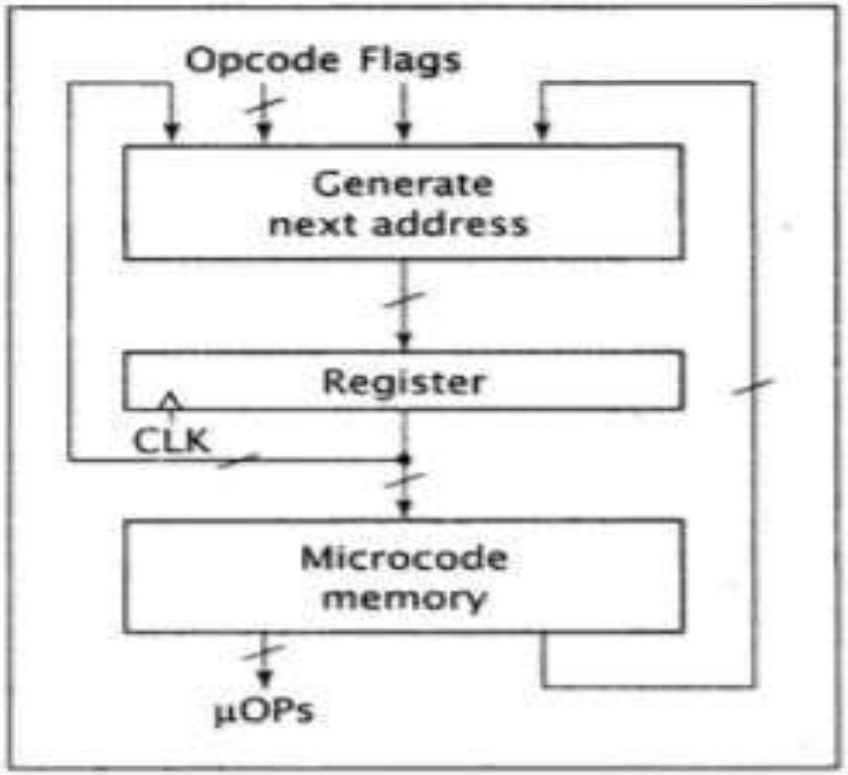
CHAPTER-5

Control Unit Design

Basic micro-sequence design and operation

Generic microsequencer organization

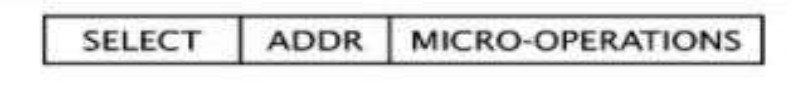


Contd...

- REGISTER stores a value that corresponds to 1-state in the CPU state diagram. It serves as the address i.e. input to the micro-code memory.
- MICRO-CODE memory outputs a micro-instruction and the content of memory location for this address.
  =>MICRO-INSTRUCTION consists of several bit of fields which can be broadly classified into two groups.
  A. micro-operation that generates the control signal.
  B. generate next address to be stored in the register. These bits, along with the instruction opcode and flag values are input to the combinational logic that generates the address of next micro-instruction.
- The 'Generate Next Address' block of micro-sequencer typically generate all possible next address and then select correct next address to pass to the register.
  =>One possible value of the next address is the current address + 1.
  =>Another possible address is an absolute address supplied by the micro-code. For example, at the end of every execute routine, the micro-sequencer must jump back to the beginning of fetch routine.

