UNIT – 2b

Microprogrammed Control

Microprogrammed Control

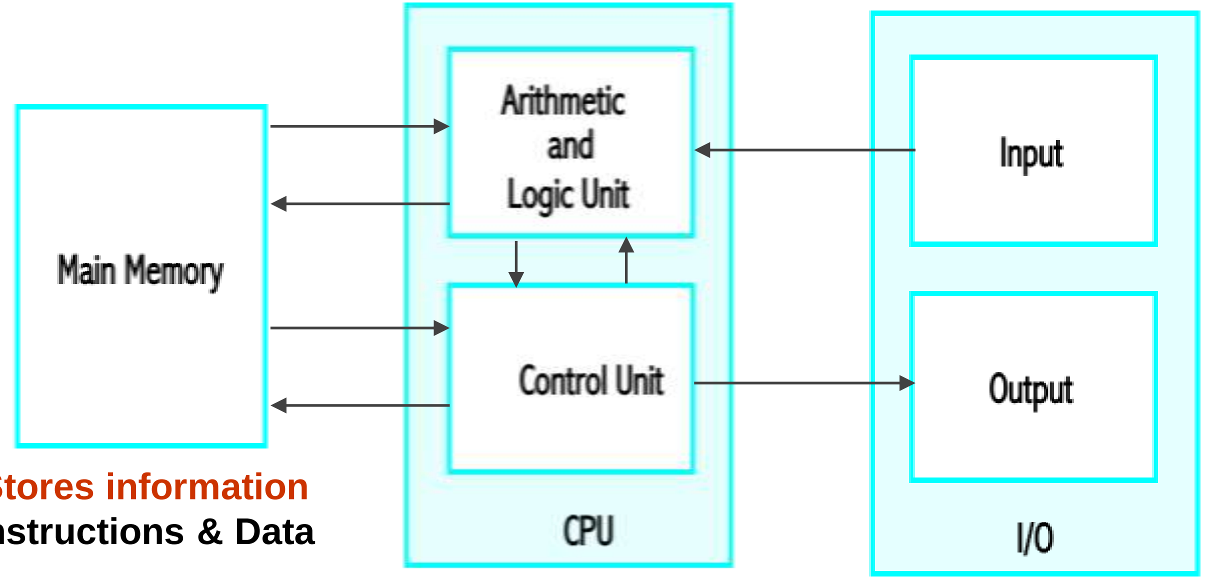
- Control Memory
- Address Sequencing
- Microprogram Example
- Design of Control Unit

Functional Blocks of Computer

Arithmetic and logic unit(ALU)
Performs the desired operations on the input information as determined by instructions in the memory

Input unit accepts information:

- **Human operators**
- **Electromechanical devices (keyboard)**
- **Other computers**



Stores information
Instructions & Data

A **control unit, interprets** the **instructions in memory** and causes them to be **executed**

Output unit sends results of processing:

- **To a monitor display/printer**

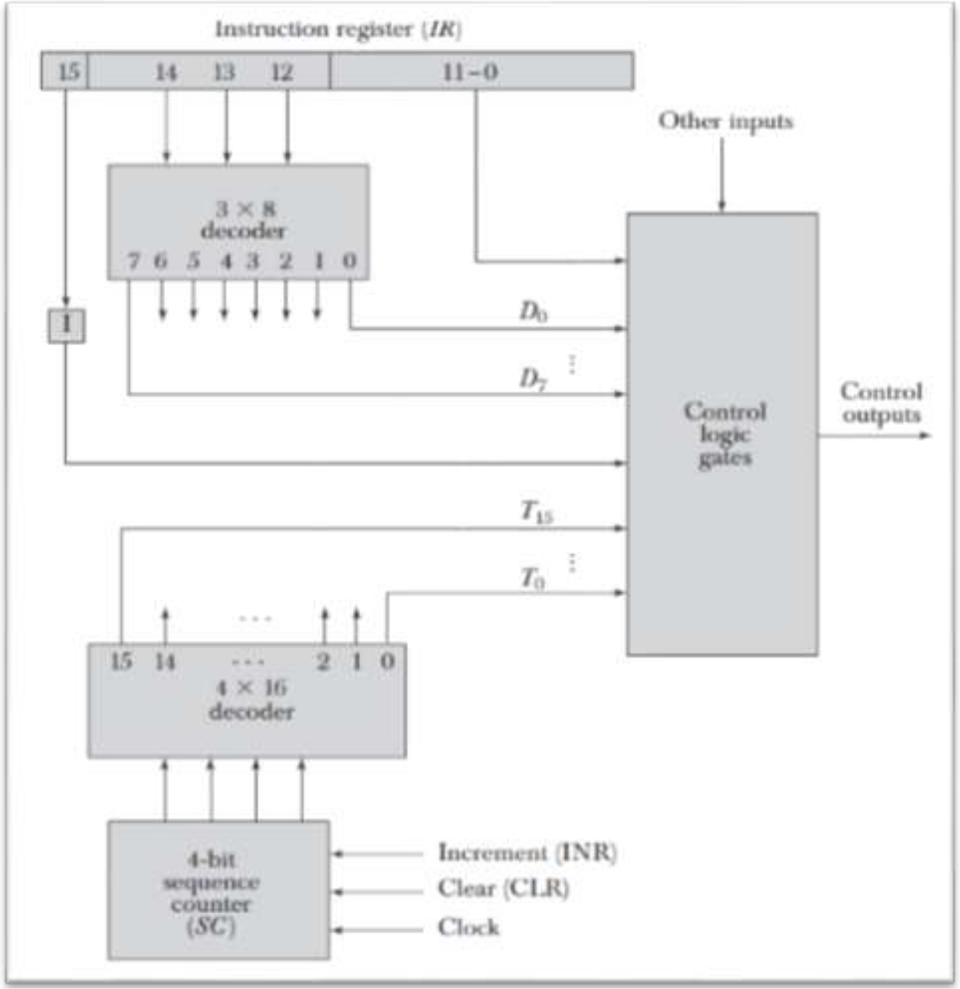
Important Terms

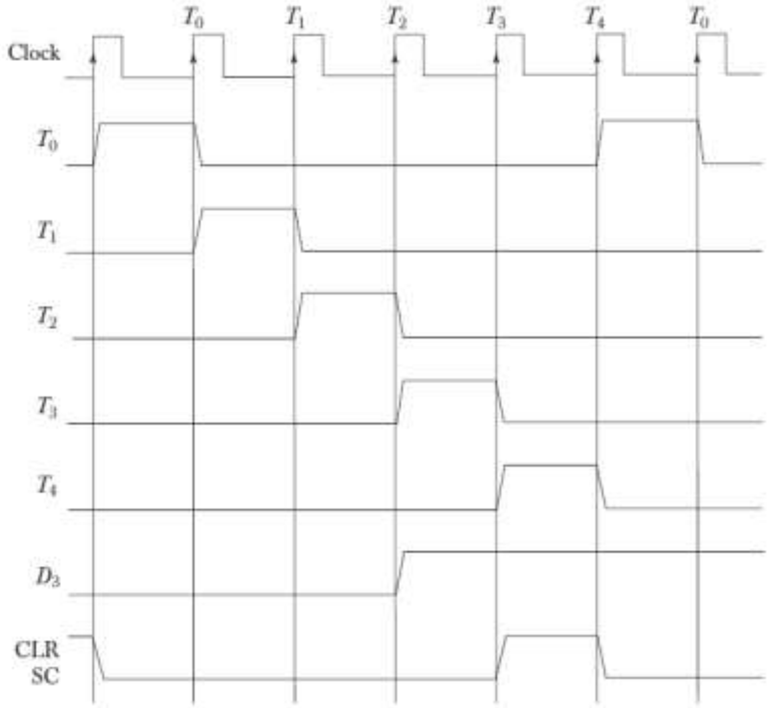
- **Hardwired Control Unit:** When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.
- **Micro programmed control unit:** Control unit whose binary control variables are stored in memory
- **Control Memory:** part of control unit that holds a fixed microprogram.
- **Microprogram:** generates all the control signals required to execute the instruction set correctly. Consists of microinstructions
- **Microinstruction:** Contains a control word and a sequencing word
    - Control Word - is control variable represented by a string of 1's and 0's. All the control information required for one clock cycle
    - Sequencing Word - Information needed to decide the next microinstruction address
- **Microoperations:** specified by Microinstruction. Low-level instructions used to implement complex machine instructions(macro-instructions)

Control Unit

- The function of the control unit in a digital computer is to initiate sequences of microoperations.
- Two methods of implementing control unit are
  - Hardwired control
  - Microprogrammed control.
- Hardwired Control
  - When control signals are generated by hardware using fixed instructions, fixed logic blocks of and/or arrays, encoders, decoders, etc., CU is said to be hardwired.
  - Key characteristics: high-speed operation, expensive, relatively complex, and no flexibility of adding new instructions.
  - Example CPUs: Intel 8085, Motorola 6802, and any RISC (Reduced Instruction Set Computer) CPUs.

