BSC CSIT 3rd SEM
CHAPTER 4
MICROPROGRAMMED CONTROL
LH-4

CONTENTS

4.1 Introduction: Hardwired and Microprogrammed Control Unit, Control Word, Microprogram, Control Memory, Control Address Register, Sequencer,

4.2 Address Sequencing: Conditional Branch, Mapping of Instructions, Subroutines, Microinstruction Format, Symbolic Microinstructions

4.3 Design of Control Unit: Decoding, Microprogram Sequencer.

4.1 Introduction

- The function of the control unit in a digital computer is to initiate sequences of microoperations. The number of different types of microoperations that are available in a given system is finite. The complexity of the digital system is derived from the number of sequences of microoperations that are performed. Two techniques used for implementing control unit are hardwired and microprogrammed.
- **Hardwired Control:**

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired. (Refer chapter 3, timing and control unit for diagram)

- **Microprogrammed Control:**

Microprogramming is a second alternative for designing the control unit of a digital computer which uses microoperations sequences.

A computer that employs a microprogrammed control unit will have two separate memories: a main memory and a control memory.

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.

Control Memory

- **Control Memory (Control Storage: CS):** Storage in the microprogrammed control unit to store the microprogram.
- **Control word:** It is a string of control variables (0's and 1's) occupying a word in control memory.
- **Microprogram:**

➢ Program stored in control memory that generates all the control signals required to execute the instruction set correctly

➢ Consists of microinstructions

- **Microinstruction:** microins code format 20 bits
  - Contains a control word and a sequencing word CW 9 bits SW 11 bits
  - Control Word – contains all the control information required for one clock cycle
  - Sequencing Word - Contains information needed to decide the next microinstruction address
- **Microoperation:**
  - A microinstruction contains one or more microoperations to be completed.
- **Writable Control Memory (Writable Control Storage: WCS):**

CS whose contents can be modified:

- Microprogram can be changed
- Instruction set can be changed or modified

A computer that employs a microprogrammed control unit will have two separate memories: main memory and a control memory. The user's program in main memory consists of machine instructions and data whereas control memory holds a fixed microprogram that cannot be altered by the user. Each machine instruction initiates a series of microinstructions in control memory.

The general configuration of a microprogrammed control unit is demonstrated in the following block diagram:

Block Diagram of Microprogrammed control unit

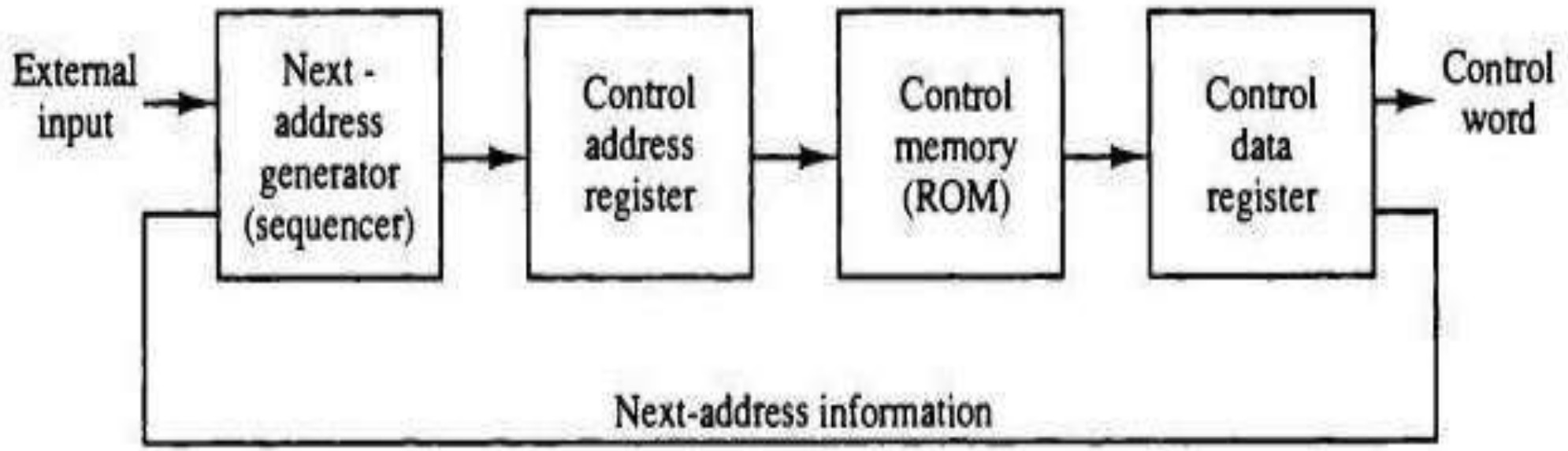


Fig: Microprogrammed control organization

Dynamic Microprogramming

- A more advanced development that permits a microprogram to be loaded initially from an auxiliary memory such as magnetic disk.
- Computer system whose control unit is implemented with a microprogram in WCS.
- Microprogram can be changed by a systems programmer or a user.

