Unit-II

MICROPROGRAMMED CONTROL

Introduction:

- ➢ The function of the control unit in a digital computer is to initiate sequence of microoperations.
- ➢ Control unit can be implemented in two ways
  - o Hardwired control
  - o Micro programmed

controlHardwired Control:

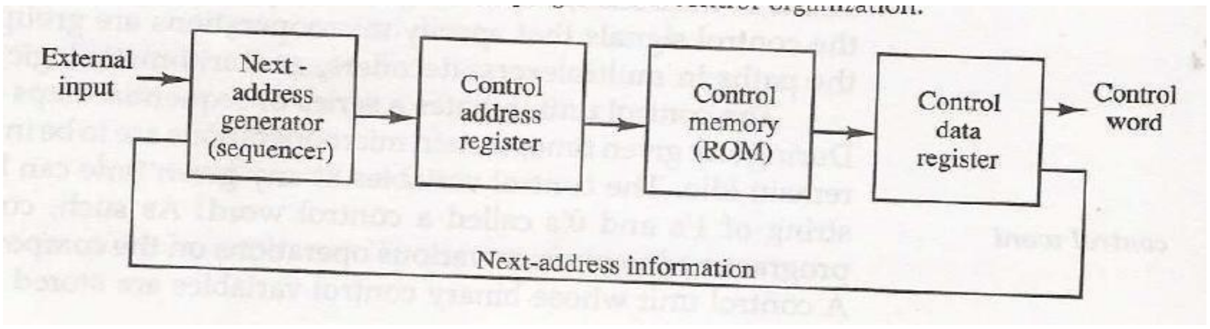
- ✓ When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be *hardwired.*
- ✓ The key characteristics are
  - o High speed of operation
  - o Expensive
  - o Relatively complex
  - o No flexibility of adding new instructions
- ✓ Examples of CPU with hardwired control unit are Intel 8085, Motorola 6802, Zilog 80, and any RISC CPUs

- ✓ .Microprogrammed Control:

- ✓ Control information is stored in control memory.
- ✓ Control memory is programmed to initiate the required sequence of micro-operations.
- ✓ The key characteristics are
  - o Speed of operation is low when compared with hardwired
  - o Less complex
  - o Less expensive
  - o Flexibility to add new instructions
- ✓ Examples of CPU with microprogrammed control unit are Intel 8080, Motorola 68000 and any CISC CPUs.

1. Control Memory:

- ➢ The control function that specifies a microoperation is called as ***control variable.***
- ➢ When control variable is in one binary state, the corresponding microoperation is executed. For the other binary state, the state of registers does not change.
- ➢ The active state of a control variable may be either 1 state or the 0 state, depending on the application.
- ➢ For bus-organized systems the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units.
- ➢ ***Control Word:*** The control variables at any given time can be represented by a string of 1's and 0's called a control word.
- ➢ All control words can be programmed to perform various operations on the components of the system.
- ➢ ***Microprogram control unit:*** A control unit whose binary control variables are stored in memory is called a microprogram control unit.
- ➢ The control word in control memory contains within it a *microinstruction.*
- ➢ The microinstruction specifies one or more micro-operations for the system.
- ➢ A sequence of microinstructions constitutes a *microprogram*.
- ➢ The control unit consists of control memory used to store the microprogram.
- ➢ Control memory is a permanent i.e., read only memory (ROM).
- ➢ The general configuration of a micro-programmed control unit organization is shown as block diagram below