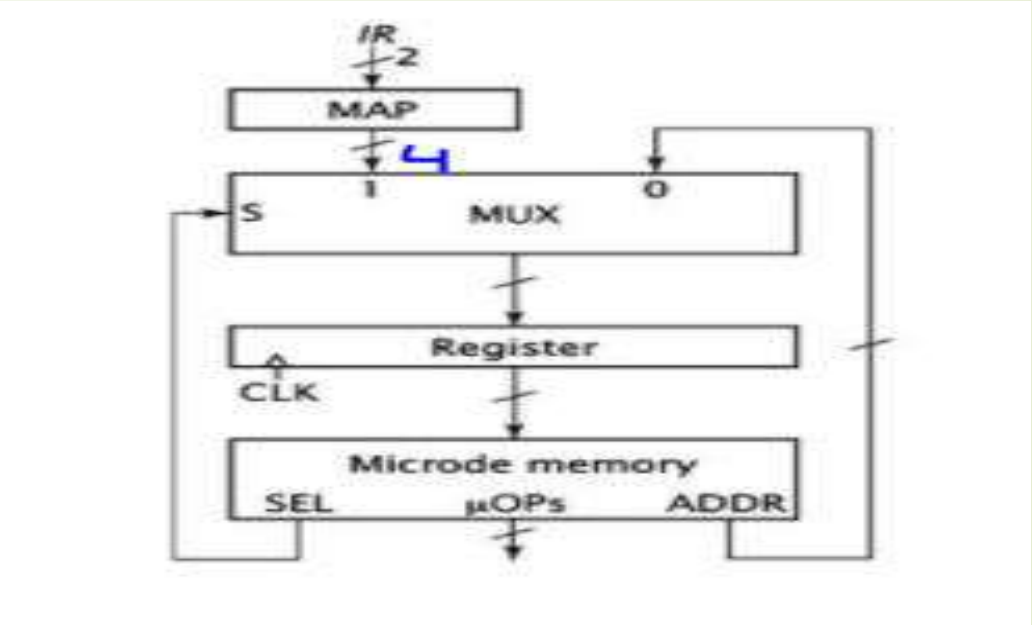
Microinstruction format

- SELECT field: determines source of address of next microinstruction. It doesn't specify the actual address, but only the source of the address.
- ADDR field: specifies an absolute address. Used when performing absolute jump.
  => Microinstructions that specify another source for the next address, such as mapping address do not use bits of this field.
- MICRO-OPERATIONS field: three primary methods.
  => horizontal microcode
  => vertical microcode
  => Direct generation of control signals



Design and implementation of a very simple microsequencer:

- Consider the state diagram in the CPU design.
- In this only two possible next addresses are used; the opcode mapping and an absolute jump.
- Fetch3 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.



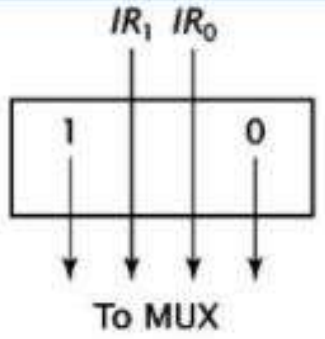
Contd...

- 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 sequence generation and mapping logic

? The microsequencer uses the mapping function namely 1IR[1..0]0 which provides the address of 1000,1010, 1100, 1110 for ADD1, AND1, JMP1 and INC1 respectively.

State	Address
FETCH1	0000 (0)
FETCH2	0001 (1)
FETCH3	0010 (2)
ADD1	1000 (8)
ADD2	1001 (9)
AND1	1010 (10)
AND2	1011 11)
JMP1	1100 (12)
INC1	1110 (14)



Assign each state of the finite state machine to an address in microcode

By choosing the addresses wisely we can minimize connections

Contd...

State	Address	SEL	ADDR
FETCH1	0000 (0)	0	0001
FETCH2	0001 (1)	0	0010
FETCH3	0010 (2)	1	xxxx
ADD1	1000 (8)	0	1001
ADD2	1001 (9)	0	0000
AND1	1010 (10)	0	1011
AND2	1011 11)	0	0000
JMP1	1100 (12)	0	0000
INC1	1110 (14)	0	0000

- SEL = 0 will get next address from ADDR field
- FETCH3 must map to correct execute routine, so SEL = 1 to use that address

? For eg, to go from Fetch1 to Fetch2, the microsequencer will specify SEL=0 and ADDR = 0001

Generation of micro operation using horizontal microcode

(micro operations and their mnemonics for the very simple microsequencer)

- Need to list every micro operation so we can develop the code.
- The Very Simple CPU has nine micro operations.
- 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

Contd...

