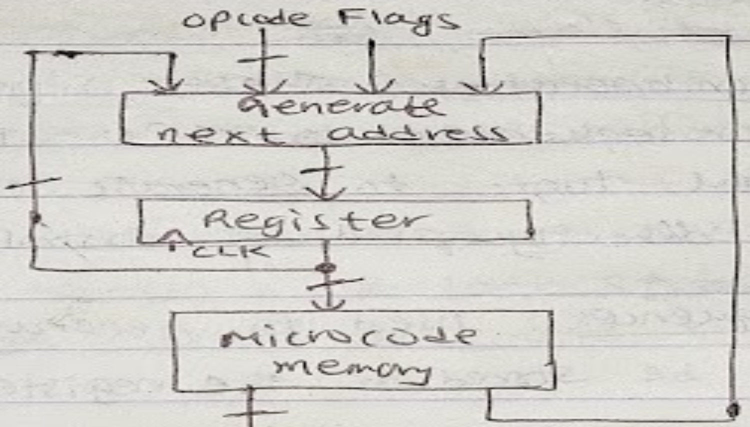
chapter 5
Control unit Design

Microsequencer or microprogrammed control unit.

In microprogrammed control unit, the control signals are generated & stores in lookup ROM, a microcode memory.
(Microsequencer generates control signals in correct sequence to perform micro-operations)

Microsequencer Design & operations

Miscrosequencer is designed as a finite state machine.



microoperation's

fig:- Generic microsequencer organisations control memory

Microsequencer stores it control signals in a lookup ROM. This control unit outputs the control signals values by reading data from lookup ROM

Microsequencer consist of 3 main parts:

i) microcode memory (control memory)

ii) next address generator

iii) register.

i) Microcode memory stores microcode or microprogram. It contists of microinstruction to fetch, decode & execute every instruction in microprocessor's instruction set.

ii) Next address generator ensures that microsequencer access the microinstructions in the order necessary to process each instruction in instruction set properly. It uses mapping logic to access correct execute routine for each instruction

T_1: MAR ← PC
T_2: MBR ← Memory
T_3: IR ← MBR
PC ← PC + 1

instruction = opcode + address field

⇒ The register stores a value that corresponds to one state in CPU's state diagram. It serves as address that is input to microcode memory. This memory outputs a microinstruction. (the contents of memory location for that address). All microinstructions gives the microcode or microprogram for CPU.

The microinstruction consists of several bit fields. It can be divided into two groups:-

i) micro operations
ii) sequencer

i) <u>micro operations</u>: These signals are output from microsequencer to CPU. They are input to combinatorial logic to generate CPU's control signals or they directly produce control signals.

ii) <u>sequencer</u>: used to generate the next address to be stored in the register

These bits, along with instructions opcode & flag values are input to combinational logic that generates address of next microinstruction

The "Generate next address block of microsequencer generates all possible next address & select correct next address to pass along to register

A microsequencer uses a parallel adder to generate the value of current address plus 1 as a possible next address ✓

The other possible next addresses are absolute address, mappic logic, & a microsubroutine ~~&~~ return address.

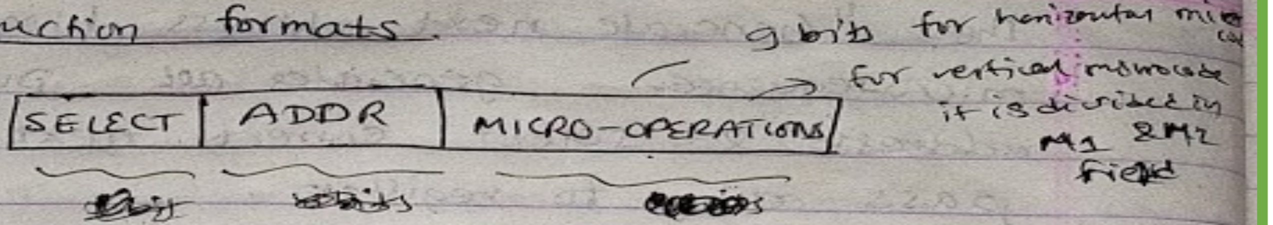
absolute address: for jump instructions.

mapping logic : maps opcode to the address

microsubroutine return address : for subroutine.