- **Sequencer:** The device or program that generates address of next microinstruction to be executed is called sequencer. While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

- **Control Address Register:** CAR contains address of microinstruction.
- **Control Data Register:** CDR contains microinstruction read from memory. The microinstruction contains a control word that specifies one or more microoperations. The data register is sometimes called a *pipeline register.*

It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction. This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.

4.2 Address Sequencing

Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction. Process of finding address of next microinstruction to be executed is called ***address sequencing***. The address sequencing capabilities required in a control memory are:

a) Incrementing of the control address register.
b) Unconditional branch or conditional branch, depending on status bit conditions.
c) A mapping process from the bits of the instruction to an address for control memory.
d) A facility for subroutine call and return.

Following is the block diagram for control memory and the associated hardware needed for selecting the next microinstruction address.

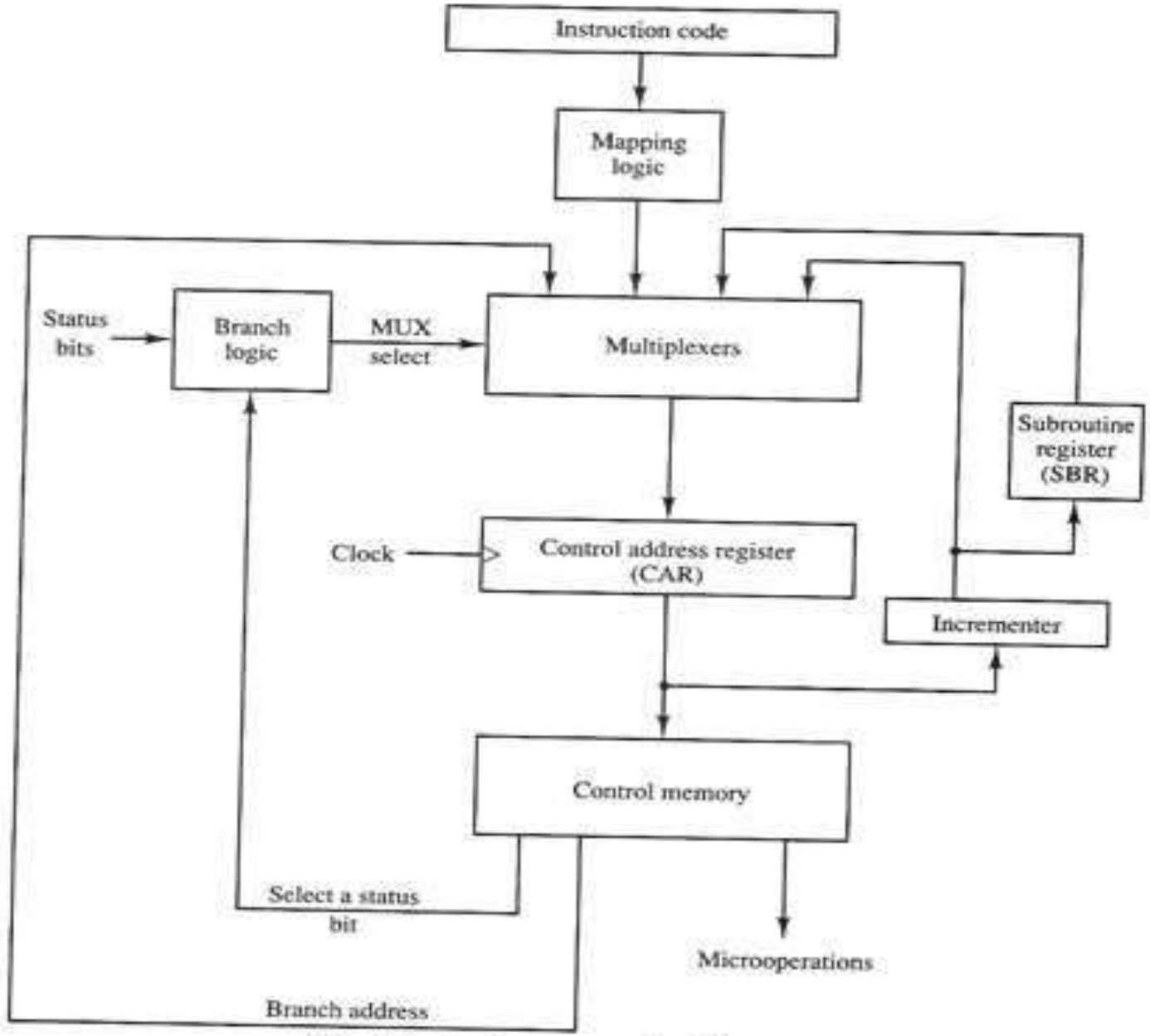


Fig: Block diagram of address sequencer.