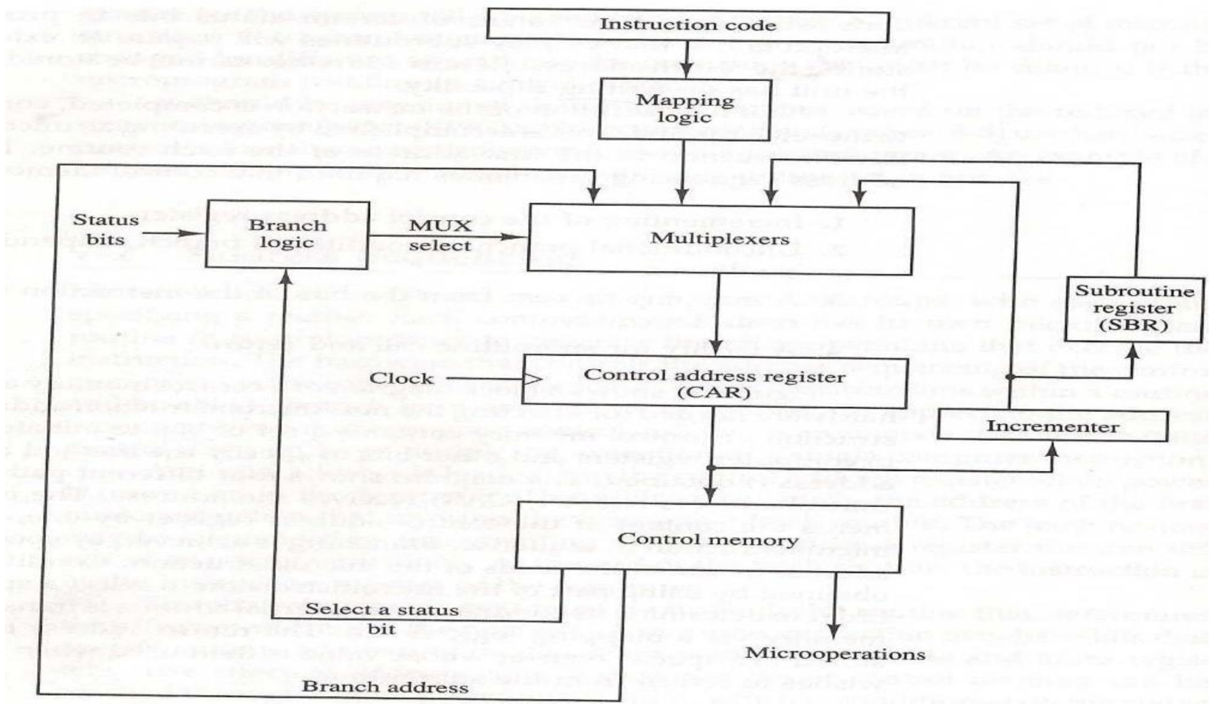
- The control memory is ROM so all control information is permanently stored.
- The control memory address register (CAR) specifies the address of the microinstruction and the control data register (CDR) holds the microinstruction read from memory.
- The next address generator is sometimes called a microprogram sequencer. It is used to generate the next micro instruction address.
- The location of the next microinstruction may be the one next in sequence or it may be located somewhere else in the control memory.
- So, it is necessary to use some bits of the present microinstruction to control the generation of the address ofthe microinstruction.
- Sometimes the next address may also be a function of external input conditions.
- The control data register holds the present microinstruction while next address is computed and read from memory. The data register is times called a *pipeline register.*

- A computer with a microprogrammed control unit will have two separate memories: a main memory and a control memory
- The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations
- These microinstructions generate the microoperations to:
  - fetch the instruction from main memory
  - evaluate the effective address
  - execute the operation
  - return control to the fetch phase for the next instruction

2. Address Sequencing:

- Microinstructions are stored in control memory in groups, with each group specifying a *routine.*
- Each computer instruction has its own microprogram routine to generate the microoperations.
- The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another
- Steps the control must undergo during the execution of a single computer instruction:
  - Load an initial address into the CAR when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction fetch routine – IR holds instruction
  - The control memory then goes through the routine to determine the effective address of the operand – AR holds operand address
  - The next step is to generate the microoperations that execute the instruction by considering the opcode and applying a *mapping process.*
    - *The transformation of the instruction code bits to an address in control memory where the routine of instruction located is referred to as mapping process.*
  - After execution, control must return to the fetch routine by executing an unconditional branch
- In brief the address sequencing capabilities required in a control memory are:
  - Incrementing of the control address register.
  - Unconditional branch or conditional branch, depending on status bit conditions.
  - A mapping process from the bits of the instruction to an address for control memory.
  - A facility for subroutine call and return.