Hardwired Control unit of basic computer



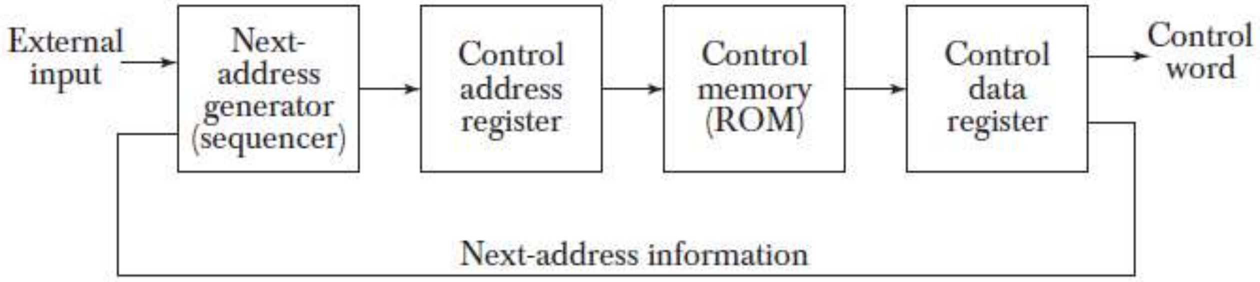


Microprogrammed Control Unit

- *Microprogramming is* a second alternative for designing the control unit of a digital computer.
  - principle of microprogramming is an elegant and systematic method for controlling the microoperation sequences in a digital computer.
- A control unit whose binary control variables are stored in memory is called a Microprogrammed control unit.
- A memory that is part of a control unit is referred to as a control memory.
  - Each word in control memory contains within it a **microinstruction.**
  - A sequence of microinstructions constitutes a **microprogram.**
  - Can be either read-only memory(ROM) or writable control memory (dynamic microprogramming)
- Main advantage - for different control sequences we only have to change microprogram residing in control memory. (No need of hardware changes)

Microprogrammed Control Unit

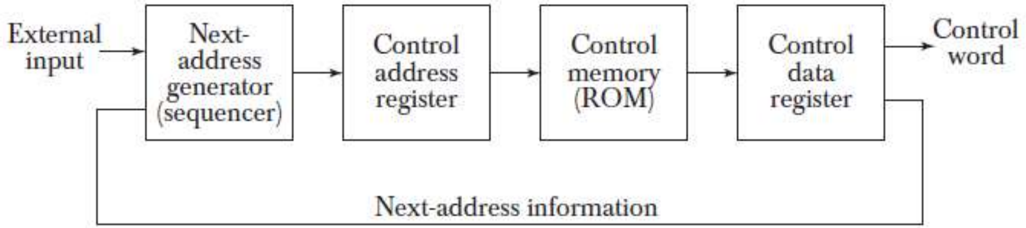
- A computer that employs microprogrammed control unit will have two separate memories.
  - **Main Memory**: available to the user for storing the programs. Contents may alter when the data are manipulated and every time that the program is changed.
  - **Control Memory**: part of a control unit that holds a fixed microprogram which cannot be altered by the user and contains various control signals.



Microprogrammed control organization.

Microprogrammed Control Unit

- Control Address Register(CAR) specifies the address of the microinstruction.
- Control Data Register(CDR) holds the microinstruction read from memory.
  - The microinstruction contains a control word that specifies one or more microoperations for the data processor.
- Once these operations are executed, next address is computed in the next address generator(sequencer) and then transferred into CAR to read the next microinstruction.
- CDR(pipeline register) holds the present microinstruction while the next address is computed and read from memory.



Microprogrammed Control Unit

- The location of the next microinstruction may be the
  - one next in sequence, or
  - it may be located somewhere else in the control memory.
- For this reason, it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction.
- The next address may also be a function of external input conditions.
- Thus, a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.

Pipeline Register

- CDR 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 CAR and the other to CDR.
- The system can operate without CDR by applying single-phase clock to CAR.
- The control word and next-address information are taken directly from the control memory.
- It must be realized that a ROM operates as a combinational circuit, with the address value as the input and the corresponding word as the output.
- Each clock pulse will execute the microoperations specified by the control word and also transfer a new address to the control address register.

Address Sequencing

- Microinstructions are stored in control memory in groups, with each group specifying **a routine**.
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction.
- **Next address generator (Microprogram Sequencer)**
  - determines the **address sequence** that is **read from control memory**
- **Typical functions** of a **microprogram sequencer** are
  - loading into CAR an address
  - incrementing the CAR by 1
  - transferring an external address, or loading an initial address to start the control operations

Executing a Single Instruction

- **Step-1 : Instruction Fetch Routine**
  - An initial address(address of first microinstruction that activates the instruction fetch routine) is loaded into CAR when power is turned on.
  - Routine is sequenced by incrementing the control address register(CAR).
  - At the end of the routine, the instruction is in the instruction register (IR)
- **Step-2 : Effective Address Computation Routine**
  - Control memory next must go through the routine that determines effective address of operand.
  - Routine can be reached through a branch microinstruction, based on the status of the mode bits of the instruction.
  - At the end of this routine, the address of the operand is in the memory address register.
- **Step-3 : Generating Microoperations that execute the fetched instruction**
  - The microoperation steps depend on the operation code part of the instruction.

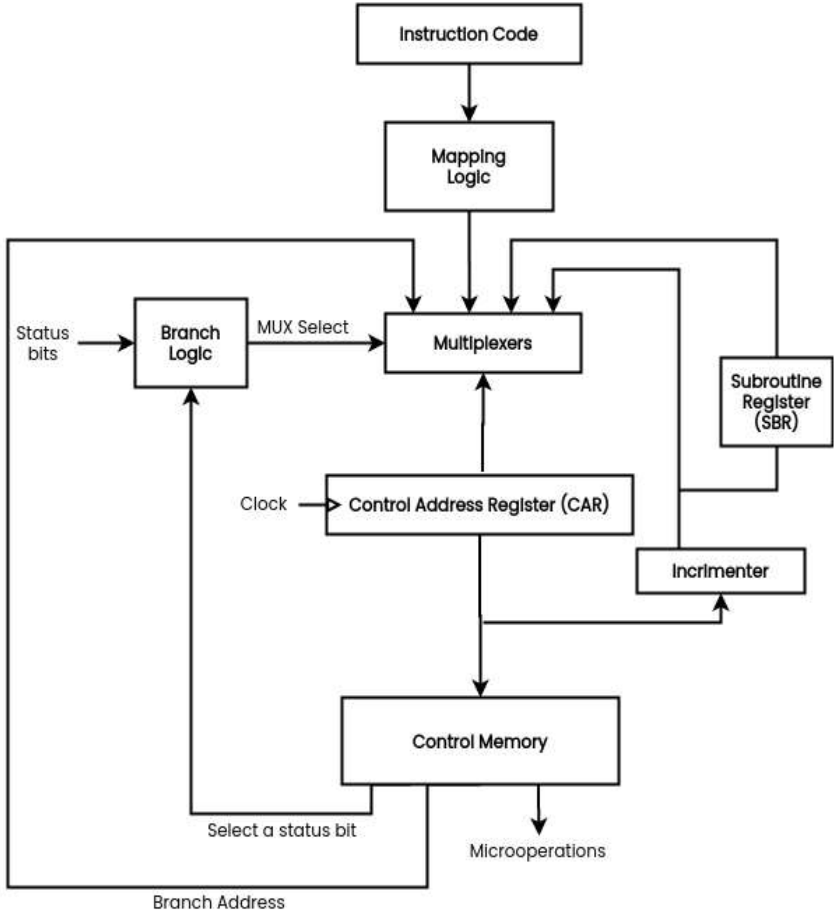
The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a **MAPPING(**rule that transforms the instruction code into a control memory address).

Subroutines

- Subroutines are programs that are used by other routines to accomplish a particular task.
- A subroutine can be called from any point within the main body of the microprogram.
    - Ex: sequence of microoperations needed to generate effective address of the operand for an instruction is common to all memory reference instructions.
    - This sequence could be a subroutine that is called from within many other routines to execute the effective address computation.
- Microprograms that employ subroutines must have a provision for **storing the return address** (cannot be stored in ROM because the unit has no writing capability).
- When the **execution of the instruction is completed**, control must **return to the fetch routine.**
- This may be accomplished by placing the incremented address from the control address register into a **subroutine register(SBR)** and executing an unconditional branch microinstruction to the first address of the fetch routine.