The diagram shows four different paths from which the control address register (CAR) receives the address. The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence. Branching is achieved by specifying the branch address in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition. An external address is transferred into control memory via a mapping logic circuit. The return address for a subroutine is stored in a special register whose value is then used when the microprogram wishes to return from the subroutine.

Control address register receives address of next microinstruction from different sources.

a) Incrementer simply increments the address by one

b) In case of branching, branch address is specified in one of the field of microinstruction.

c) In case of subroutine call, return address is stored in the register SBR which is used when returning from called subroutine.

- **Conditional Branch**:

Simplest way of implementing branch logic hardware is to test the specified condition and branch to the indicated address if condition is met otherwise address register is simply incremented. If Condition is true, hardware set the appropriate field of status register to 1. Conditions are tested for O (overflow), N (negative), Z (zero), C (carry), etc.

- **Unconditional Branch:**

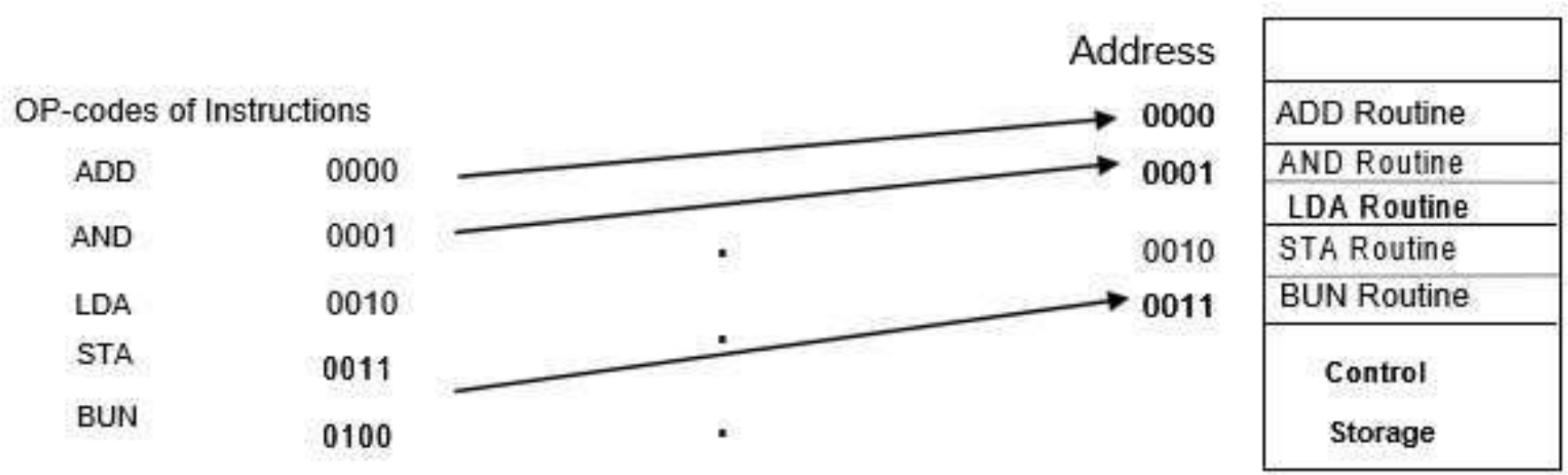
Fix the value of one status bit at the input of the multiplexer to 1. So that, branching can always be done.

- **Mapping**

Assuming operation code of 4-bits which can specify 16 (2^4) distinct instructions. Assume further and control memory has 128 words, requiring an address of 7-bits. Now we have to map 4-bit operation code into 7-bit control memory address. Thus, we have to map Op-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its subroutine in memory.