- The below figure shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.



- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.
- In the figure four different paths form which the control address register (CAR) receives the address.
  - The incrementer increments the content of the control register 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, that value is used when the micoprogram wishes to return from the subroutine.

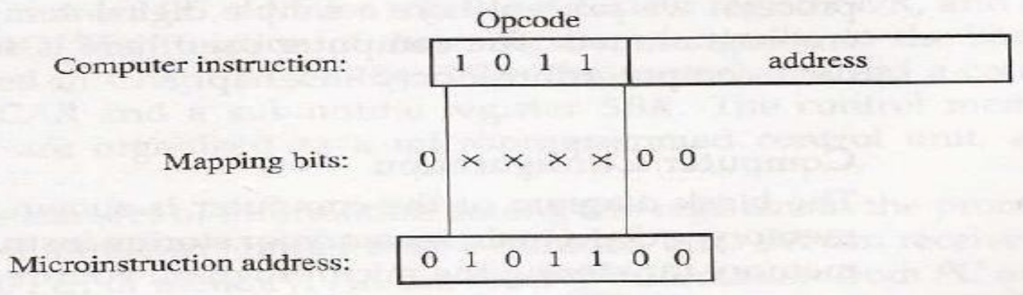
Conditional Branching:

- Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and i/o status conditions.
- The status bits, together with the field in the microinstruction that specifies a branch address, control the branch logic.
- The branch logic tests the condition, if met then branches, otherwise, increments the CAR.
- If there are 8 status bit conditions, then 3 bits in the microinstruction are used to specify the condition and provide the selection variables for the multiplexer.
- For unconditional branching, fix the value of one status bit to be one load the branch address from control memory into the CAR.

Mapping of Instruction:

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine is located.
- The status bits for this type of branch are the bits in the opcode.
- Assume an opcode of four bits and a control memory of 128 locations. The mapping process converts the 4-bit opcode to a 7-bit address for control memory shown in below figure.

Figure 7-3 Mapping from instruction code to microinstruction address.



- Mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.

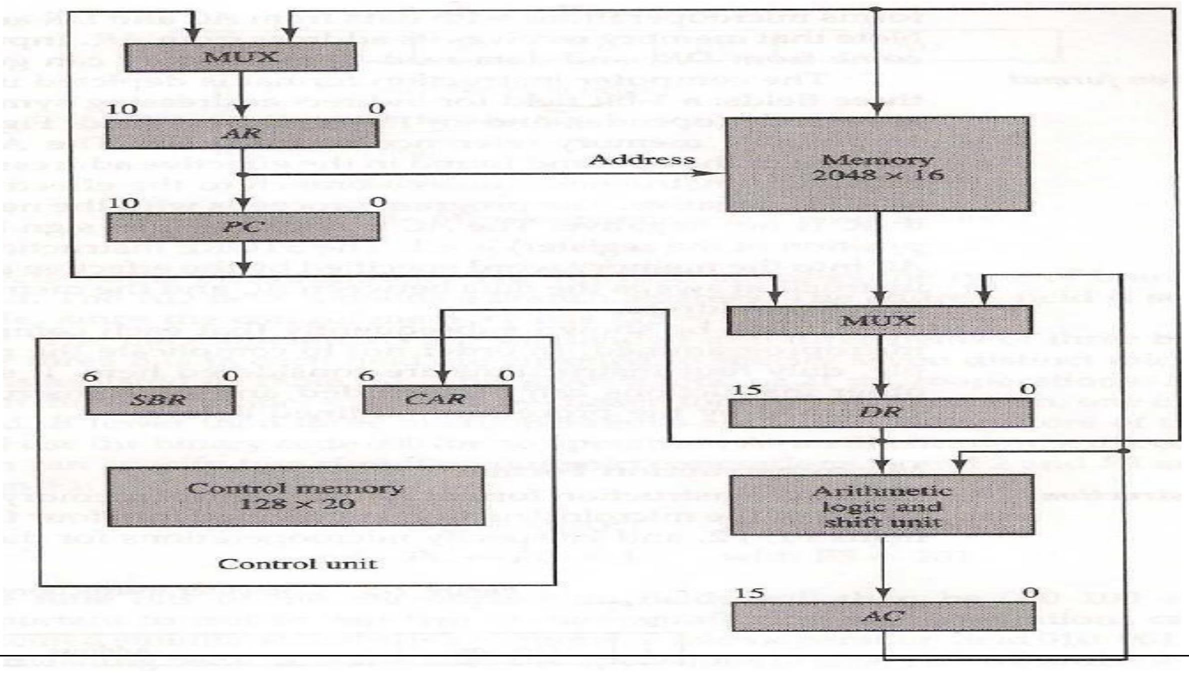
Subroutines:

- Subroutines are programs that are used by other routines to accomplish a particular task and can be called from any point within the main body of the microprogram.
- Frequently many microprograms contain identical section of code.
- Microinstructions can be saved by employing subroutines that use common sections of microcode.
- Microprograms that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return.
- A subroutine register is used as the source and destination for the addresses

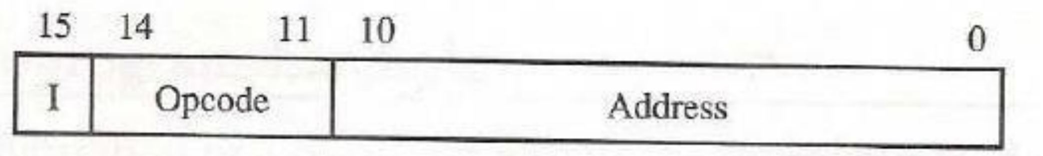
<u>3. Microprogram Example:</u>

- The process of code generation for the control memory is called *microprogramming*.
- The block diagram of the computer configuration is shown in below figure.
- Two memory units:
    - Main memory – stores instructions and data
    - Control memory – stores microprogram
- Four processor registers
    - Program counter – PC
    - Address register – AR
    - Data register – DR
    - Accumulator register - AC
- Two control unit registers
    - Control address register – CAR
    - Subroutine register – SBR

- Transfer of information among registers in the processor is through MUXs rather than a bus.



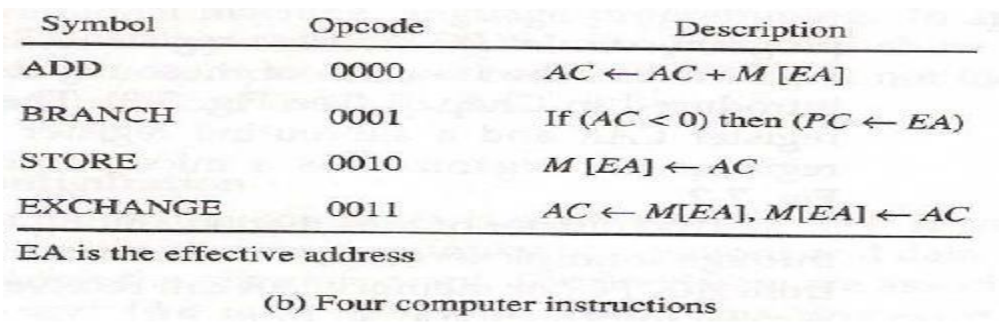
- The computer instruction format is shown in below figure.



(a) Instruction format

- Three fields for an instruction:
    - 1-bit field for indirect addressing
    - 4-bit opcode
    - 11-bit address field

- The example will only consider the following 4 of the possible 16 memory instructions

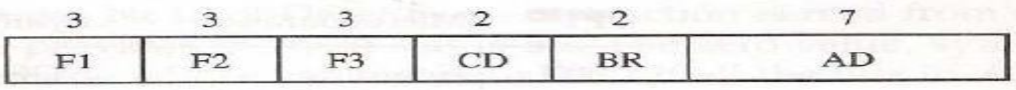


Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions

- The microinstruction format for the control memory is shown in below figure.