Every ~~m~~microsequencer uses the mapping address to go from ~~to~~ last state of fetch routine to correct execute routine.

* Microinstruction formats.


SELECT
ADDR
MICRO-OPERATIONS
9 bits for horizontal micro code
for vertical microcode it is divided in M1 & M2 field

i) The SELECT field determines the source of the address of next microinstruction. either mapping or absolute addr

For example:- consider the final microinstruction of a fetch routine, its select bits would instruct the next address generator to pass mapping address to register of microsequencer.

ii) ADDR field :- It specifies an absolute address. The microsequencer uses this address when performing an absolute jump i-e from the end of an execute routine to beginning of fetch routine. But the mapping logic doesn't use the bits of this field.

iii) MICRO-OPERATIONS field :- There are 3 methods to specify micro operations.

a) horizontal microcode
b) vertical microcode
c) Direct generations of control signals

Ow

a) Horizontal microcode :-
Firstly, listing every microoperation performed by CPU. (mnemonics are used)
Assign one bit in the microoperations field of microinstruction to each micro operation
This will result in large ~~microoperation~~ microinstruction.
For example:-
if a CPU performs 50 different micro operations, the micro-operation field uses 50 bits for every micro-operation.

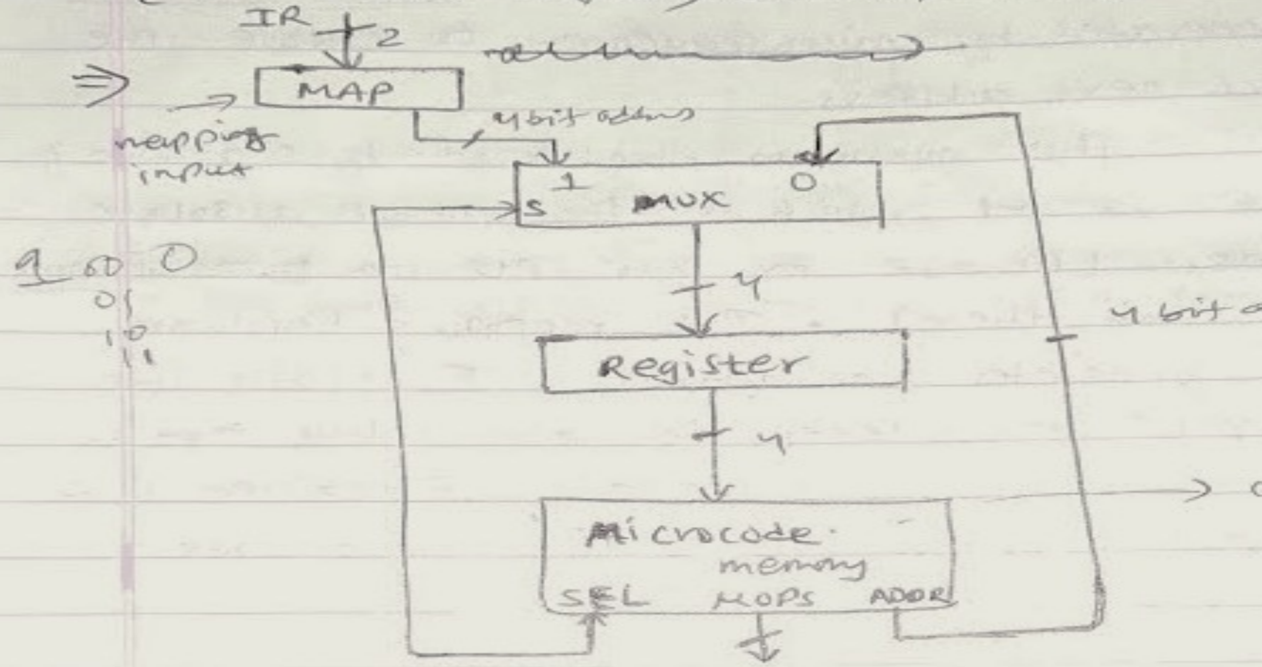
b) vertical micro code :- The number of bits in micro operation field can be reduce by using vertical microcode. In vertical microcode, the microoperations are grouped into fields. Each microoperation is assigned a unique encoded value in this field.
For example:- 16 microoperations can be encoded ~~with~~ using 4 bits, from 0000 to 1111