- **Direct mapping:**

  Directly use op-code as address of Control memory



- **Another approach of direct mapping:**

Transfer Op-code bits to use it as an address of control memory. In this mapping, one 0 is placed in the MSB and two 0s in the LSB as shown in figure:

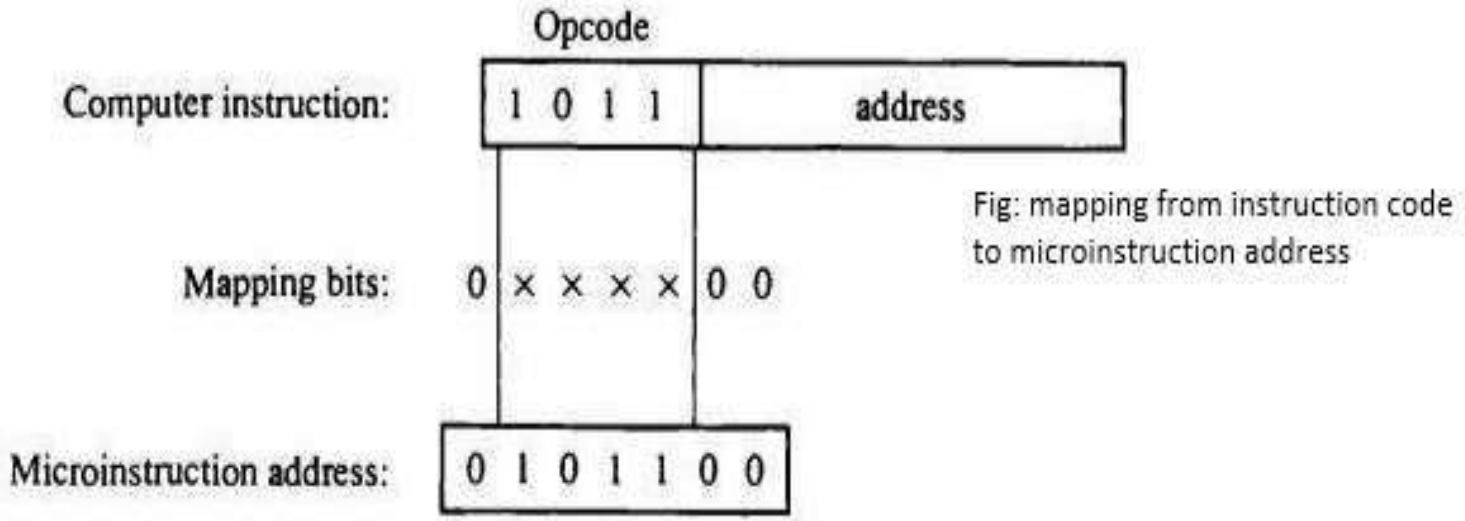
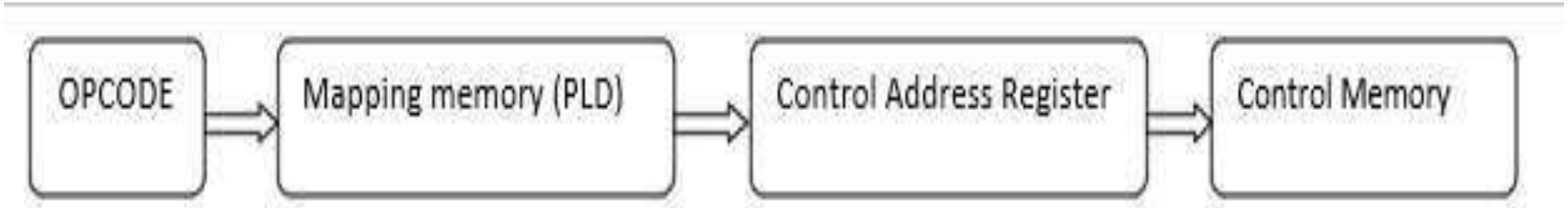


Fig: mapping from instruction code to microinstruction address

- **Extended idea: Mapping function implemented by ROM or PLD (Programmable Logic Device)**

Use op-code as address of ROM where address of control memory is stored and then use that address as an address of control memory. This provides flexibility to add instructions for control memory as the need arises.



- **Subroutines**:

Subroutines are programs that are used by another program to accomplish a particular task. Microinstructions can be saved by employing subroutines that use common sections of micro code. Example: the sequence of microoperations needed to generate the effective address is common to all memory reference instructions. Thus, this sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Subroutine register is used to save a return address during a subroutine call which is organized in LIFO (last in, first out) stack.

Microprogram Example

Once the configuration of a computer and its microprogrammed control unit is established, the designer's task is to generate the microcode for the control memory. This code generation is called microprogramming and is a process similar to conventional machine language programming.

Computer Configuration:

- It consists of two memory units: a main memory for storing instructions and data, and a control memory for storing the microprogram
- Four registers are associated with the processor unit and two with the control unit. The processor registers are PC, AR, DR and AC.
- The control unit has control address register CAR and subroutine register SBR.
- The transfer of information among the registers in processor is done through multiplexer rather than a common bus. DR can receive information from AC, PC or memory. AR can receive information from PC or DR. PC can receive information only from AR.
- The arithmetic, logic and shift unit performs microoperations with data from AC and DR and places the result in AC. Note that memory receives its address from AR. Input data written to memory come from DR, and data read from memory can go only to DR.
- The computer instruction format has three fields: a 1-bit field for indirect addressing symbolized by I, a 4-bit operation code (op-code), and an 11-bit address field. The figure below lists four of the 16 possible memory reference instructions.

