Chapter 5

CONTROL UNIT DESIGN

Assistant Professor

Er. Shiva Ram Dam

Contents:

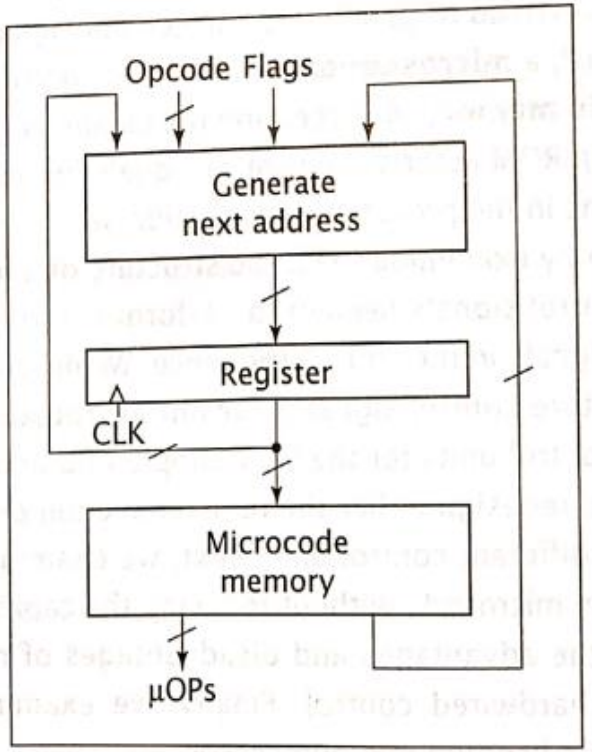
1. Basic Micro-sequencer (Control Unit) Design and Operations
2. Micro-instruction Formats
3. Design and Implementation of a Very Simple Micro-sequencer
4. Control Unit: Layout, Control Sequence Generation, Mapping Logic
5. Generation of Micro-operations using Horizontal and Vertical Microcode
6. Directly generating the control signal from the Microcode
7. Reducing tile Number of Micro-instructions
8. Micro-programmed Vs Hardwired Control Unit

5.0 Introduction to Micro sequencer

- Hardwired control is one of the two most commonly used control methods.
- The another popular control method is the micro-sequencing.
- Instead of generating the control signals using combinational and sequential logic, a microsequencer stores its control signals in a lookup ROM, the microcode memory.
- By accessing the locations of the ROM in the correct order, the lookup ROM asserts the control signals in the proper sequence to realize the instructions in the processor's instruction set.

5.1 Basic Microsequencer (CU) Design and Operations

- Like in hardwired control unit, a microsequencer is also designed as a finite state machine.
- Figure aside shows a generic microsequencer.



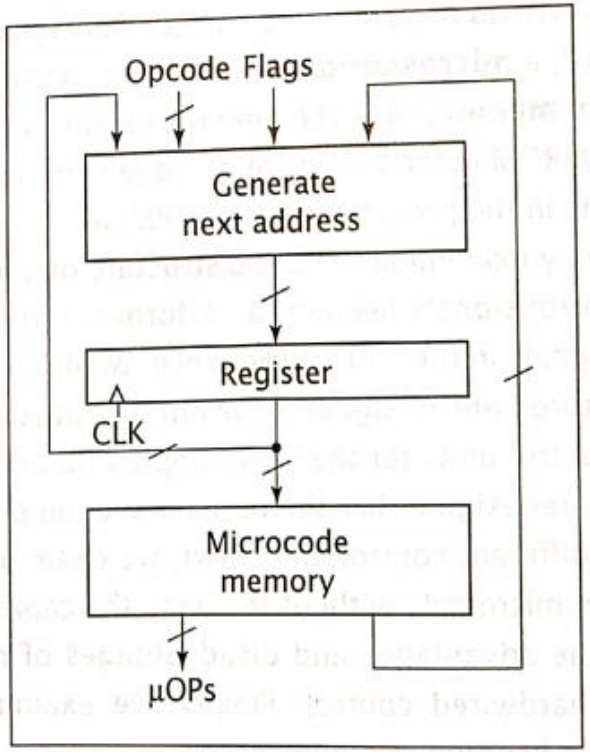
Generic microsequencer Organization

5.1 Basic Microsequencer (CU) Design and Operations

1. **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.