dew bar

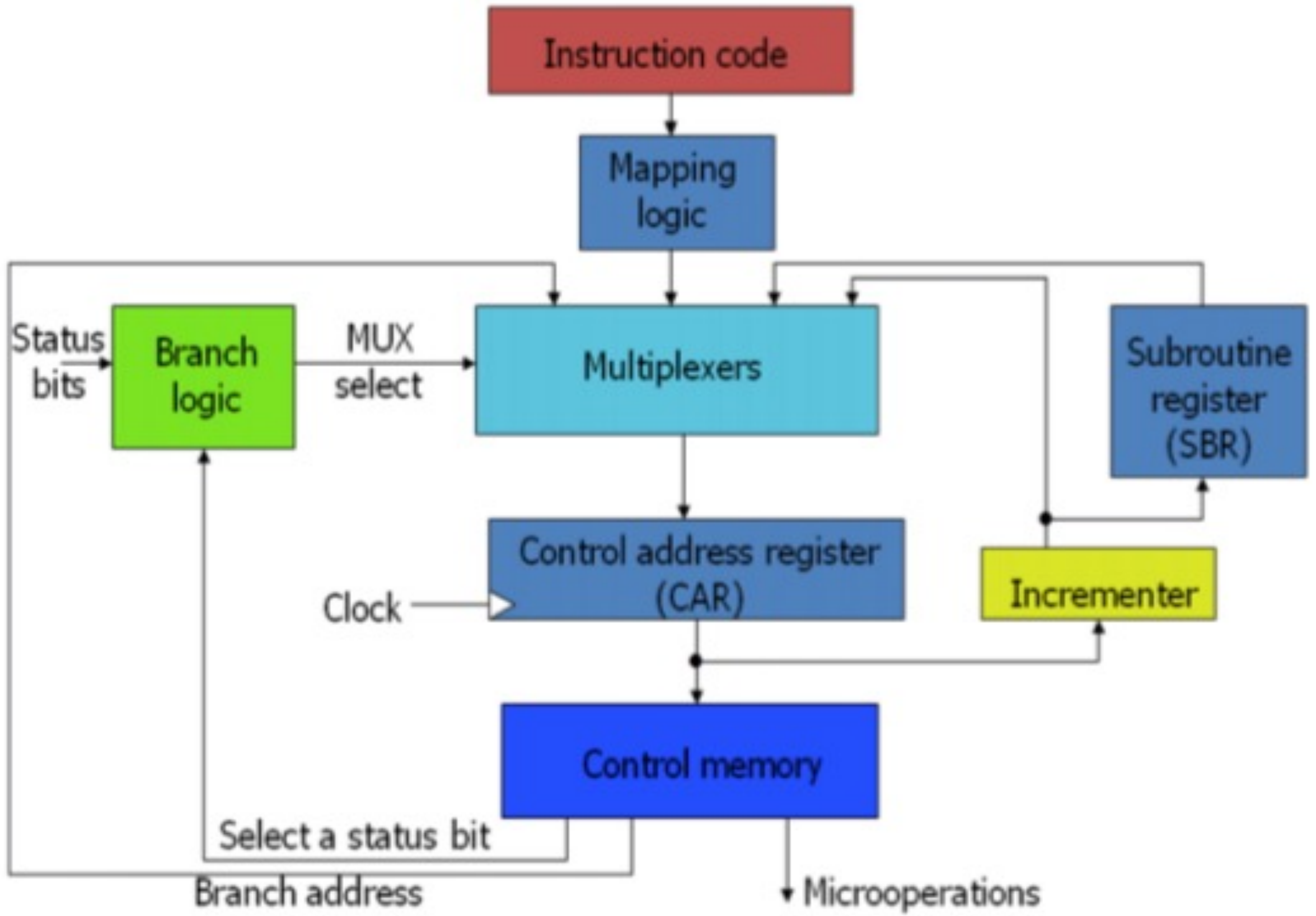
Vertical microinstruction require fewer bits than their equivalent horizontal micro instructions. However, microsequencers that use vertical microcode must include a decoder for each micro-operation field to generate actual micro-operation signals.