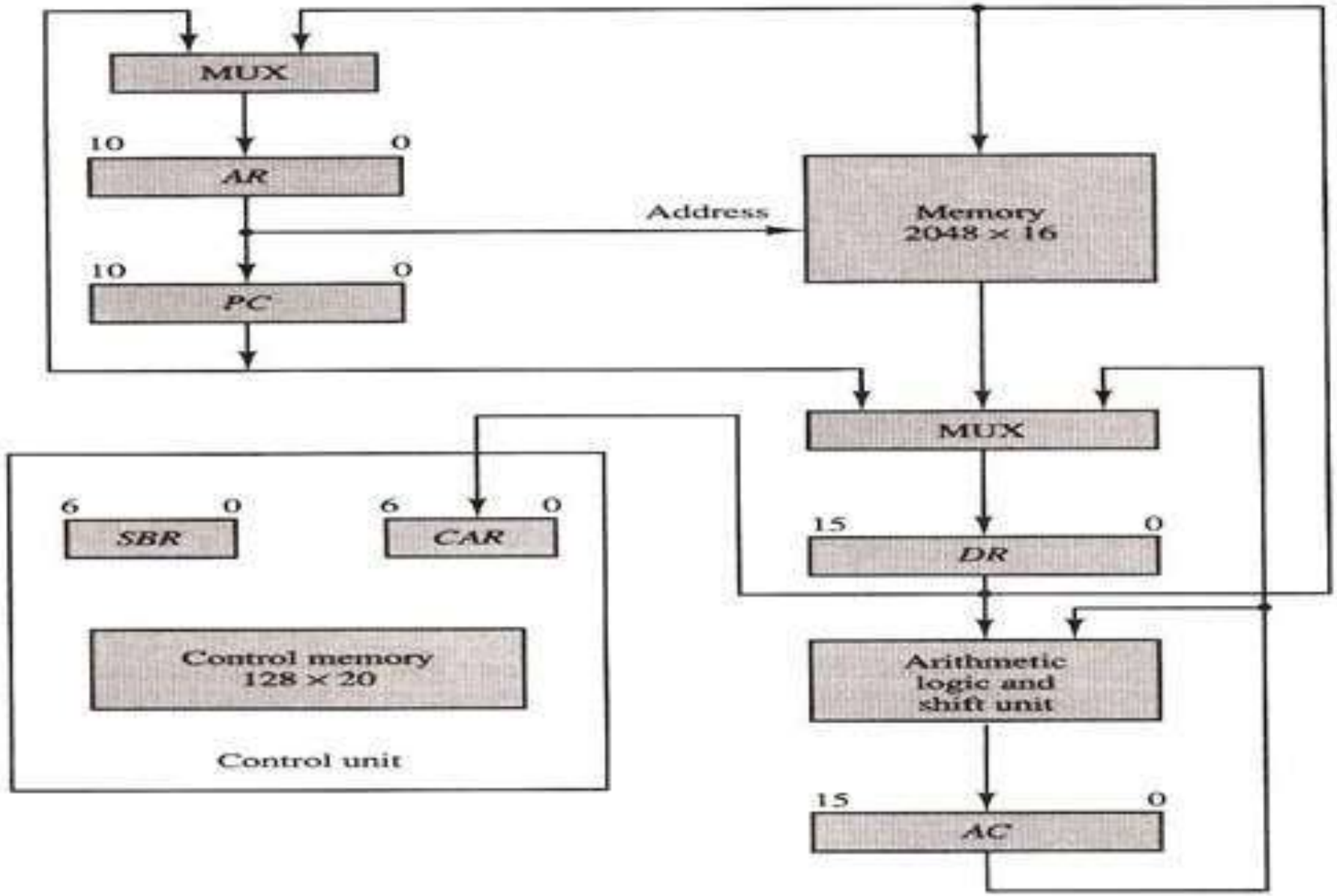
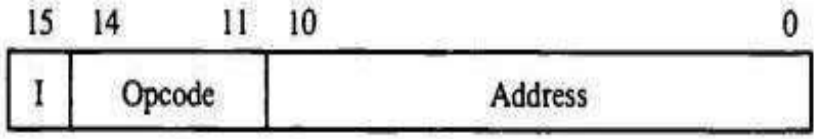


Fig: Computer hardware configuration

- The ADD instruction adds the content of the operand found in the effective address to the content of AC.
- The BRANCH instruction causes a branch to the effective address if the operand in AC is negative. The program proceeds with the next consecutive instruction if AC is not negative. The AC is negative if its sign bit is a 1.
- The STORE instruction transfers the content of AC into the memory word specified by the effective address.
- The EXCHANGE instruction swaps the data between AC and the memory word specified by the effective address.