2. **MICRO-CODE** memory outputs a micro-instruction (the content of memory location for this address). Collectively, all of the microinstructions comprise the microcode, or microprogram, for the CPU.

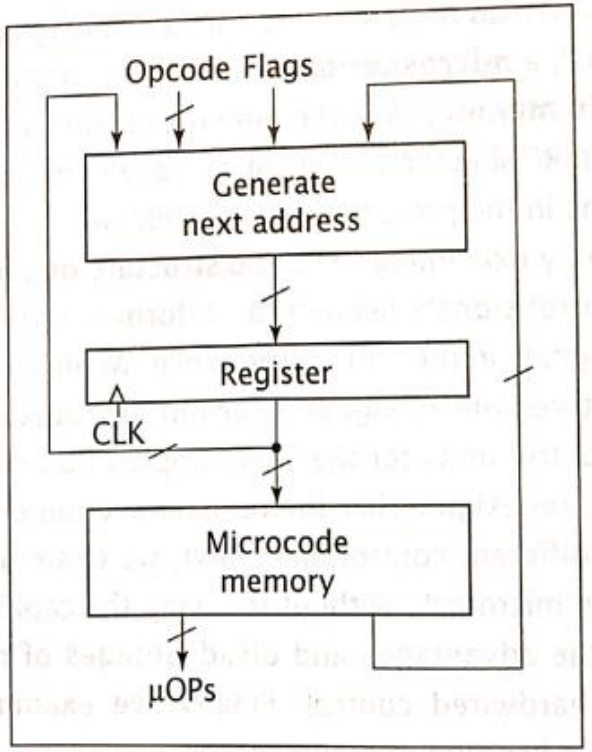
- **MICRO-INSTRUCTION** consists of several bit of fields which can be broadly classified into two groups:
  - **a)** **micro-operation** that generates the control signal.
  - **b)** **Sequencer** is used to generate the next address to be stored in the register.



Generic microsequencer Organization

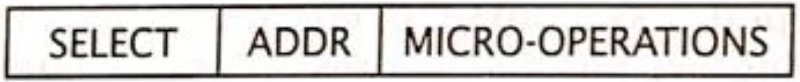
5.1 Basic Microsequencer (CU) Design and Operations

3. **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 memory.



Generic microsequencer Organization

2.5 Micro-instruction formats



- **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

a) Horizontal Microcode

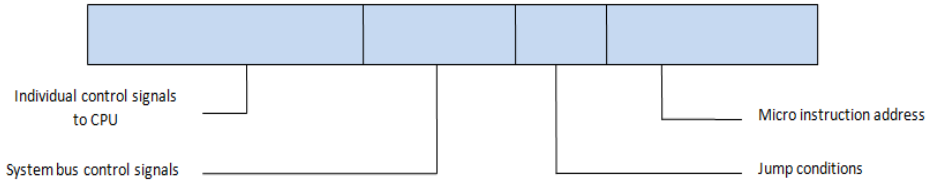


Figure: Horizontal microinstruction format

- In horizontal micro-instruction, each control signal is represented by a single bit in the control word. Thus, if the design has 500 control signals, it requires 500 bits in each control word to store the control bits. In this format, the control store looks horizontal in shape, since the control words are wide.
- The format of the horizontal instruction is as:
  - One bit for each internal processor control line
  - One bit for each system bus control line
  - Condition bit for flags
  - Address field
- **Advantage:** Fast operation as the control signals will be ready as soon as the control word is fetched form the control store.
- **Disadvantages:** The control word occupies more space of the control store.