(Preliminary horizontal microcode for the very simple microsequencer)

Preliminary horizontal microcode for the Very Simple Microsequencer

State	Address	SEL	ARPC	ARDR	PCIN	PCDR	DRM	IRDR	PLUS	AND	ACIN	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	0	0001
FETCH2	0001 (1)	0	0	0	1	0	1	0	0	0	0	0010
FETCH3	0010 (2)	1	0	1	0	0	0	1	0	0	0	XXXX
ADD1	1000 (8)	0	0	0	0	0	1	0	0	0	0	1001
ADD2	1001 (9)	0	0	0	0	0	0	0	1	0	0	0000
AND1	1010 (10)	0	0	0	0	0	1	0	0	0	0	1011
AND2	1011 (11)	0	0	0	0	0	0	0	0	1	0	0000
JMP1	1100 (12)	0	0	0	0	1	0	0	0	0	0	0000
INC1	1110 (14)	0	0	0	0	0	0	0	0	0	1	0000

Contd..

(optimize)

- For all states, ARDR and IRDR have the same value
- Don't need two outputs
- Use one output to drive both micro operations, AIDR, combining
  - AR ← DR[5..0] and IR ← DR[7..6]

Contd...

(optimized horizontal microcode for the very simple microsequencer)

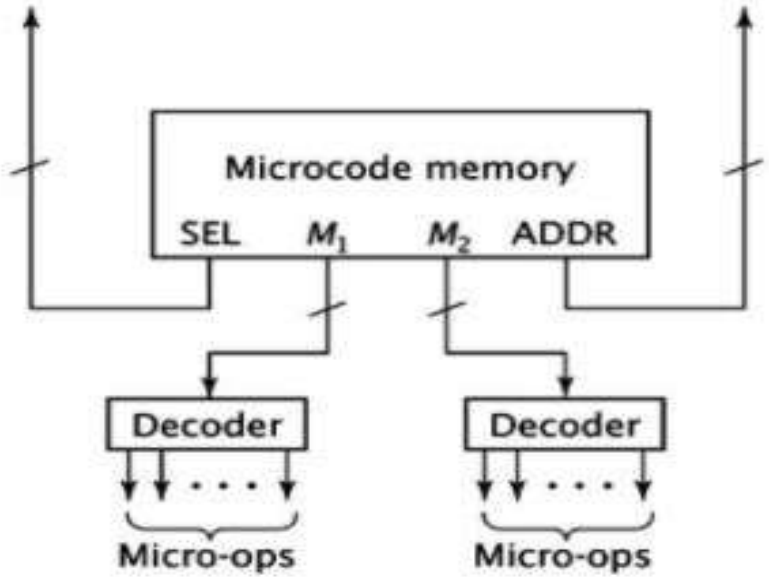
Optimized horizontal microcode for the Very Simple Microsequencer

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
AND1	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

Generation of micro operation using vertical microcode

(vertical microcode offers a way to reduce a number of bits)

- Horizontal code table consists mostly of zeros – 85%
- Vertical micro operations are grouped into fields
- Can use decoders to generate the instructions



Contd..

(guidelines to design microsequencer using vertical microcode)

1. Whenever two micro operations occur during the same state, assign them to different fields.
2. Include a NOP in each field if necessary.
   - Needed because some value must be output every cycle
3. Distribute the remaining micro operations to make best use of the micro operation field bits.
4. Group together micro operations that modify the same registers in the same fields.

Contd...

- DRM and PCIN both occur in FETCH2, must be assigned to different fields
  - Will need at least two fields for the Very Simple CPU

M1	M2
NOP	NOP
DRM	PCIN

- Since PCIN and PCDR both modify PC, add PCDR to M2
- Then arbitrarily assign the remaing micro operations to the fields, keeping micro operations that change the same register in the same field.

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

Contd...

Micro-operation field assignments and 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

- To optimize, it is best to maximize to a power of two, so is possible, put eight micro operations in M1 and two in M2.
- M1 requires 3 bits, M2 only one bit, for total of 4 bits.

Contd...