In both horizontal & vertical microcode, the CPU must convert micro-operation signals to the control signals that load, clear & increment registers, enable buffers & select the ~~signals~~ ~~that load~~ functions to be performed by ALU.

* Design & implementation of a very simple microsequencer
(control unit layout) control sequence generation & mapping logic



Here, only two possible next address are used : i) the opcode mapping & ii) absolute jump
The last State of Fetch cycle, Fetch 3 goes to one of the 4 execute routines (ADD, AND, JMP, INC) - This is implemented by mapping input.
The remaining states must each must go to one specific next state, which is implemented by using an absolute jump.
Since there are 2 possible addresses,


Instruction code
Mapping logic
Status bits
Branch logic
MUX select
Multiplexers
Subroutine register (SBR)
Control address register (CAR)
Clock
Incrementer
Control memory
Select a status bit
Branch address
Microoperations

the microsequencer select one of them. So, there is a multiplexer to do this. The select bit is generated by microsequencer to choose the correct next address.

The minimum number of bits needed to select is 4, which is the size of absolute address. (Because for our CPU, a total of nine states are there). The mapping hardware also generates an address of 4 bits wide. The o/p of multiplexer is also 4 bits that is input to register & output of register that is input to address inputs of microcode memory.

9 states
$2^3 = 8$
$2^4 = 16$

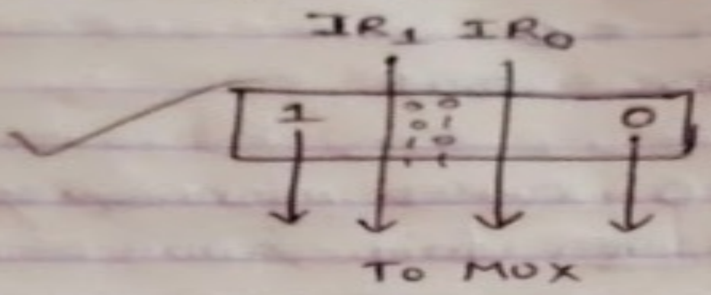
*** Generating correct sequence & designing the mapping logic.**

⇒ For this CPU, the microsequencer use the same mapping function that was used for hardwired control unit.

1 IR[1--0] 0

This produces address of 1000, 1010, 1100 & 1110 (8, 10, 12 & 14) for ADD1, AND1, JMP1 & INC1.

The mapping logic will be



The address for remaining states can be assigned somewhat randomly. Here we assign consecutive states to consecutive location in microcode memory (control memory)

State	Address
fetch1	0000 (0)
fetch 2	0001 (1)
fetch 3	0010 (2)
ADD1	1000 (8)
ADD2	1001 (9)
AND 1	1010 (10)
AND 2	1011 (11)
JMP1	1100 (12)
INC 1	1110 (14)

SEL=0

ADD1 → ADD2
AND1 → AND2

To go from fetch1 to fetch 2,
SEL = 0
& ADDR = 0001

Setting SEL=0 causes microsequencer to get its next address from ADDR field. & setting ADDR field to 0001 causes it to go to location corresponding to state fetch 2

But,

fetch 3 must map to correct execute routine so it requires SEL=1 to select mapping address. Now, the contents of ADDR field are not used in microinstruction, it doesn't ~~[illegible]~~ matter what value it has.