a) Horizontal Microcode

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

b) Vertical Microcode

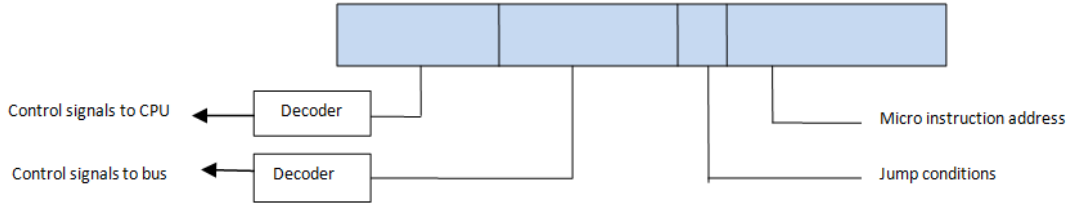


Figure: Vertical microinstruction format

- A vertical micro-instruction format is quite similar to the horizontal micro-instruction. It also consists of four fields:
  - Microinstruction address field
  - Condition field
  - Control signals to CPU
  - Control signals to system bus
- Here, the difference is that: instead of a single 1-bit for each element, a code is used for each action to be performed. However, a decoder is used for translation of these codes into individual control signals.
- These are more compact than the horizontal microinstruction. Hence, called vertical since the width is reduced.
- For eg: for each 16 ALU operations, 16 individual bits are required in horizontal micro-instruction. But only 4 encoded bits are used in vertical micro-operation.
- **Advantage:** shorter width of control word
- **Disadvantage:** slower implementation

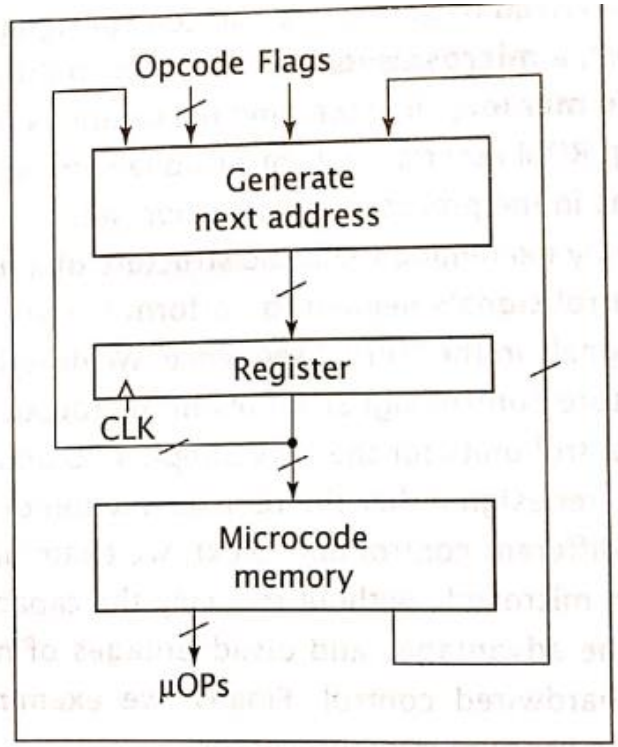
b) Vertical Microcode

- The number of bits in micro-operation filed can be reduced by using vertical micro-code.
- In vertical microcode, the micro-operations are grouped into fields.
- Each microoperation is assigned a unique encoded value in this field.
- For example:
- 16 microoperations can be encoded using 4 bits, from 0000 to 1111.
- Vertical microinstruction require fewer bits than their equivalent horizontal micro-instructions. However, micro sequencers that use vertical microcode must include a decoder for each micro-operation field to generate actual micro-operation signals.