F1, F2, F3: Microoperation fields

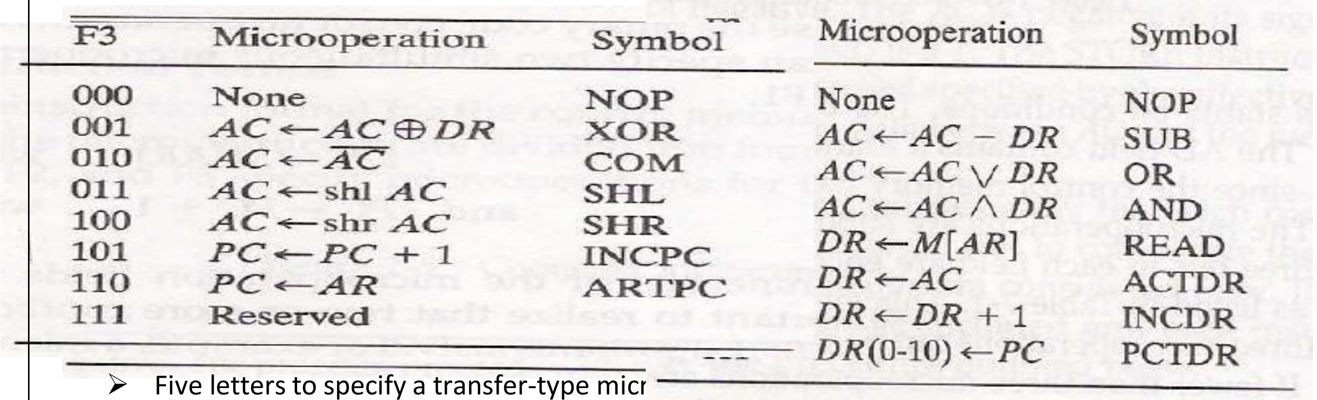
CD: Condition for branching

BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

- The microinstruction format is composed of 20 bits with four parts to it
    - Three fields F1, F2, and F3 specify microoperations for the computer [3 bits each]
    - The CD field selects status bit conditions [2 bits]
    - The BR field specifies the type of branch to be used [2 bits]
    - The AD field contains a branch address [7 bits]
- Each of the three microoperation fields can specify one of seven possibilities.
- No more than three microoperations can be chosen for a microinstruction.
- If fewer than three are needed, the code 000 = NOP.
- The three bits in each field are encoded to specify seven distinct microoperations listed in below table.



F3	Microoperation	Symbol	Microoperation	Symbol
000	None	NOP	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow \overline{AC}$	COM	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow$ shl AC	SHL	$AC \leftarrow AC \wedge DR$	AND
100	$AC \leftarrow$ shr AC	SHR	$DR \leftarrow M[AR]$	READ
101	$PC \leftarrow PC + 1$	INCPC	$DR \leftarrow AC$	ACTDR
110	$PC \leftarrow AR$	ARTPC	$DR \leftarrow DR + 1$	INCDR
111	Reserved		$DR(0\text{-}10) \leftarrow PC$	PCTDR

- Five letters to specify a transfer-type micr
    - First two designate the source register
    - Third is a 'T'

- Last two designate the destination register
  AC ← DR F1 = 100 = DRTAC

- The condition field (CD) is two bits to specify four status bit conditions shown below

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*

- The branch field (BR) consists of two bits and is used with the address field to choose the address of the next microinstruction.

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

- Each line of an assembly language microprogram defines a symbolic microinstruction and is divided into five parts
  1. The label field may be empty or it may specify a symbolic address. Terminate with a colon (:).
  2. The microoperations field consists of 1-3 symbols, separated by commas. Only one symbol from each field. If NOP, then translated to 9 zeros
  3. The condition field specifies one of the four conditions
  4. The branch field has one of the four branch symbols
  5. The address field has three formats
     a. A symbolic address – must also be a label
     b. The symbol NEXT to designate the next address in sequence
     c. Empty if the branch field is RET or MAP and is converted to 7 zeros
- The symbol ORG defines the first address of a microprogram routine.
- ORG 64 – places first microinstruction at control memory 1000000.