Microcode

next address

State	Address	SEL	ADDR
fetch1	0000	0	0001
fetch2	0001	0	0010
fetch3	0010	1	XXXX
ADD1	1000	0	1001
ADD2	1001	0	0000
AND1	1010	0	1011
AND2	1011	0	0000
JMP 1	1100	0	0000
INC 1	1110	0	0000

Generating the microoperation using horizontal microcode

Microsequencer has two tasks i) to generate the correct microoperations & ii) to follow the correct sequence of states. ~~The first~~ one ~~task~~ task is completed. Now, the other task is to generate correct microoperations, and associated control signals. This can be done by horizontal microcode, vertical microcode & direct generation of control signals.

In horizontal ~~microinstruction~~ microcode, each micro~~instruction~~operation is represented by <u>one bit</u> in each microinstruction. The greater the number of microoperation, the larger the microcode will be.

The micro operations & their mnemonics are:—

Mnemonic	Micro-operation
ARPC	AR ← PC
ARDR	AR ← DR [5...0]
PCIN	PC ← PC+1
PCDR	PC ← DR [5...0]
DRM	DR ← M
IRDR	IR ← DR [7...6]
PLUS	AC ← AC + DR

AND	$AC \leftarrow AC \wedge DR$
ACIN	$AC \leftarrow AC + 1$

~~Now, the primary horizontal~~

Since there are 9 microoperations, each word of microcode requires 9 bits to represent them, 1 bit per microoperation.
A value of 1 means the microoperation is to occur & a value of 0 means that it does not.
Therefore, the horizontal microcode for very simple microsequencer.

see micro operat

State	Address	SEL	ARPC	ARDR	PCIN	PCDR
fetch1	0000 (0)	0	1	0	0	0
Fetch 2	0001 (1)	0	0	0	1	0
fetch3	0010 (2)	1	0	1	0	0
ADD1	1000 (8)	0	0	0	0	0
ADD2	1001 (9)	0	0	0	0	0
AND1	1010 (10)	0	0	0	0	0
AND2	1011 (11)	0	0	0	0	0
JMP1	1100 (12)	0	0	0	0	1
INC1	1110 (14)	0	0	0	0	0

9bits

fetch1 : $AR \leftarrow PC$ Moperation हु [illegible]·so,
fetch1 मा 1 [illegible] -

ARDR & IRDR have same value so, we can use one output to drive both microoperations. Let AIDR represent both.

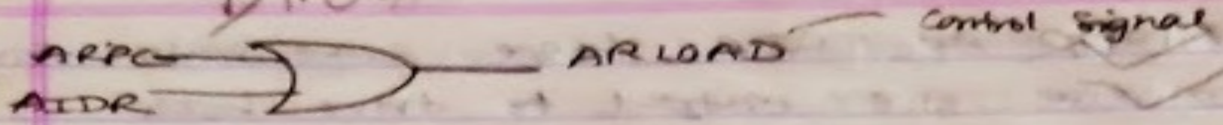
~~So, 9 microoperations are reduced to 8 μP.~~

For AR ← PC microoperation, this need 9 bits (as shown in ARPC column). but mostly bits are 0 (inactive).

control signals next state

DRM	IRDR	PLUS	AND	ACIN	ADDR
0	0	0	0	0	0001
1	0	0	0	0	0010
0	1	0	0	0	XXXX
1	0	0	0	0	1001
0	0	1	0	0	0000
1	0	0	0	0	1011
0	0	0	1	0	00[illegible]
0	0	0	0	0	0000
0	0	0	0	1	0000

Direct Generation of Control Signals



The control signal values are

Control signals

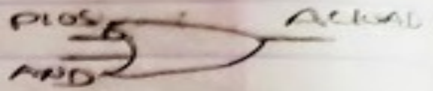
ARLOAD → ARPC ∨ AIDR

PCLOAD → PCDR

PCINC → PCIN

Data register load

DRLOAD → DRM

ACLOAD → PLUS ∨ AND 

ACINC → ACIN

[illegible]