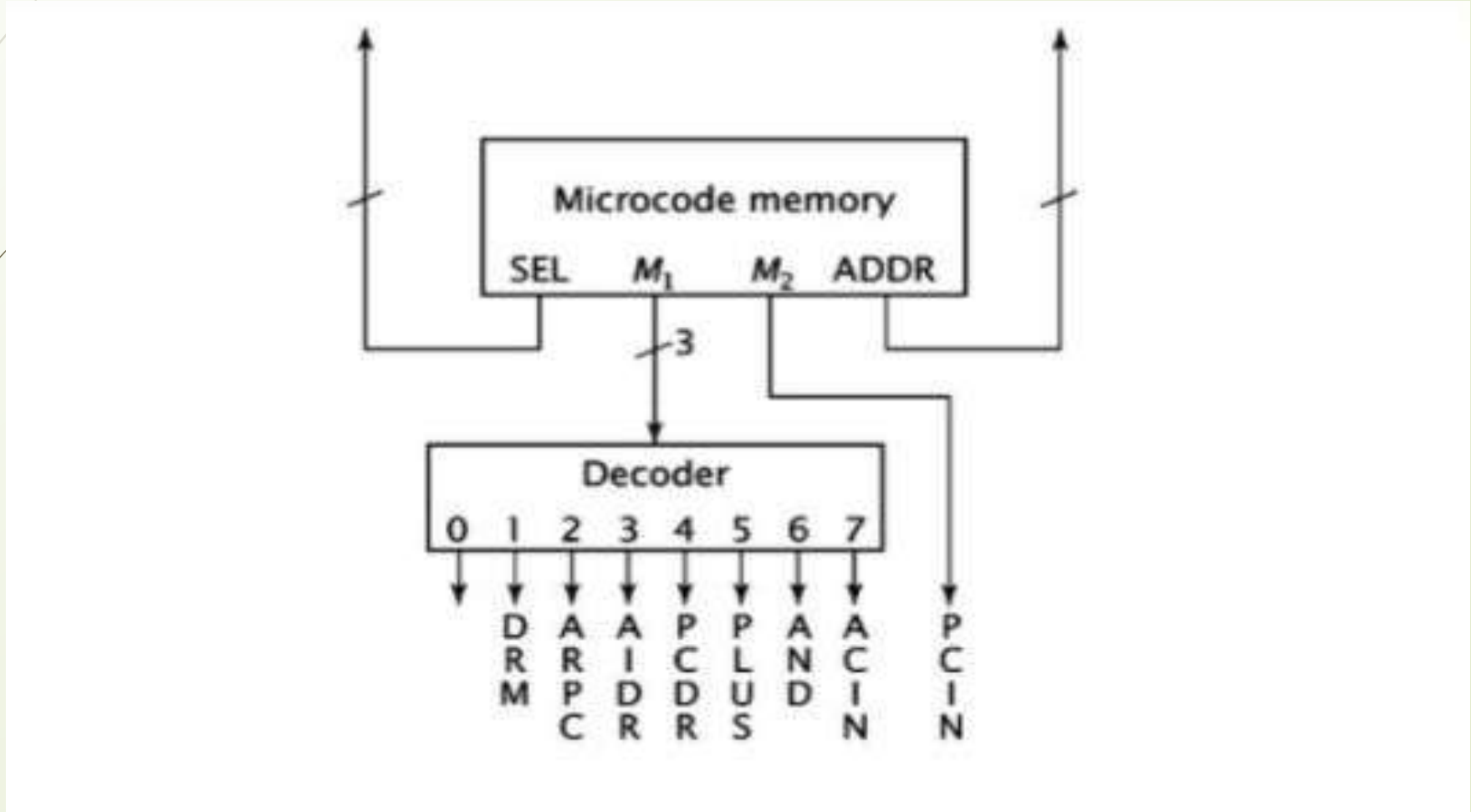
(vertical microcode for the very simple microsequencer)

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

Contd..