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

5.3 Design and Implementation of a Very Simple Micro-sequencer

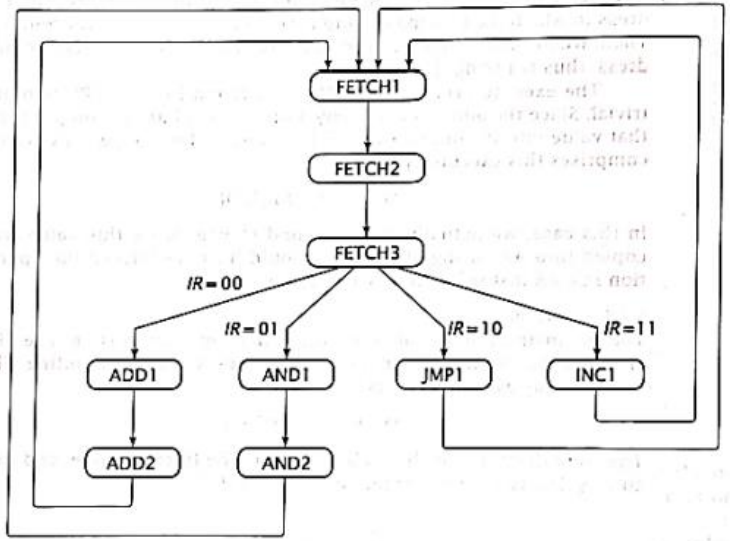
- To illustrate the process of microsequencer design, let us consider the very simple CPU that was developed in the previous chapter.
- We redesign this CPU ,this time using a microsequencer instead of hardwired control.
- The instruction set, finite state machine, data paths and SLU are the same.



Generic microsequencer organization

5.3 Design and Implementation of a Very Simple Micro-sequencer

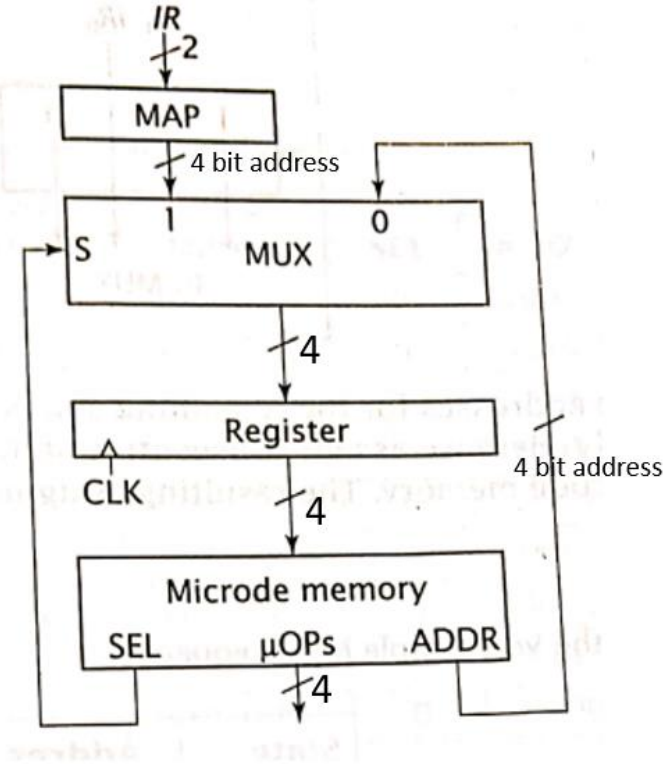
- Here, only two possible next address are used:
  - i) the ***opcode mapping*** and
  - ii) ***Absolute jump.***
- The last state of fetch cycle, fetch 3 goes to one of the 4 execute routines (ADD, AND , JMP, INC). This is implemented by ***mapping input.***
- The remaining states , each must go to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