IRLOAD → AIDR

ALUSEL → AND

MEMBUS → DRM

PCBUS → ARPC

DRBUS → AIDR ∨ PCDR ∨ PLUS ∨ AND

READ → DRM

→ Generating micro-operation using vertical microcode

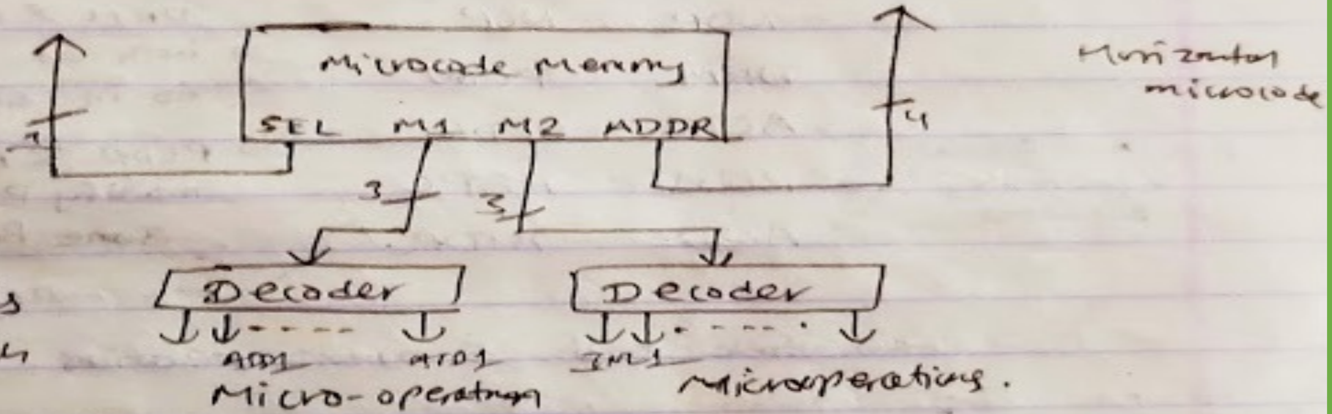
Vertical microcode reduce the number of ~~bits~~ in section of microcode.

In vertical microcode, the microoperations are grouped into fields such that no more than one microoperation in a field is active during ~~any~~ state. Then a unique field value is assigned to each microoperation in the field

For e.g:- a field with 8 different micro-operations would require 3 bits each value from ~~000~~ to 111 would be assigned to one of the eight micro-operations. The microoperation field bits are o/p from microcode memory to a decoder

$2^3 = 8$

3×8 decoder



M1 & M2 are bit fields of 3 bits each

(fig: Generation of μoperations from vertical microcode)

a) when two microoperations occurs during the same state, assign them to different fields.

for e.g:- DRM & PCIN both occur during fetch 2 states so, must be assign to different fields, one in M1 field & other in ~~[illegible]~~ M2

b) Include a NOP (No operation) if necessary, when no microoperation is active.

c) Group together micro-operation in same field that modify same register

The CPU requires atleast two fields for its microoperation. labeled as M1 & M2.

M1	M2
NOP	NOP
DRM	PCIN
ACIN	PCDR
PLUS	ARPC
AND	AIDR.

DRM & PCIN both at same state so in diff. field.