(a) Instruction format

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions

Microinstruction Format and Description

We know the computer instruction format (explained in previous chapter) for different set of instruction in main memory. Similarly, microinstruction in control memory has 20-bit format divided into 4 functional parts as shown below.

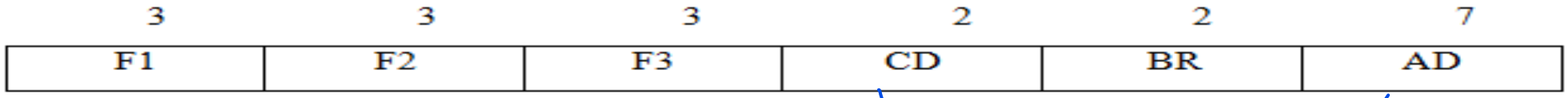


Fig: Microinstruction code format (20 bits)

F1, F2, F3: Microoperation fields

CD: Condition for branching indirect, sign, zero

BR: Branch field JMP, CALL,RET,MAP

AD: Address field

Each microoperation below is defined using resister transfer statements and is assigned a symbol for use in symbolic microprogram.

Description of CD

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	*DR*(15)	I	Indirect address bit
10	*AC*(15)	S	Sign bit of *AC*
11	*AC* = 0	Z	Zero value in *AC*

Description of BR

BR	Symbol	Function
00	JMP	*CAR* ← *AD* if condition = 1
		CAR ← *CAR* + 1 if condition = 0
01	CALL	*CAR* ← *AD*, *SBR* ← *CAR* + 1 if condition = 1
		CAR ← *CAR* + 1 if condition = 0
10	RET	*CAR* ← *SBR* (Return from subroutine)
11	MAP	*CAR*(2-5) ← *DR*(11-14), *CAR*(0,1,6) ← 0

CD (condition) field consists of two bits representing 4 status bits and BR (branch) field (2-bits) used together with address field AD, to choose the address of the next microinstruction.

Microinstruction fields (F1, F2, F3)

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

- Here, microoperations are subdivided into three fields of 3-bits each. These 3 bits are used to encode 7 different microoperations. No more than 3 microoperations can be chosen for a microinstruction, one for each field. If fewer than 3 microoperations are used, one or more fields will contain 000 for no operation.

Symbolic Microinstructions

Symbols are used in microinstructions as in assembly language. A symbolic microprogram can be translated into its binary equivalent by a microprogram assembler.

Format of Microinstruction:

Contains five fields: label; micro-ops; CD; BR; AD

Label: may be empty or may specify a symbolic address terminated with a colon

Micro-ops: consists of one, two, or three symbols separated by commas

CD: one of {U, I, S, Z},

Where

U: Unconditional Branch

I: Indirect address bit

S: Sign of AC

Z: Zero value in AC

BR: one of {JMP, CALL, RET, MAP}

AD: one of {Symbolic address, NEXT, empty (in case of MAP and RET)}

Symbolic Microprogram (example)

FETCH Routine: During FETCH Read an instruction from memory and decode the instruction and update PC

- Sequence of microoperations in the *fetch cycle*:

$$AR \leftarrow PC$$

$$DR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$

$$AR \leftarrow DR(0\text{–}10), \quad CAR(2\text{–}5) \leftarrow DR(11\text{–}14), \quad CAR(0,1,6) \leftarrow 0$$

The fetch routine needs three microinstructions, which are placed in control memory at address 64, 65 and 66.

Symbolic microprogram for the fetch cycle:

```
        ORG 64
FETCH:  PCTAR           U    JMP    NEXT
        READ, INCPC     U    JMP    NEXT
        DRTAR           U    MAP
```

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

Binary Microprogram

4.3 Design of control unit
F-field decoding

for reducing the complexity, we perform only 5 microoperation
PCTAR DRTAR DRTAC ADD AND

- The 9-bits of the microoperation field are divided into 3 subfields of 3 bits each. The control memory output of each subfield must be decoded to provide distinct microoperations. The outputs of the decoders are connected to the appropriate inputs in the processor unit. Fig below shows 3 decoders and connections that must be made from their outputs.