The **address sequencing capabilities** required in a control memory

1. Incrementing the control address register.
2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return.



Selection of next microinstruction address for control memory

- **Control Address Register (CAR)** receives the **address** from **FOUR** different paths.
- **Incrementer** increments the **content** of the **CAR** 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** depends on specific **status bit** in order **to determine** its **condition**
- **External Address** is **transferred** into **control memory** via **MAPPING Logic Circuit**
- **Return Address** for a **subroutine** is **stored** in a **Subroutine Register (SBR)**

Conditional Branch

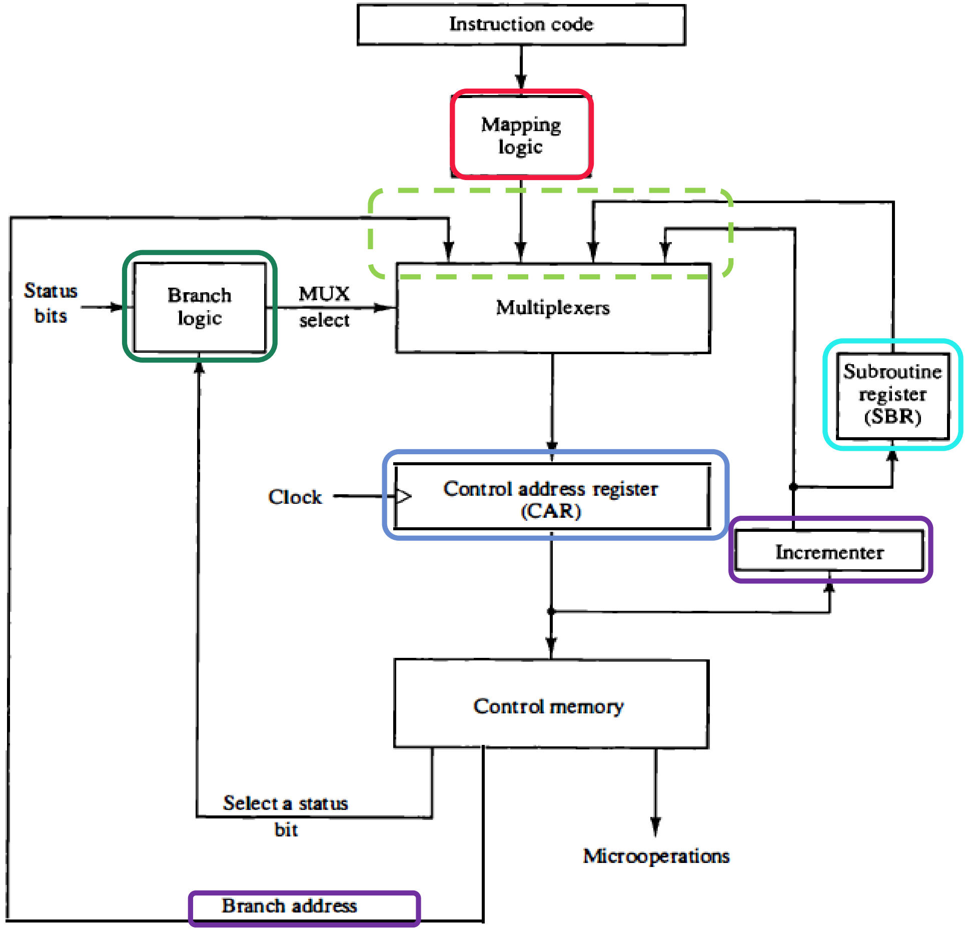
If Condition is true, then Branch (address from the next address field of the current microinstruction)

else Fall Through

Conditions to Test: N(negative), Z(zero), C(carry), etc.

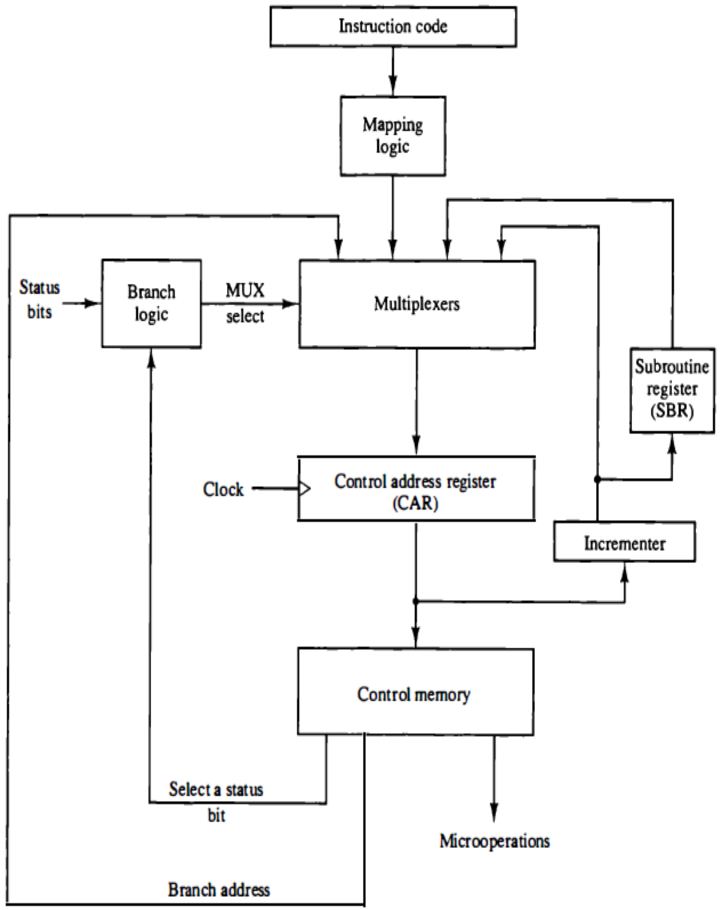
Unconditional Branch

Fixing the value of one status bit at the input of the multiplexer to 1



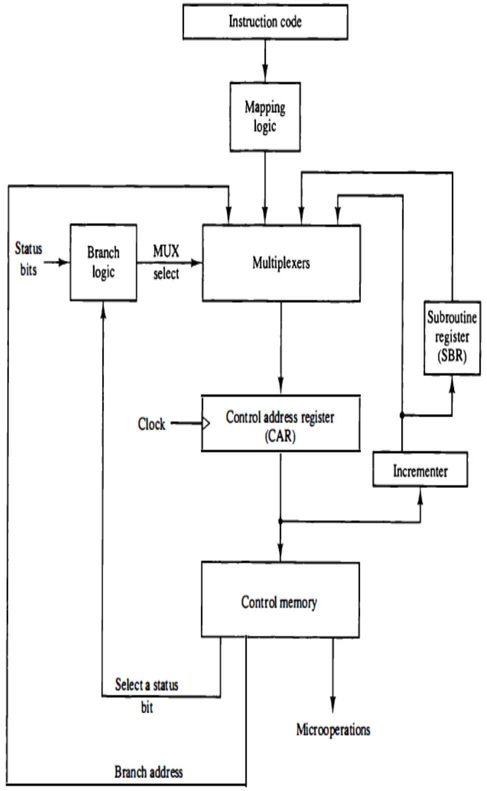
Conditional Branching

- Branch logic provides decision-making capabilities in the control unit.
- The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions.
- The status bits provide parameter information such as
  - carry-out of an adder
  - sign bit of a number
  - mode bits of an instruction
  - input or output status conditions.
- Simplest way to implement branch logic is to test the specified condition and branch to an address if the condition is met; else increment the address register.



Branch Logic

- Branch Logic hardware may be implemented in a variety of ways
  - Simplest is to test the specified condition and branch to the indicated address if the condition is met;
  - otherwise, the address register is incremented
  - This can be implemented with a Multiplexer
- Assume, there are EIGHT status bit CONDITIONS in the system
  - Three bits in the microinstruction are used to specify any one of eight status bit conditions.
  - These three bits provide the selection variables for the multiplexer.
  - If the selected status bit is in the 1 state, multiplexer transfer the branch address into CAR
  - Otherwise, it is 0, multiplexer causes the address register to be incremented
- An unconditional branch microinstruction can be implemented by loading the branch address from control memory into CAR.



Mapping of Instruction

- Ex: A computer instruction has an **operation code of 4 bits** (specifies 16 instructions)
- Assume Control Memory has 128 words (requires 7 bits for address)
- For each **Operation Code** there exists a **Microprogram Routine** in control memory
- One simple MAPPING PROCESS that converts the 4-bit operation code to a 7-bit address for Control Memory is shown in Fig.
  - This mapping consists of placing a 0 in the MSB of the address
  - Transferring the four operation code bits, and clearing the two LSBs of CAR
- This provides for each Computer Instruction a Microprogram Routine with a capacity of Four Microinstructions