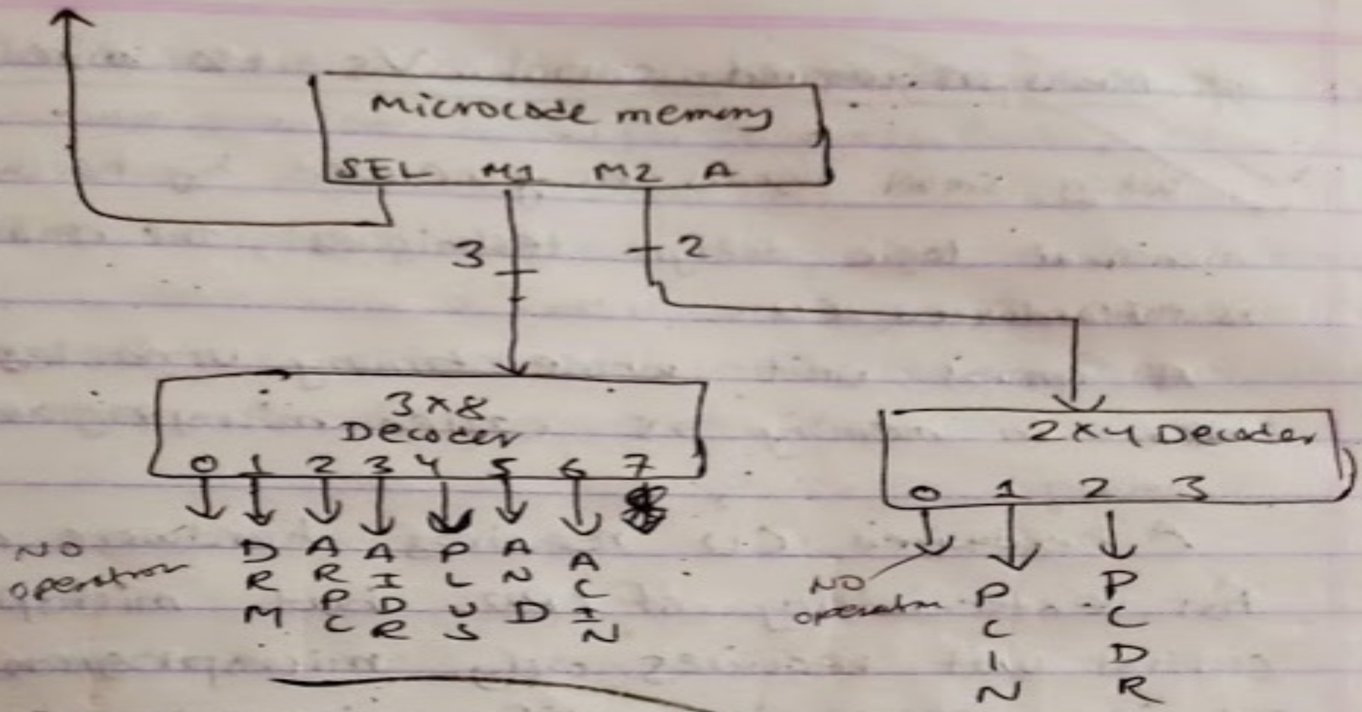
PCDR & PCIN both modify PC so in same fields

both modify AR

PLUS, AND & ACIN both modify accumulator

Each field has 5 microoperations.

	M1	M2
000 →	NOP	NOP → 00
001 →	DRM	PCIN → 01
010 →	ARPC	PCDR → 10
011 →	AIDR	
100 →	PLUS	
101 →	AND	
110 →	ACIN	



Reducing the no. of microinstructions

The number of microinstructions can be reduced by ~~following~~ ways:—

i) Using microsubroutines to combine repeated microoperations into a single block of microinstructions which are accessed by two or more execute routines.

ii) Using microcode jumps to access microinstructions shared by two or more routines.