Complete state diagram for the Very Simple CPU

5.3 Design and Implementation of a Very Simple Micro-sequencer

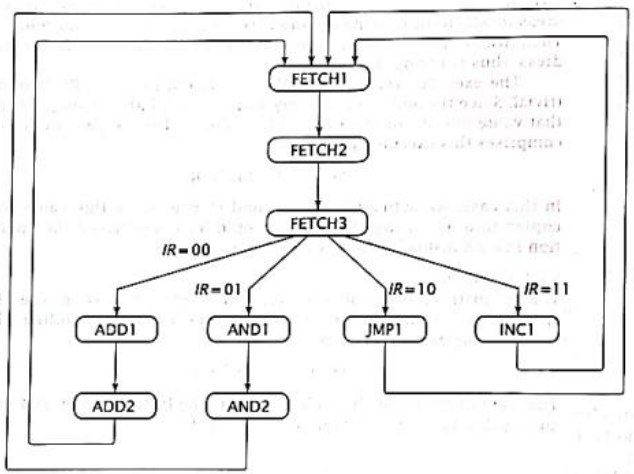
- Since, there are two possible addresses, the micro-sequencer must select one of them. So, there is a **multiplexer** to do this.
- The **select bit** is generated by microsequencer to choose the correct next address.
- The minimum number of bits needed to select is 4, which is the size of absolute address (because for our CPU, a total of 9 states are there). The mapping hardware also generates an address of 4 bits wide. The output of multiplexer is also 4 bits that is input to register and output of register that is input to address inputs of microcode memory.



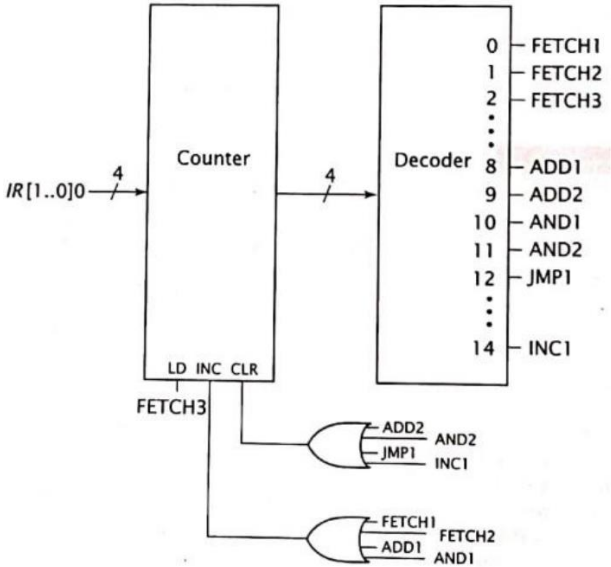
5.4 Control unit :

Layout, Control Sequence Generation, Mapping Logic

- Consider below state diagram and Hardwired Control Unit for a relatively small CPU (from previous chapter).



Complete state diagram for the Very Simple CPU

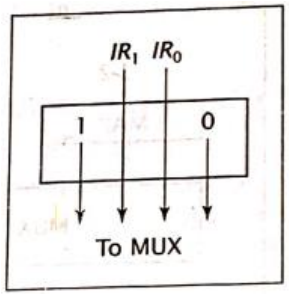


Hardwired control unit for the Very Simple CPU

FETCH1:	$AR \leftarrow PC$
FETCH2:	$DR \leftarrow M$, $PC \leftarrow PC + 1$
FETCH3:	$IR \leftarrow DR[7..6]$, $AR \leftarrow DR[5..0]$
ADD1:	$DR \leftarrow M$
ADD2:	$AC \leftarrow AC + DR$
AND1:	$DR \leftarrow M$
AND2:	$AC \leftarrow AC \wedge DR$
JMP1:	$PC \leftarrow DR[5..0]$
INC1:	$AC \leftarrow AC + 1$

RTL code

- For this CPU, the micro-sequencer use the same mapping function that was used for hardwired control unit. 1 IR[1..0]0
- This produces address of 1**00**0, 1**01**0, 1**10**0 and 1**11**0 (8, 10, 12 and 14) for ADD1, AND 1 JMP and INC1.
- The address for remaining states can be assigned somewhat randomly. Here we assign consecutive states to consecutive location in microcode memory (control memory)



Mapping logic for the Very Simple Microsequencer

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)

State addresses for the Very Simple Microsequencer

- To go from Fetch1 to Fetch2, SEL=0 and ADDR=0001
- Setting SEL=0 causes micro-sequencer to get its next address from ADDR field; and setting ADDR field to 0001 causes it to go to location corresponding to state Fetch2.
- 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 does not matter what value it has.