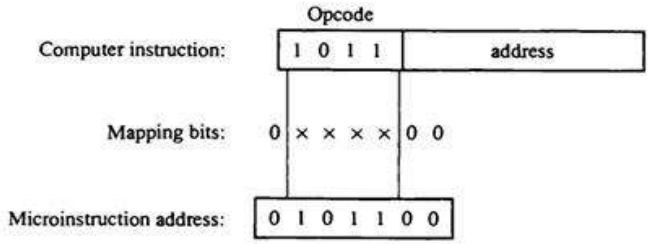
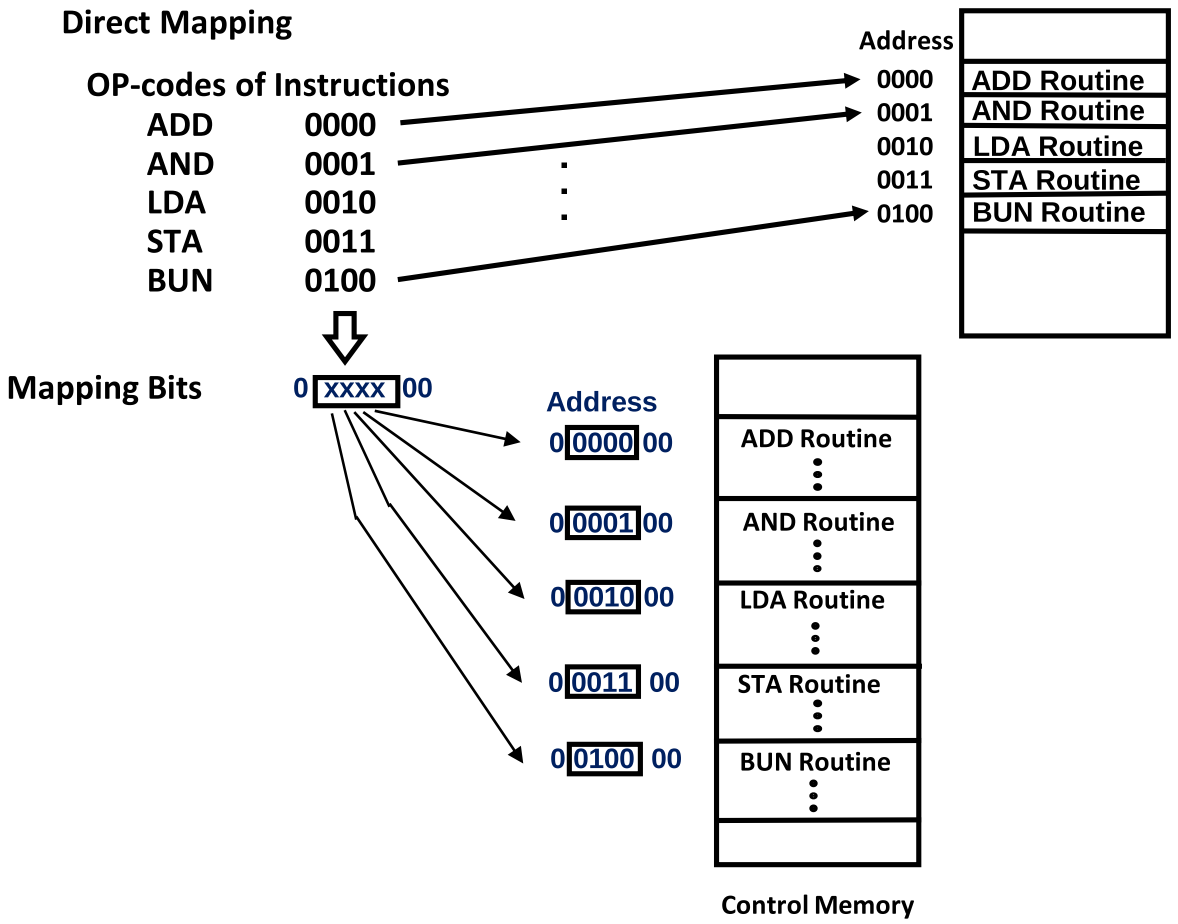


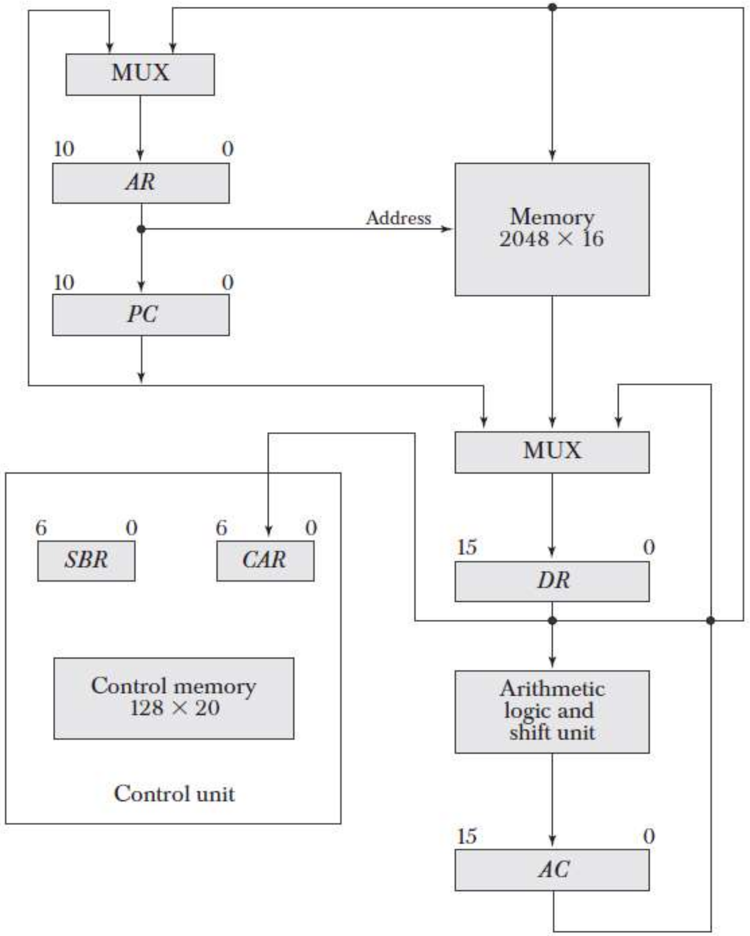
Fig.: Mapping from Instruction Code to Microinstruction Address

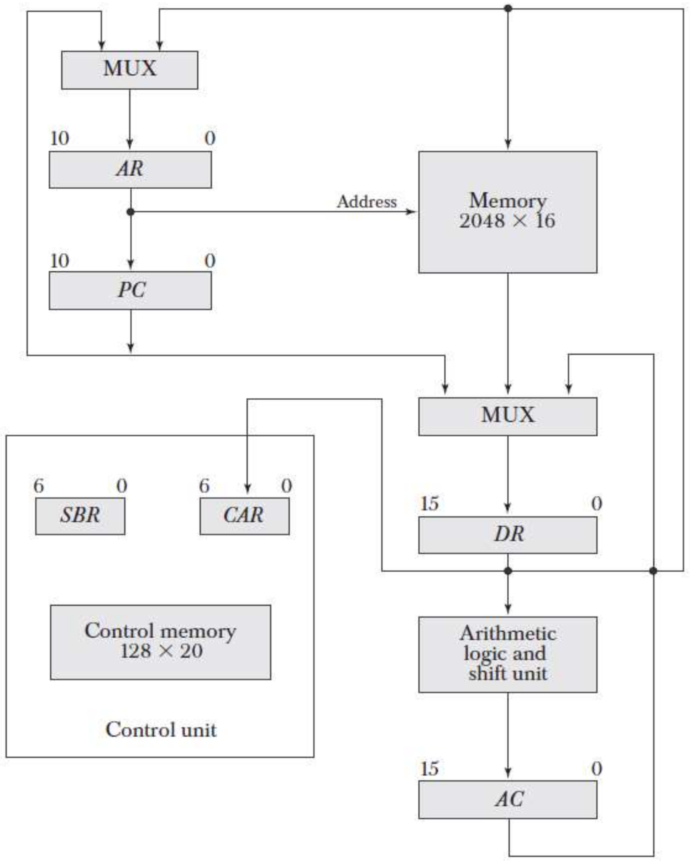
Mapping of Instructions



Microprogram Example

- The block diagram of the **computer** hardware **configuration** is shown in **Fig**.
- It consists of **two memory units**:
  - **main memory** for **storing instructions** and **data**
  - **control memory** for **storing** the **microprogram**
- **Four registers(AR, PC, DR, AC)** are associated with the **processor unit**
- **Two** with the **control unit**
  - Control Address Register **CAR**
  - Subroutine register **SBR**





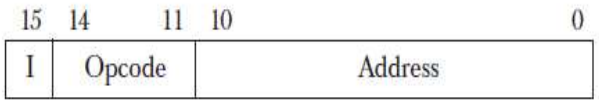
The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

- *DR* can receive information from *AC, PC, or memory.*
- *AR* can receive information from *PC or DR.*
- *PC* can receive information only from *AR.*

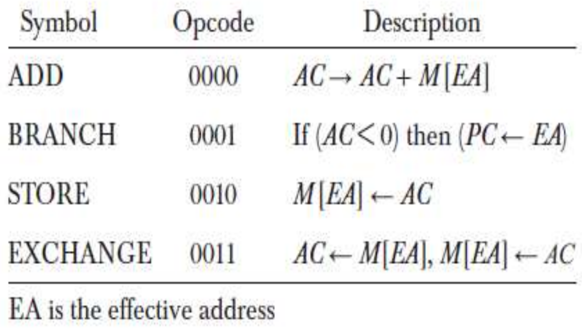
The arithmetic, logic, and shift unit performs microoperations with data from *AC* and *DR* and places the result in *AC.*

- Memory address are stored in *AR.*
- Input data written to memory come from *DR.*
- Data read from memory can go only to *DR*.