* Microprogrammed control Vs Hardwired Controlun

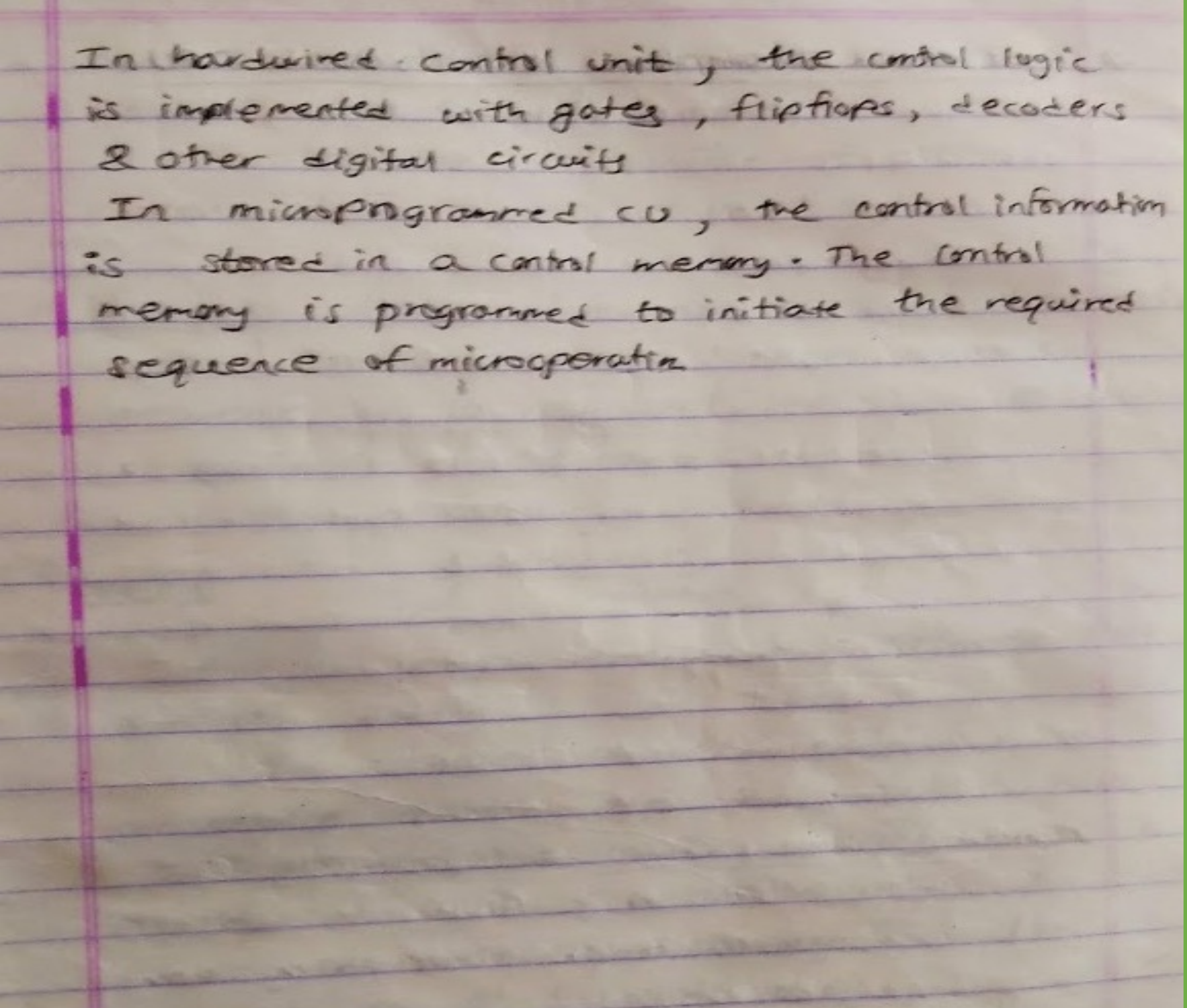
when control signals are generated by hardware using conventional logic design techniques, the control unit is Hardwired CU.

A control unit whose binary variables are stored in memory is called microprogrammed CU.

A hardwired CU requires hardware modification for extensibility of CPU but microprogrammed control unit requires only microprogram changes It is easier to modify microcode than to redesign hardware.

A hardwired CU has faster instruction execution than microprogrammed CU. (clock speed is high)

Microsequencer are less complex than hardwired CU.



In hardwired control unit, the control logic is implemented with gates, flipflops, decoders & other digital circuits

In microprogrammed CU, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperatin

The End