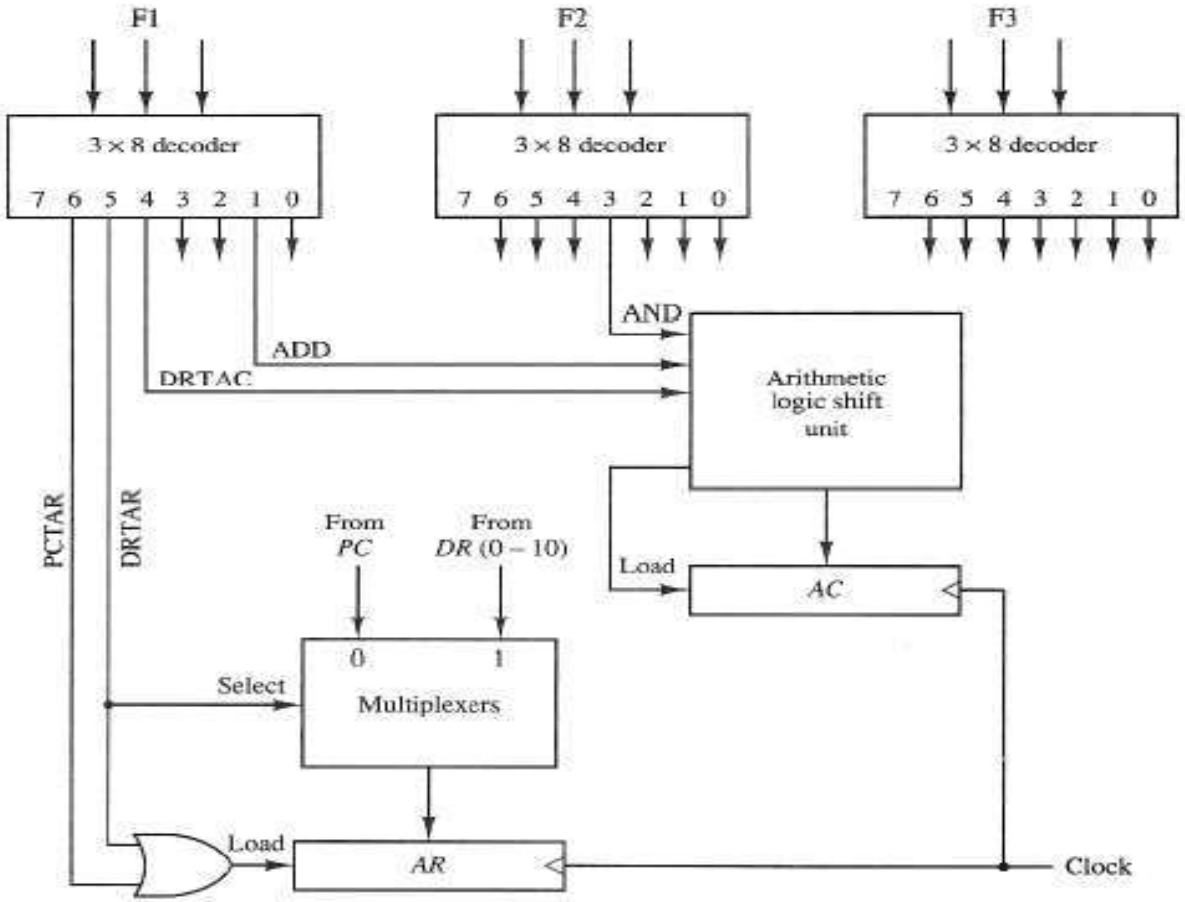


Fig: Decoding of microoperation fields

E.g. when F1=101 (binary 5), next clock pulse transition transfers the content of DR(0-10) to AR (DRTAR). Similarly when F1=110(6), there is a transfer from PC to AR (PCTAR). Outputs 5 & 6 of decoder F1 are connected to the load inputs of AR so that when either is active information from multiplexers is transferred to AR.