Fetch Routine:

- The control memory has 128 locations, each one is 20 bits.
- The first 64 locations are occupied by the routines for the 16 instructions, addresses 0-63.
- Can start the fetch routine at address 64.
- The fetch routine requires the following three microinstructions (locations 64-66).
- The microinstructions needed for fetch routine are:

$$AR \leftarrow PC$$

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

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

➢ It's Symbolic microprogram:

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

➢ It's Binary microprogram:

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

4. Design of control Unit:

➢ The control memory out of each subfield must be decoded to provide the distinct microoperations.
➢ The outputs of the decoders are connected to the appropriate inputs in the processor unit.
➢ The below figure shows the three decoders and some of the connections that must be made from their outputs.

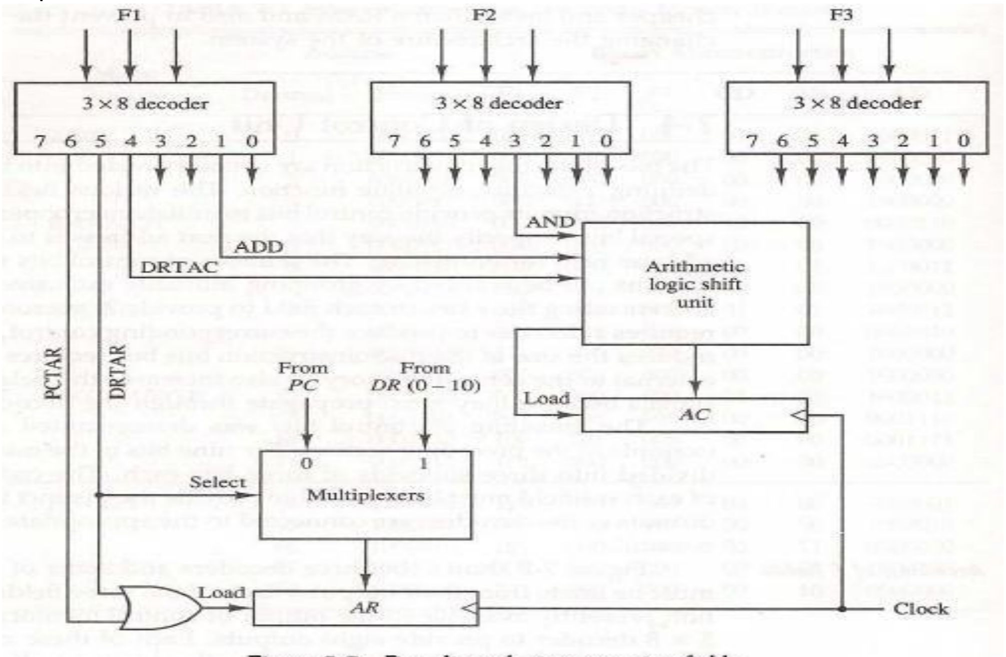


Figure 7-7 Decoding of microoperation fields.

➢ The three fields of the microinstruction in the output of control memory are decoded with a 3x8 decoder to provide eight outputs.
➢ Each of the output must be connected to proper circuit to initiate the corresponding microoperation as specified in previous topic.
➢ When F1 = 101 (binary 5), the next pulse transition transfers the content of DR (0-10) to AR.
➢ Similarly, when F1= 110 (binary 6) there is a transfer from PC to AR (symbolized by PCTAR). As shown in Fig, outputs 5 and 6 of decoder *F1* are connected to the load input of AR so that when either one of these outputs is active, information from the multiplexers is transferred to AR.
➢ The multiplexers select the information from DR when output 5 is active and from PC when output 5 is inactive.
➢ The transfer into AR occurs with a clock transition only when output 5 or output 6 of the decoder is active.
➢ For the arithmetic logic shift unit the control signals are instead of coming from the logical gates, now these inputs will now come from the outputs of AND, ADD and DRTAC respectively.