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

Partial Microcode for the very simple microprocessor

5.5 Generation of Micro-operations

Generating the micro-operations using Horizontal Microcode

- Microsequencer has two tasks:
  - i) to generate the correct micro-operations, and
  - ii) to follow the correct sequence of states.
- One task is complete. Now, the other task is to generate correct micro-operations, and associated control signals. This can be done by horizontal microcode, vertical microcode and Direct generation of control signals.
- In horizontal microcode, each microinstruction is represented by one-bit in each micro-instruction. The greater the number of microoperation, the larger the microcode will be.
- The microoperations and their mnemonics are:

Mnemonic	Micro-Operation
ARPC	$AR \leftarrow PC$
ARDR	$AR \leftarrow DR[5..0]$
PCIN	$PC \leftarrow PC + 1$
PCDR	$PC \leftarrow DR[5..0]$
DRM	$DR \leftarrow M$
IRDR	$IR \leftarrow DR[7..6]$
PLUS	$AC \leftarrow AC + DR$
AND	$AC \leftarrow AC \wedge DR$
ACIN	$AC \leftarrow AC + 1$

Micro-operations and their mnemonics for the Very Simple CPU

- Since there are 9 micro-operations, each word of microcode requires 9 bits to represent them, 1 bit per micro-operation.
- A value of 1 means the micro-operation is to occur and a value of 0 means that it does not.
- To complete the microcode, we simply fill in the values for micro-operations.
- The resultant microcode is shown in table aside (below).

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

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←AC ^ DR
ACIN	AC←AC + 1

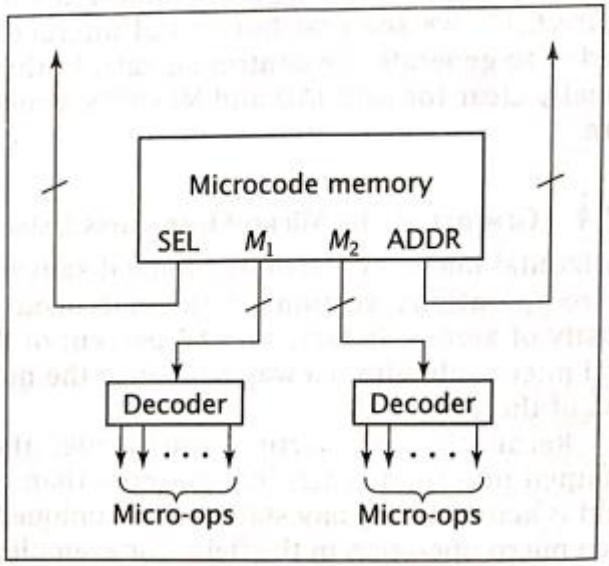
FETCH1: AR←PC
FETCH2: DR←M, PC←PC + 1
FETCH3: IR←DR[7..6], AR←DR[5..0]
ADD1: DR←M
ADD2: AC←AC + DR
AND1: DR←M
AND2: AC←AC ^ DR
JMP1: PC←DR[5..0]
INC1: AC←AC + 1

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

Preliminary horizontal microcode for the Very Simple Microsequencer

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 value is assigned to each microoperation in the field.
- For eg, a field with 8 (i.e. 2^3=8) different microoperations would require 3 bits each value from 000 to 111 would be assigned to one of the eight microoperations. The microoperation field bits are output from microcode memory to a decoder.



Generic generation of micro-operations from vertical microcode

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

For eg, DRM and PCIN both occur during Fetch2 states. So, must be assigned to different fields, one in M1 field and other in M2

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

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

The CPU requires at least two fields for its microoperation labeled as M1 and M2.