Microinstruction Format



(a) Instruction format



Symbol	Opcode	Description
ADD	0000	$AC \rightarrow AC + M[EA]$
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

The computer instruction format is depicted in Fig. (a).

Instruction Format consists of three fields:

- 1-bit field for indirect addressing (*I*)
- 4-bit operation code (opcode)
- 11-bit address field.

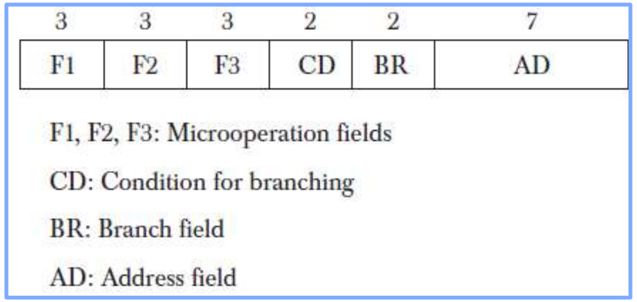
Figure.(b) lists four instructions out of 16 possible memory-reference instructions.

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

"Each computer instruction must be microprogrammed."

Microinstruction Format

- Generating microcode for the control memory is called microprogramming.
- **Microinstruction** contains **20 bits** divided into **four** functional parts.
- Three fields **Fl, F2, and F3 specify microoperations** for the computer.
    - Each field is **3 bits** which provides **21 microoperations**.
- **CD** : selects status bit conditions.
    - 2 bits are encoded to specify 4 status bits conditions.
- **BR** : specifies the type or branch to be used.
    - Used in conjunction with the AD field, to choose the address of the next microinstruction.
- **AD** : contains a branch address. Since the control memory has 128 = 2^7 words, AD has 7 bits.



3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

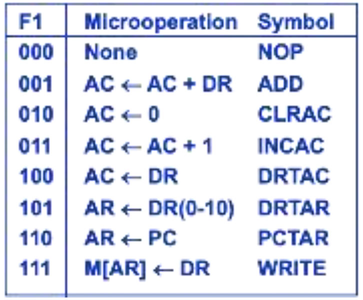
F1, F2, F3: Microoperation fields

CD: Condition for branching

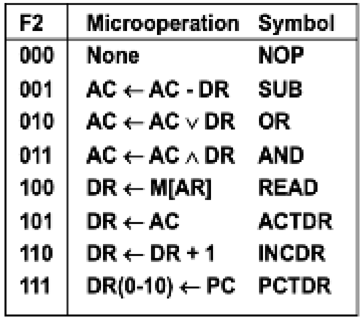
BR: Branch field

AD: Address field

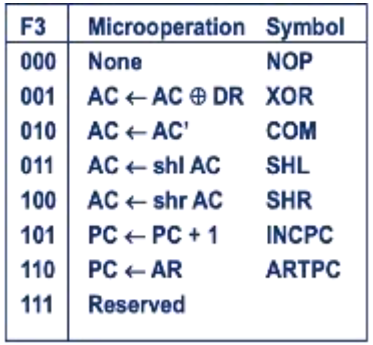
Microinstruction Field Description



F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE



F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR



F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

- No more than three microoperations can be chosen for a microinstruction, one from each field.
- If fewer than three microoperations are used, one or more of the fields will use the binary code 000 for no operation.
- As an illustration, a microinstruction can specify two simultaneous microoperations from F2 and F3 and none from F1.
- The nine bits of the microoperation fields will then be 000 100 101.
- Two or more conflicting microoperations CANNOT be specified simultaneously. For example,

a microoperation field 010 001 000 has NO MEANING

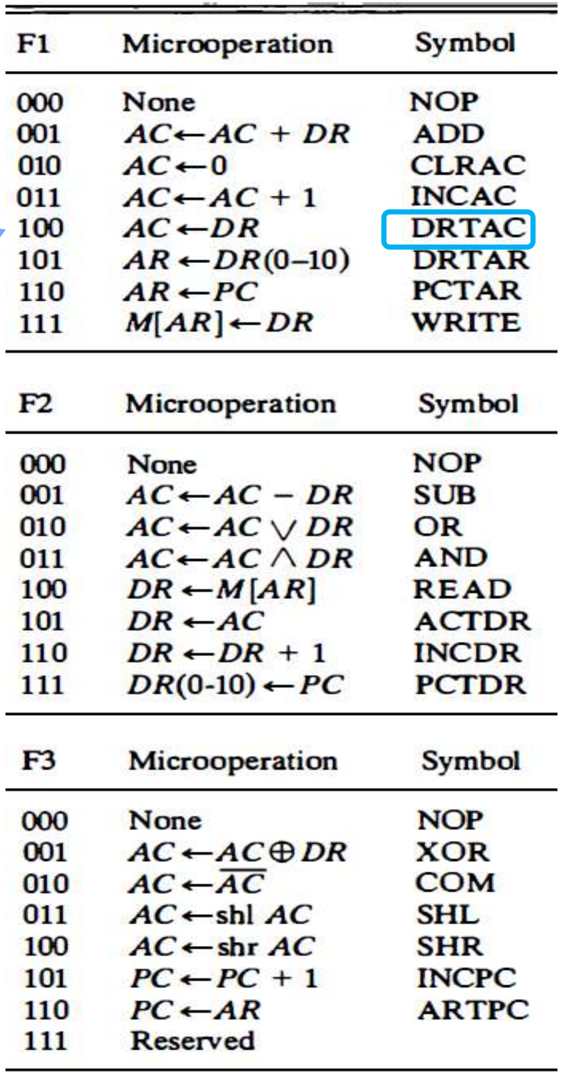
AC ←0 (F1) AC ← AC-DR (F2) NOP

because it specifies the operations to clear AC to 0 and subtract DR from AC at the same time.

- **TRANSFER-TYPE microoperations** symbols use **FIVE letters**
    - **First Two Letters** designate the **Source Register**
    - **Third Letter** is always a **‘T’**
    - **Last Two Letters** designate the **Destination Register**

For example,

- **microoperation** that specifies the **transfer AC 🡐 DR (F1 = 100)** has the symbol **DRTAC** which stands for a transfer from **DR to AC**



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	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Microinstruction Field Description

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

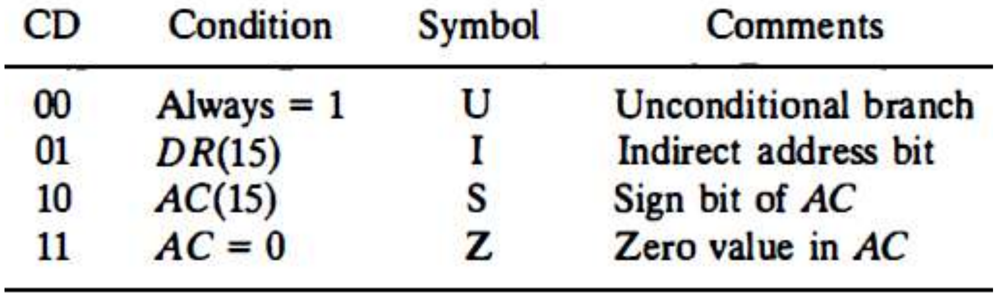
F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC’	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

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

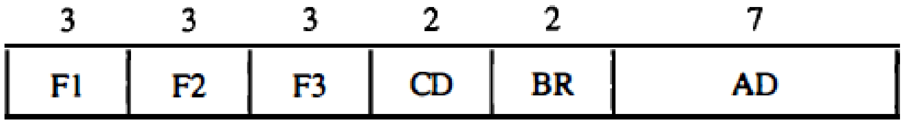
BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1
		CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Condition field

- **CD (Condition) field** consists of **2 bits**
- These bits are **encoded** to specify **4 Status Bit Conditions** (as listed in Table)
  - **U, I, S** and **Z**
- **First condition** is always **1**
  - So, **CD = 00** (symbol **U**) will always find the **condition to be TRUE**
  - In **conjunction** with **BR field**, it provides an **UNCONDITIONAL BRANCH** operation
- **Indirect bit 'I'** (bit 15 of DR) after an instruction **is read from memory**
- **Sign bit** of **AC** (bit 15 of AC) provides the **next status bit**
- **Zero value 'Z'** is a **binary variable** whose value is **equal to 1** if **all the bits in AC = 0**