Microprogram Sequencer:

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection part is called a microprogram sequencer.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The next-address logic of the sequencer determines the specific address source to be loaded into the control address register.
- The block diagram of the microprogram sequencer is shown in below figure.
- 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 four sources and routes it into control address register *CAR.*
    - 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 content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register *SBR.*
- The other three inputs to multiplexer come from
    - The address field of the present microinstruction
    - From the out of SBR
    - From an external source that maps the instruction
- The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer.
- If the bit selected is equal to 1, the T variable is equal to 1; otherwise, it is equal to 0.
- The *T* value together with two bits from the BR (branch) field goes to an input logic circuit.
- The input logic in a particular sequencer will determine the type of operations that are available in the unit.
-

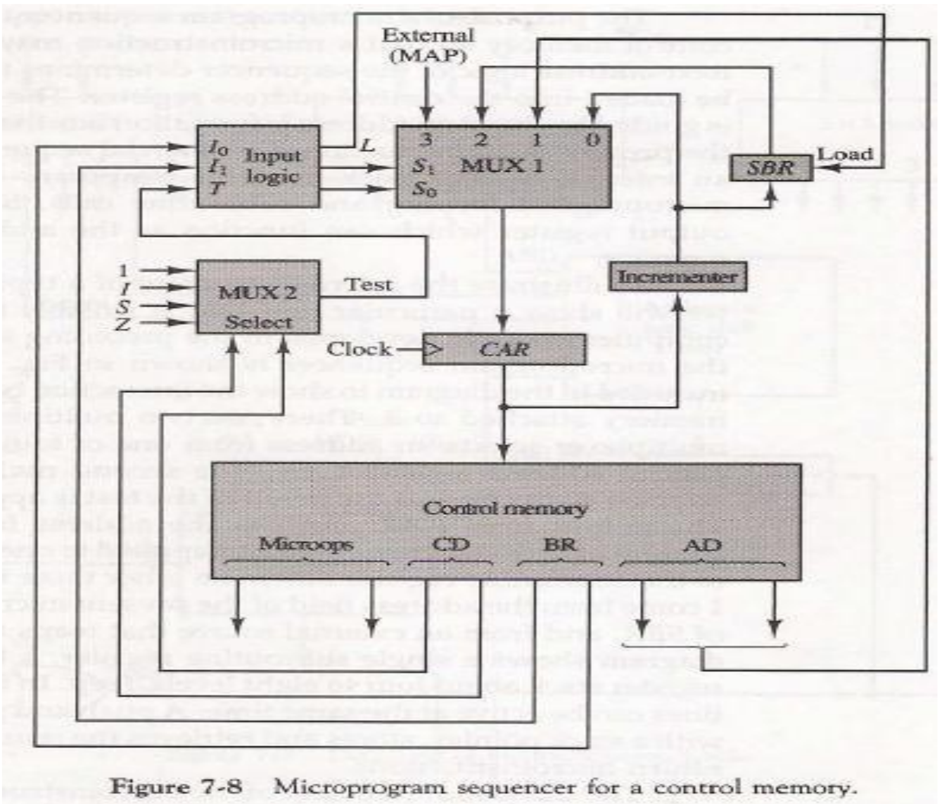
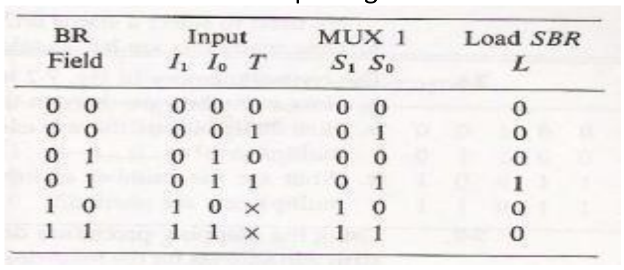


Figure 7-8 Microprogram sequencer for a control memory.

- The input logic circuit in above figure has three inputs I_0, I_1, and T, and three outputs, S_0, S_1, and L.
- Variables S_0 and S_1 select one of the source addresses for CAR. Variable L enables the load input in SBR.
- The binary values of the selection variables determine the path in the multiplexer.
- For example, with $S_1 S_0$ = 10, multiplexer input number 2 is selected and establishes transfer path from SBR toCAR.
- The truth table for the input logic circuit is shown in Table below.