Mnemonic	Micro-Operation
ARPC	$AR \leftarrow PC$
ARDR	$AR \leftarrow DR[5..0]$
PCIN	$PC \leftarrow PC + 1$
PCDR	$PC \leftarrow DR[5..0]$
DRM	$DR \leftarrow M$
IRDR	$IR \leftarrow DR[7..6]$
PLUS	$AC \leftarrow AC + DR$
AND	$AC \leftarrow AC \wedge DR$
ACIN	$AC \leftarrow AC + 1$

FETCH1: $AR \leftarrow PC$
FETCH2: $DR \leftarrow M$, $PC \leftarrow PC + 1$
FETCH3: $IR \leftarrow DR[7..6]$, $AR \leftarrow DR[5..0]$
ADD1: $DR \leftarrow M$
ADD2: $AC \leftarrow AC + DR$
AND1: $DR \leftarrow M$
AND2: $AC \leftarrow AC \wedge DR$
JMP1: $PC \leftarrow DR[5..0]$
INC1: $AC \leftarrow AC + 1$

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

Micro-operation field assignments and values

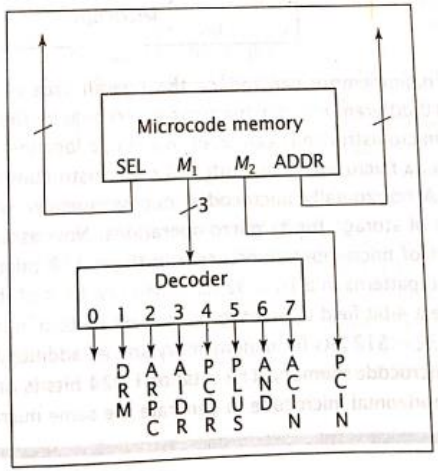
FETCH1: $AR \leftarrow PC$
FETCH2: $DR \leftarrow M,\ PC \leftarrow PC + 1$
FETCH3: $IR \leftarrow DR[7..6],\ AR \leftarrow DR[5..0]$
ADD1: $DR \leftarrow M$
ADD2: $AC \leftarrow AC + DR$
AND1: $DR \leftarrow M$
AND2: $AC \leftarrow AC \wedge DR$
JMP1: $PC \leftarrow DR[5..0]$
INC1: $AC \leftarrow AC + 1$

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

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

Vertical Microcode for the Very Simple Micro-sequencer



Generating micro-operations from vertical microcode for the very simple CPU

5.6 Directly generating the control signals from the Microcode

- Instead of outputting micro-operations form the microcode memory and then generating the control signals from these micro-operations, we can output the control signals directly.
- As before, the sequencing portion of the control unit design remains the same.
- This portion includes the mapping logic, next address multiplexer, register, and microcode memory as well as microcode fields SEL and ADDR.
- In place of the micro-operations(in the horizontal microcode design) or M1 and M2 (in the vertical microcode design), the micro-sequencer can include 1 bit for each control signal.
- Foe each word of microcode memory, each control bitis set to 1 if the signal is to be active , and 0 otherwise.

- To illustrate this, consider the micro-operations that occur during FETCH2, DR← M and PC← PC +1
- To implement DR← M , we must enable READ to output data from memory, MEMBUS to allow this data onto the internal system bus, and DRLOAD to load the data from the bus into DR.
- Also, we must enable PCINC to perform the second microoperation.
- All the other signals must be set to zero for this microinstruction so that no other micro-operation is triggered.
- Doing this for every microinstruction produces the microcode shown in Table in next slide.

- In table above, it is to be noted that DRLOAD, MEMBUS and READ always have the same value.

Table 7.9
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
[illegible]	1110 (14)	0	0	0	0	0	0	1	0	0	0	0	0	0	0000

- Directly generating the control signals offers advantage over the other two methods.
- It does not require additional logic to convert the outputs of the microcode memory to control signals.
- However, this type of code is less readable and thus more difficult to debug.