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

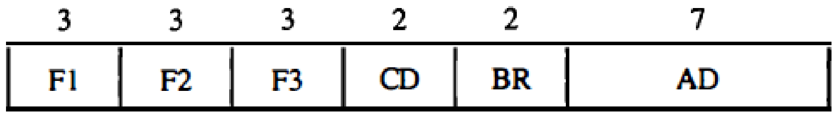


3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

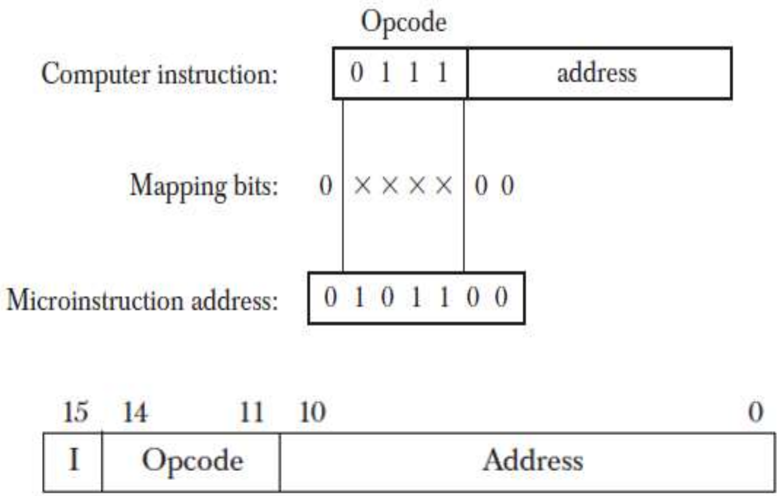
Branch field

- **BR (Branch) field** consists of **2 bits**
- It is used, in conjunction with the **AD field,** to choose the **address** of the **next microinstruction.**
    - When **BR = 00,** the control performs a **jump (JMP) operation** (which is similar to a branch)
    - When **BR = 01,** it performs a **call to subroutine (CALL)** operation
- **Two operations are identical except that**
    - **a CALL microinstruction** stores the **RETURN address** in the **Subroutine Register (SBR)**
- **JUMP and CALL operations depend on the value of the CD field**
    - **If CD = 01 (indirect address bit),** the next address in the **AD field is transferred to CAR**
    - Otherwise, **CAR is incremented by 1**
- The **return from subroutine** is accomplished when **BR = 10.** This causes the **transfer of the return address from SBR to CAR.**

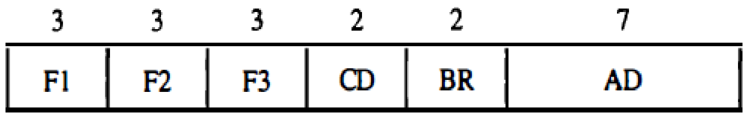
BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$



Branch field



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



Mapping from **operation code bits of instruction** to an address for **CAR** is accomplished when **BR = 11.**

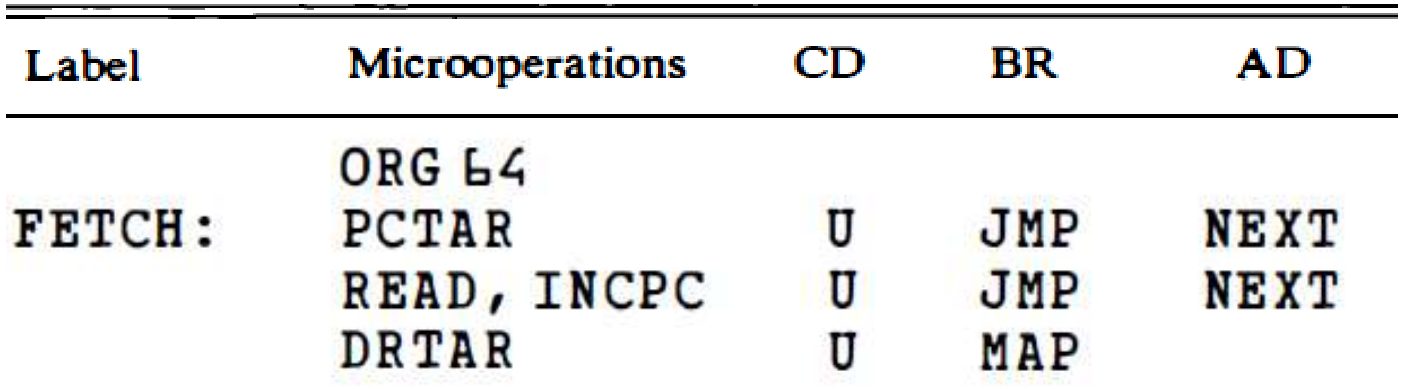
The bits of the operation code are in **DR(11–14)** after an instruction is read from memory.

Note that the Last two conditions in the BR field are independent of the values in the CD and AD fields.

Symbolic Microinstructions

- **Each line** of the **Assembly Language Microprogram** defines a **symbolic microinstruction.**
- Symbolic microprogram can be translated into its binary equivalent by means of an **Assembler.**
- Each symbolic microinstruction is divided into five fields
  - **Label**: may be empty or have a symbolic address. It is terminated with a colon (:)
  - **Microoperations**: consists of one, two, or three symbols. There may be no more than one symbol from each **F** field. **NOP** symbol is used when the microinstruction has no microoperations.
  - **CD**: has one of the letters **U, I, S, or Z**.
  - **BR**: contains one of the four symbols **JMP, CALL, RET, MAP**
  - **AD**: specifies a value for the address field of the microinstruction in one of three possible ways:
    - With a **symbolic address**, which must also appear as a **label**.
    - With the symbol **NEXT** to designate the **next address in sequence**.
    - When the BR field contains a **RET or MAP** symbol, the AD field is **left empty**.

- **ORG** is used to define the origin, or **first address** of a **microprogram routine**.
- ORG 64 informs the assembler to place the next microinstruction in control memory at decimal address 64, which is equivalent to the binary address 1000000.



Label	Microoperations	CD	BR	AD
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	

Symbolic Microprogram - Fetch Routine

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

Sequence of microoperations in the fetch cycle:

$AR \leftarrow PC$
$DR \leftarrow M[AR], PC \leftarrow PC + 1$
$AR \leftarrow DR(0\text{-}10), CAR(2\text{-}5) \leftarrow DR(11\text{-}14), CAR(0,1,6) \leftarrow 0$

Symbolic microprogram for the fetch cycle:

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

Binary equivalents translated by an assembler

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

The Fetch Routine(Fetch & Decode)

- Control Storage: 128 20-bit words
- First 64 words(0-63): Routines for the 16 machine instructions(EX: ADD, BRANCH, STORE, EXCHANGE)
- Last 64 words(64-127): Used for other purpose (e.g., fetch routine and other subroutines)
- Mapping: OP-code XXXX into 0XXXX00, the first address for the 16 routines are
  0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20, ..., 60

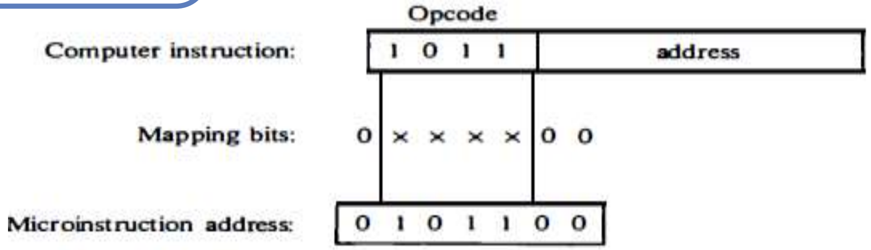
A convenient starting location for the **FETCH routine** is **address 64**

MICROINSTRUCTIONS needed for the **FETCH Routine** are:

$AR \leftarrow PC$

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

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



(Microinstruction-1) $AR \leftarrow PC$

(Microinstruction-2) $DR \leftarrow M[AR], \quad PC \leftarrow PC + 1$

(Microinstruction-3) $AR \leftarrow DR(0\text{–}10), \quad CAR(2\text{–}5) \leftarrow DR(11\text{–}14), \quad CAR(0,1,6) \leftarrow 0$

- **FETCH routine** needs **THREE MICROINSTRUCTIONS**, which are **placed** in **control memory** at **addresses 64, 65, and 66**

- Using the assembly language conventions, the **SYMBOLIC MICROPROGRAM** for the **FETCH routine** :

```
               ORG 64
(64)   FETCH:  PCTAR          U    JMP    NEXT
(65)           READ, INCPC    U    JMP    NEXT
(66)           DRTAR          U    MAP
```

$AR \leftarrow PC$
$DR \leftarrow M[AR], \quad PC \leftarrow PC + 1$
$AR \leftarrow DR(0-10), \quad CAR(2-5) \leftarrow DR(11-14), \; CAR(0,1,6) \leftarrow 0$

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

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	$AR \leftarrow DR(0–10)$	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	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

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

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

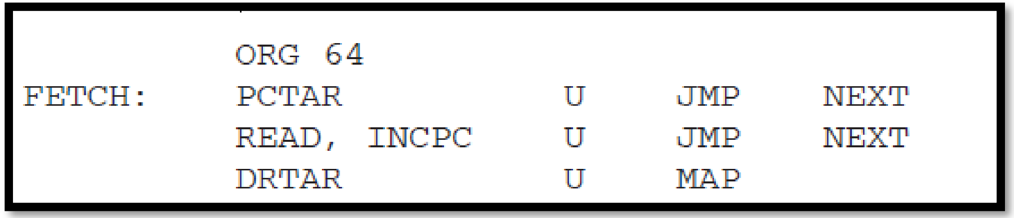
3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

Translation of the **symbolic microprogram** to **BINARY** produces the following **BINARY MICROPROGRAM:**

	Binary Address	F1	F2	F3	CD	BR	AD
(64)	1000000	110	000	000	00	00	1000001
(65)	1000001	000	100	101	00	00	1000010
(66)	1000010	101	000	000	00	11	0000000

NOTE

- The three microinstructions that constitute the fetch routine are listed in three different representations.
- The **register transfer representation** shows the **internal register transfer operations** that each microinstruction implements.
- The **symbolic representation** is useful for writing microprograms in an **assembly language format.**
- **The binary representation** is the **actual** internal **content** that must be **stored in control memory**.
- It is customary to write **microprograms in symbolic form** and then use an **assembler** program to obtain a **translation to binary.**



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

- The execution of the third **(MAP) microinstruction** in the **fetch routine** results in a branch to address **0xxxx00**, where xxxx are the **four bits of the operation code**.