(generating micro operations for vertical microcode for the very simple CPU)



Control signal values for the very simple CPU

Control signal values for the Very Simple CPU

Signal	Value
ARLOAD	ARPC ∨ AIDR
PCLOAD	PCDR
PCINC	PCIN
DRLOAD	DRM
ACLOAD	PLUS ∨ AND
ACINC	ACIN
IRLOAD	AIDR
ALUSEL	AND
MEMBUS	DRM
PCBUS	ARPC
DRBUS	AIDR ∨ PCDR ∨ PLUS ∨ AND
READ	DRM

Directly generating control signals from the microcode

(Instead of outputting micro operation from the microcode memory and generating control signals from these micro operations, we could output the control signals directly)

- Use one bit for each control signal
  - Set to 1, it is active
  - Set to 0, it is not active
- Example

  FETCH2: DR ← M and PC ← PC + 1

  READ output data from memory

  MEMBUS to allow data onto internal bus

  DRLOAD to load data from bus into DR

  PCINC to perform second micro operation

Set those four to 1, the others to 0

Contd...

Microcode to directly generate control signals for the Very Simple Microsequencer

State	Address	SEL	ARLOAD	PCLOAD	PCINC	DRLOAD	ACLOAD	ACINC	IRLOAD	ALUSEL	MEMBUS	PCBUS	DRBUS	READ	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	0	1	0	0	0001
FETCH2	0001 (1)	0	0	0	1	1	0	0	0	0	1	0	0	1	0010
FETCH3	0010 (2)	1	1	0	0	0	0	0	1	0	0	0	1	0	XXXX
ADD1	1000 (8)	0	0	0	0	1	0	0	0	0	1	0	0	1	1001
ADD2	1001 (9)	0	0	0	0	0	1	0	0	0	0	0	1	0	0000
AND1	1010 (10)	0	0	0	0	1	0	0	0	0	1	0	0	1	1011
AND2	1011 (11)	0	0	0	0	0	1	0	0	1	0	0	1	0	0000
JMP1	1100 (12)	0	0	1	0	0	0	0	0	0	0	0	1	0	0000
INC1	1110 (14)	0	0	0	0	0	0	1	0	0	0	0	0	0	0000

Contd...

Optimized microcode to directly generate control signals

State	Address	SEL	ARLOAD	PCLOAD	PCINC	DMR	ACLOAD	ACINC	IRLOAD	ALUSEL	PCBUS	DRBUS	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	1	0	0001
FETCH2	0001 (1)	0	0	0	1	1	0	0	0	0	0	0	0010
FETCH3	0010 (2)	1	1	0	0	0	0	0	1	0	0	1	XXXX
ADD1	1000 (8)	0	0	0	0	1	0	0	0	0	0	0	1001
ADD2	1001 (9)	0	0	0	0	0	1	0	0	0	0	1	0000
AND1	1010 (10)	0	0	0	0	1	0	0	0	0	0	0	1011
AND2	1011 (11)	0	0	0	0	0	1	0	0	1	0	1	0000
JMP1	1100 (12)	0	0	1	0	0	0	0	0	0	0	1	0000
INC1	1110 (14)	0	0	0	0	0	0	1	0	0	0	0	0000

Advantage to directly generating control signals

- Does not require additional logic to convert the outputs of the microcode memory to control signals
- However, less readable and more difficult to debug

Microprogrammed Vs Hardwired control unit

HARDWIRED CONTROL UNIT	MICROPROGRAMMED CONTROL UNIT
The control unit whose control signals are generated by the hardware through a sequence of instructions is called a hardwired control unit.	The control unit whose control signals are generated by the data stored in control memory and constitute a program on the small scale is called a microprogrammed control unit
The control logic of a hardwired control is implemented with gates, flip flops, decoders etc.	The control logic of a micro-programmed control is the instructions that are stored in control memory to initiate the required sequence of microoperations.
Wiring changes are made in the hardwired control unit if there are any changes required in the design.	Changes in a microprogrammed control unit are done by updating the microprogram in control memory.
Hardwired control unit are faster and known to have complex structure.	Microprogrammed control unit is comparatively slow compared but are simple in structure.

As the number of instruction increases, the complexity of hardwire to generate control signals also increases. Therefore, for fewer number of instruction set, hardwire control unit is better, but for higher number of instruction set, microprogram may be suitable.