5.7 Reducing the number of micro-instructions

- Just as with the micro-operations ARDR and IRDR in the horizontal microcode design, we can combine these signals because they always have the same value.
- One bit can be used instead of three; that one bit can drive all three signals.
- The revised microcode with the one bit labeled DMR is shown in Table 7.10

Table 7.10
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

5.8 Micro-programmed Vs Hardwired Control Unit

	Hardwired		Micro-programmed
1.	Uses flags, decoder, logic gates and other digital circuits	1.	Uses sequence of micro-instruction in micro-programming language
2.	As name implies, it is a hardware control unit.	2.	It is midway between Hardware and software.
3.	On the basis of input signal, output is generated.	3.	It generates a set of control signal on the basis of control line.
4.	Difficult to design, test and implement.	4.	Easy to design and implement
5.	Inflexible to modify	5.	Flexible to modify
6.	Faster mode of operation	6.	Slower mode of operation
7.	Expensive and high error	7.	Cheaper and less error
8.	Used in RISC processor	8.	Used in CISC processor

Exam questions:

7. There is a Very Simple CPU for the given set of Instructions: [PU 2017 Fall]
   - LDA (Load Accumulator with a value from memory) with opcode OO; AND) performing and operation)
   - AND (performing and operation to the content of accumulator with the value at some memory address) with opcode 01;
   - SKIP (skipping an instruction) with opcode 10;
   - DEC (decrementing the value at accumulator) with opcode 11;

   Let the instruction width be 8 bits and address is 6 bits. Design the CPU's Register section, state diagram and ALU.

1. **Design a microsequencer control unit with horizontal microcode for the CPU described in question 7 (i.e. above question). [PU 2017 Fall]**

7. There is a very simple CPU for the given set of Instruction: [PU 2017 Spring]

Instruction	Instruction Code	Micro-operation
ADD	00 AAAAAA	AC←AC+M[AAAAAA]
SUB	01 AAAAAA	AC←AC-M[AAAAAA]
SKIP	10 XXXXXX	PC←PC+1
DEC2	11 XXXXXX	AC←AC-2

Let the instruction width be 8 bits and address is 6 bits. Design the CPU's register section, state diagram and ALU.

2. Design a microsequencer control unit which directly generate control signals for the CPU described in question 7 (Above question) . [PU 2017 Spring]

7. There is a very simple CPU for the given set of Instruction: [PU 2018 Fall]

Instruction	Instruction Code	Operations
STA	00 AAAAAA	M[AAAAAA}←AC
XNOR	01 AAAAAA	AC←ACOM[AAAAAA]
JMP	10 AAAAAA	GOTO AAAAAA
SKIP	11 XXXXXX	PC←PC+1

Let the instruction width be 8 bits and address is 6 bits. Design the CPU's register section, state diagram and ALU.

3. Design a microsequencer control unit with horizontal microcode for the CPU described in question 7 (above question) [PU 2018 Fall]

7. **There is a very simple CPU for the given set of Instruction: [PU 2018 Spring]**

Instruction	Instruction Code	Operations
STR	00 AAAAAA	AC→M[AAAAAA]
NAND	01 AAAAAA	AC←(AC^M[AAAAAA])`
JMP	10 AAAAAA	PC← AAAAAA
INC2	11 XXXXXX	AC←AC+2

Let the instruction width be 8 bits and address is 6 bits. Design the CPU's register section, state diagram and ALU.

4. **Design a microsequencer control unit with horizontal microcode for the CPU described in question 7. [PU 2018 Spring]**

7. **There is a very simple CPU for the given set of Instruction: [PU 2019 Spring]**

Instruction	Instruction Code	Operations
STR	00 AAAAAA	AC→M[AAAAAA]
AND	01 AAAAAA	AC←AC^M[AAAAAA]
JMPR	10 AAAAAA	PC←PC+AAAAAA
DEC2	11 XXXXXX	AC←AC-2

Let the instruction width be 8 bits and address is 6 bits. Design the CPU's register section, state diagram and ALU.

5. Design a microsequencer control unit with horizontal microcode for the CPU described in question 7. [PU 2019 Spring]

End of chapter 5