- For example, if the instruction is an **ADD** instruction whose operation code is **0000**, the **MAP** microinstruction will **transfer to CAR the address 0000000**, which is the **start address** for the ADD routine in control memory.

- The **first address** for the **BRANCH and STORE routines** are **0 0001 00 (decimal 4)** and **0 0010 00 (decimal 8),** respectively.

- The first address for the other **13** routines are at address values **12, 16, 20, . . . , 60.**

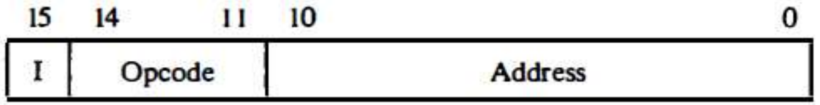
- This gives **four words in control memory for each routine.**

Evaluating Effective Address

- In each routine, several **MICROINSTRUCTIONS** are provided for
  - **evaluating the effective address** and
  - **executing the instruction**
- If the **microinstructions** for the **indirect address** are stored **as** a **subroutine**
  - it is symbolized by **INDRCT**
  - and is located **after fetch routine**, (shown in Below Table)

Symbolic Microprogram (Partial)

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	



(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

- **Fig (a) COMPUTER INSTRUCTION format**
- **Fig (b)** lists **FOUR** of the **16 possible Memory-Reference instructions**
- **each Computer Instruction** must be **Microprogrammed**

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

- To see how the transfer and return from the indirect subroutine occurs, assume **MAP microinstruction** at the **end of the fetch routine** caused a branch to **address 0**, where the **ADD routine is stored**.
- The **first microinstruction** in the **ADD** routine **calls** subroutine **INDRCT**, conditioned on **status bit I.**
- If **I=1, branch to INDRCT** occurs and **return address** (address **1**) is stored in **SBR**.
- The INDRCT subroutine has two microinstructions:

```
INDRCT:    READ    U     JMP     NEXT
           DRTAR   U     RET
```

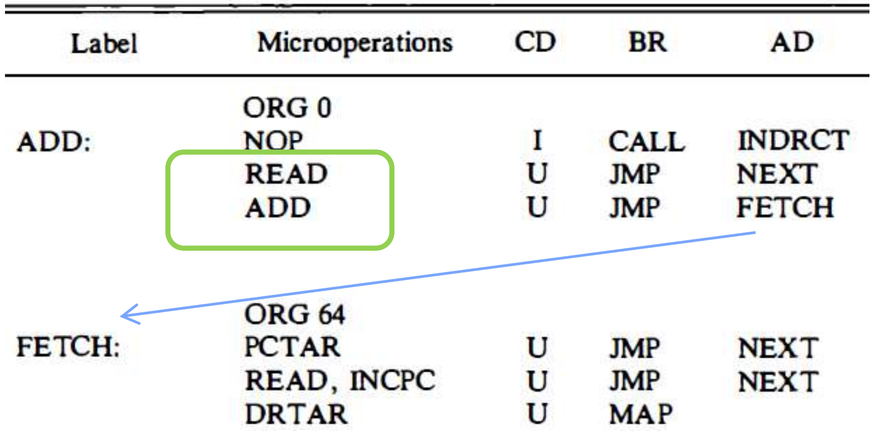
- **Memory** has to be **accessed (READ)** to get the **Effective Address**
- then **transferred** to **AR (DRTAR)**
- **Return** from **subroutine (RET) transfers** the **address from SBR to CAR**
- Thus **returning** to the **Second microinstruction** of the **ADD routine**

- **ADD instruction** execution is carried out by the **microinstructions** at **addresses 1** and **2**

- **First MICROINSTRUCTION**
  - **READS** the **operand** from memory **into DR**

- **Second MICROINSTRUCTION**
  - **performs** an **ADD microoperation** with the content of **DR** and **AC** and
  - then **JUMPS back** to the **beginning of the FETCH routine**

(addresses 0)
(addresses 1)
(addresses 2)



Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	

TABLE: Binary Microprogram for Control Memory (Partial)

Micro Routine	Address		Binary Microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

TABLE: Symbolic Microprogram (Partial)

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

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	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

- The **Binary microprogram** listed in Table specifies the **WORD CONTENT** of the **Control Memory**
- When a **ROM** is used **for** the **control memory**,
  - the **microprogram binary list** provides the **truth table** for **fabricating** the unit
  - **Bits** of **ROM are FIXED (internal links** are **fused** during the **hardware production**)
- **To modify the instruction set** of the computer,
  - it is necessary to **generate a NEW MICROPROGRAM** and **mask a NEW ROM**

- If a **WRITABLE control memory** is employed,
  - the **ROM** is **REPLACED** by a **RAM**

- **Advantage** of employing a **RAM** for the control memory is that
  - **Microprogram** can be **altered** simply by **writing** a **new pattern** of 1's and 0's **without resorting to hardware procedures**
  - There is **flexibility** of **choosing** the **instruction set** of a computer **dynamically** by **changing** the microprogram under processor control

- Most microprogrammed systems use a **ROM** for the **control memory**
  - because it is **CHEAPER** and **FASTER** than a RAM
  - and **to PREVENT the occasional user from CHANGING the ARCHITECTURE of the system**

Design of Control Unit – Issues

- **The number of control bits that initiate microoperations can be reduced by grouping mutually exclusive variables into fields** and encoding the K bits in each field to provide 2^K microoperations.

- **Each field requires a decoder** to produce the corresponding **control signals** which **reduces the size of the microinstruction bits** but requires **additional hardware** external to the control memory.

- As the control signals propagate through the decoding circuits **delay increases**.

- The encoding of control bits was demonstrated in the programming example of the preceding section.

Design of Control Unit

- Bits of the microinstruction are divided into fields, with each field defining a separate function.(F1-F2-F3, CD, BR, AD)
- Each field requires a decoder to produce the corresponding control signals.
- Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3x8 decoder to provide eight outputs.
- Each of these outputs must be connected to the proper circuit to initiate the corresponding microoperation.

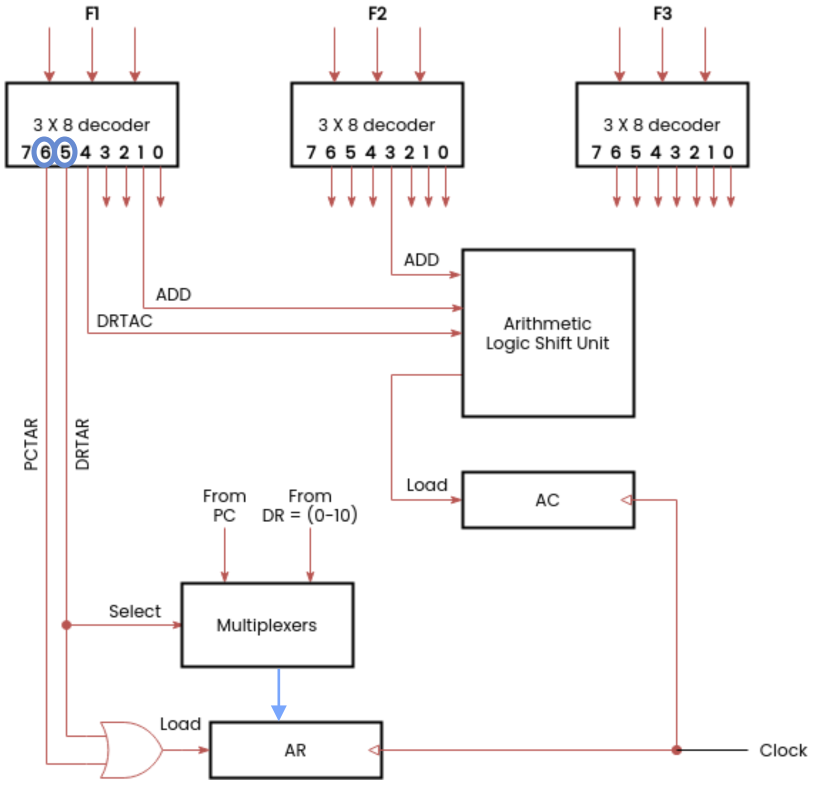


Fig. Decoding of microoperation fields

Microprogram Sequencer

- The basic components of microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection part is called as microprogram sequencer.
- Purpose of a microprogram sequencer is
  - to present an ADDRESS to the control memory
  - read and execute microinstruction
- The next address logic of the sequencer determines the specific address source to be loaded into the CAR.
- The choice of the address source is guided by the next address information bits that the sequencer receives from the present microinstruction.