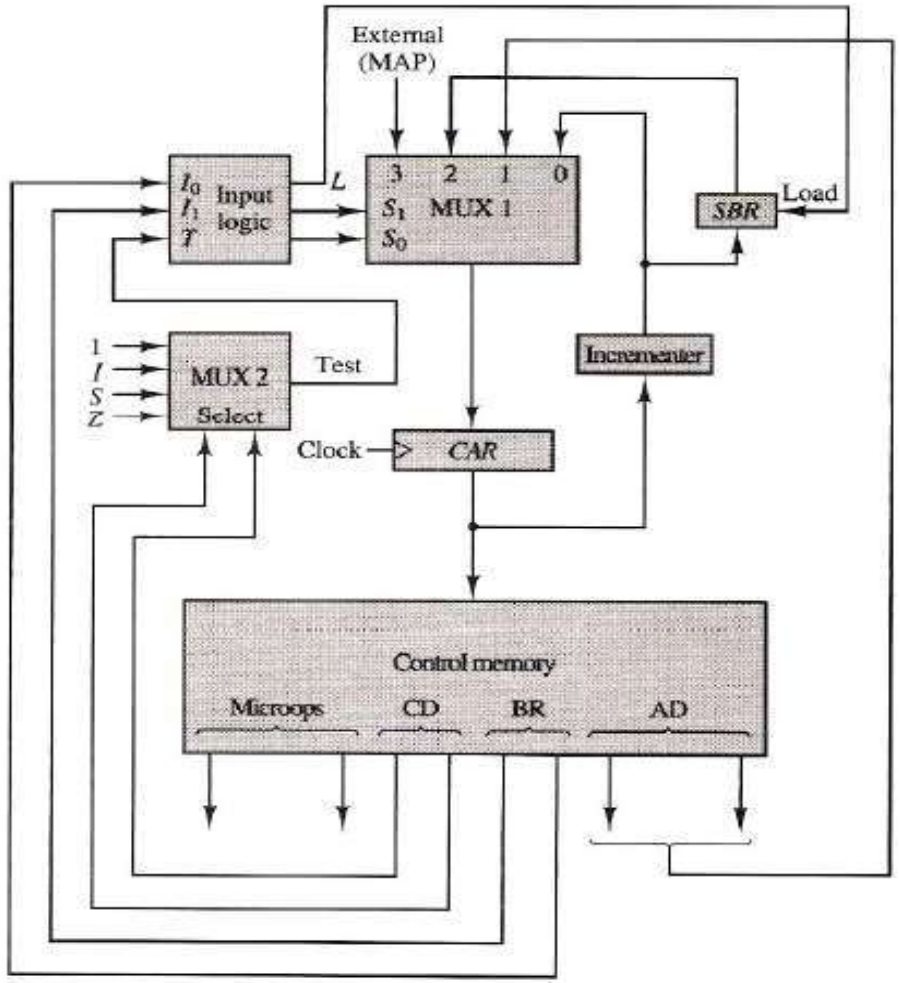
Arithmetic logic shift unit instead of using gates to generate control signals, is provided inputs with outputs of decoders (AND, ADD and ARTAC).

Microprogram Sequencer

- Basic components of a microprogrammed control unit are control memory and the circuits that select the next address. This address selection part is called a microprogram sequencer.
- The purpose of microprogram sequencer is to load CAR so that microinstruction may be read and executed. Commercial sequencers include within the unit an internal resister stack to store addresses during microprogram looping and subroutine calls.
- Internal structure of a typical microprogram sequencer is shown below in the diagram. It consists of input logic circuit having following truth table.

BR Field		Input			MUX 1		Load *SBR*
		I_1	I_0	T	S_1	S_0	L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	×	1	0	0
1	1	1	1	×	1	1	0

Fig: Input Logic Truth for Microprogram Sequencer



-MUX1 selects an address from one of four sources of and routes it into CAR.

-MUX2 tests the value of selected status bit and result is applied to input logic circuit.

-Output of CAR provides address for the control memory

-Input logic circuit has 3 inputs I_0, I_1 and T and 3 outputs S_0, S_1 and L. variables S_0 and S_1 select one of the source addresses for CAR. L enables load input of SBR.

-e.g. when S_1S_0=10, MUX input number 2 is selected and establishes a transfer path from SBR to CAR.

Fig: Microprogram sequencer for a control memory

- The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it.
- There are two multiplexers in the circuit.
- The first multiplexer selects an address from one of the four sources and routes it into the CAR(Control Address Register).
- The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.
- The output from CAR provides the address for the control memory.
- The contents of CAR is incremented and applied to one of the multiplexer inputs and to the SBR.
- The other three input come from the address field of the present microinstruction, from the output of SBR(Subroutine Register) and from an external source that maps the instruction.

- MUX-1 selects an address from one of four sources and routes it into a CAR

- In-Line Sequencing - CAR + 1

- Branch, Subroutine Call - Take address from AD field

- Return from Subroutine - Output of SBR

- New Machine instruction - MAP

- MUX-2 Controls the condition and branching

BR Field		Input I_1	I_0	T	MUX 1 S_1	S_0	Load SBR L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	×	1	0	0
1	1	1	1	×	1	1	0

Fig: Input Logic Truth for Micro program Sequencer

- Input logic circuit has 3 inputs I0, I1 and T and 3 outputs S0, S1 and L. variables S0 and S1 select one of the source addresses for CAR. L enables load input of SBR.
- E.g. when S1S0=10, MUX input number 2 is selected and establishes a transfer path from SBR to CAR.

memroy writes means register to memory