MICROPROGRAM SEQUENCER

There are **two multiplexers** in the circuit

Mux1

- selects an **address** from one of **four sources**
- **routes** it into a control address register **CAR**

MUX2

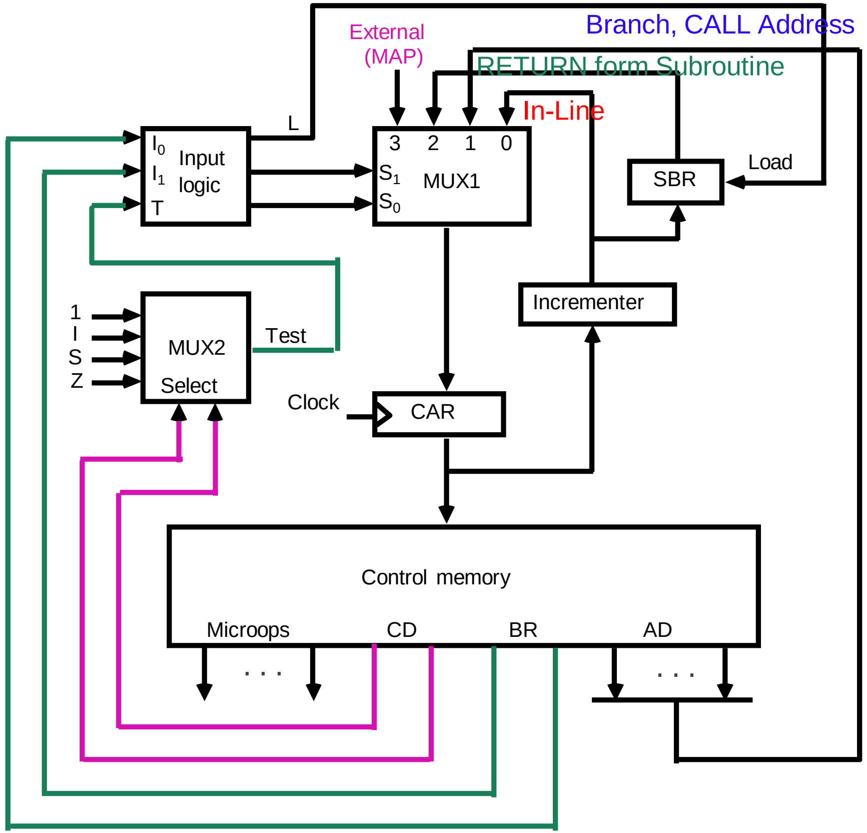
- **tests** the value of a selected **status bit**
- **result** of the test is applied to an **input logic** circuit

Output from **CAR** provides the **address** for the **control memory**

Content of CAR is **incremented** and applied to **MUX1 inputs** and to the **SBR**

Other three inputs to **MUX1** come

- from the **address field** of **present microinstruction**
- from the **output of SBR**
- from an **external source** that **MAPS** the **instruction**

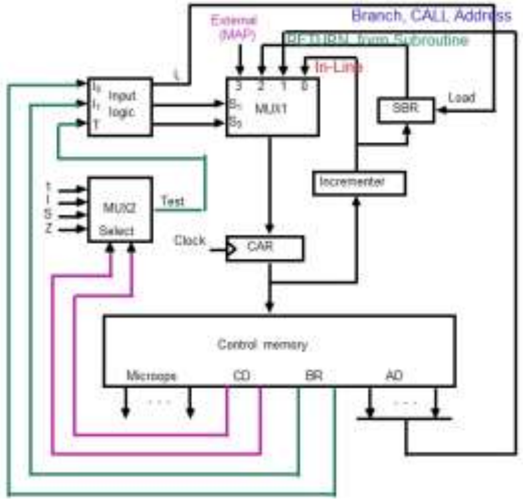


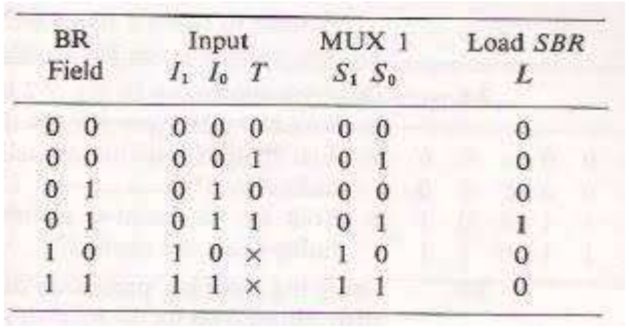
MICROPROGRAM SEQUENCER

- The truth table for the input logic circuit is shown in Table.
- Inputs I1 and I0 are identical to the bit values in the BR field.
- The bit values for S1and S0 are determined from the stated function and the path in the multiplexer that establishes the required transfer.
- The subroutine register is loaded with the incremented value of CAR during a call microinstruction (BR 01) provided that the status bit condition is satisfied (T 1).
- The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

$$S_1 = I_1$$
$$S_0 = I_1 I_0 + I_1' T$$
$$L = I_1' I_0 T$$

The circuit can be constructed with three AND gates, an OR gate, and an inverter.



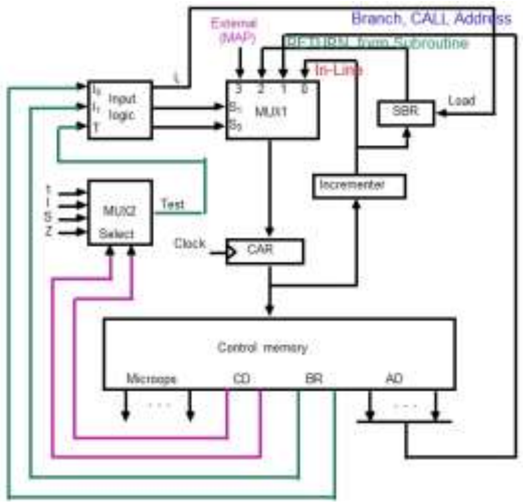


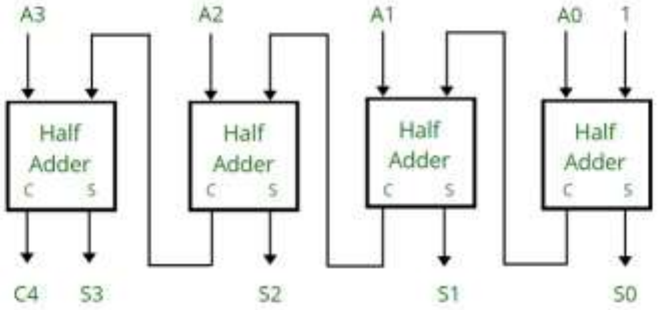
BR Field	Input I_1	I_0	T	MUX 1 S_1	S_0	Load SBR L
0 0	0	0	0	0	0	0
0 0	0	0	1	0	1	0
0 1	0	1	0	0	0	0
0 1	0	1	1	0	1	1
1 0	1	0	×	1	0	0
1 1	1	1	×	1	1	0

Input Logic Truth Table for Microprogram Sequencer

MICROPROGRAM SEQUENCER

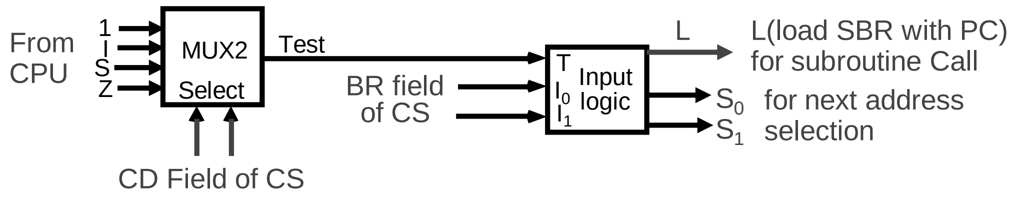
- The incrementer circuit in the sequencer is not a counter constructed with flip-flops but rather a combinational circuit constructed with gates.
- A combinational circuit incrementer can be designed by cascading a series of half-adder.
- The output carry from one stage must be applied to the input of the next stage.
- One input in the first least significant stage must be equal to 1 to provide the increment-by one operation.





4- Bit Binary Incrementer

MICROPROGRAM SEQUENCE - CONDITION AND BRANCH CONTROL -



Input Logic

I_0I_1T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	10	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	10	1
10x	RET	SBR	01	0
11x	MAP	DR(11-14)	11	0

$$S_0 = I_0$$
$$S_1 = I_0I_1 + I_0'T$$
$$L = I_0'I_1T$$