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

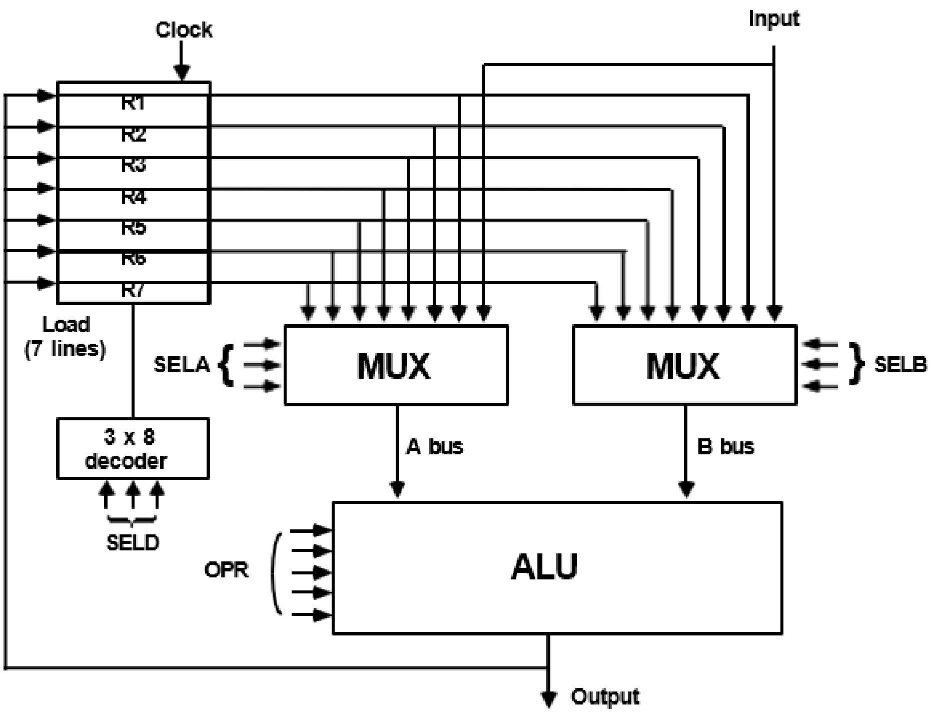
- Inputs I_1 and I_0 are identical to the bit values in the BR field.
- The bit values for S_1 and S_0 are determined from the stated function and the path in the multiplexer thatestablishes the required transfer.
- The subroutine register is loaded with the incremented value of CAR during a call microinstruction (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:

$$S_1=I_1$$

$$S_0=I_1I_0+I'_1T$$

$$L = I'_1T\,I_0$$

GENERAL REGISTER ORGANIZATION:



The control unit directs the information flow through ALU by:

- Selecting various *Components* in the system
- Selecting the *Function* of ALU

Example: R1 ß R2 + R3

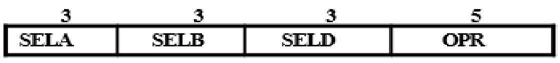
1 MUX A selector (SELA): BUS A ¬ R2

2 MUX B selector (SELB): BUS B ¬ R3

3 ALU operation selector (OPR): ALU to ADD

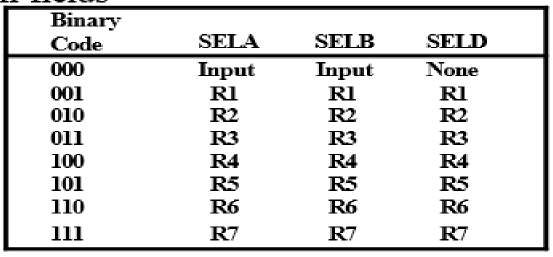
Decoder destination selector (SELD): R1 ¬ Out Bus

Control Word



3	3	3	5
SELA	SELB	SELD	OPR

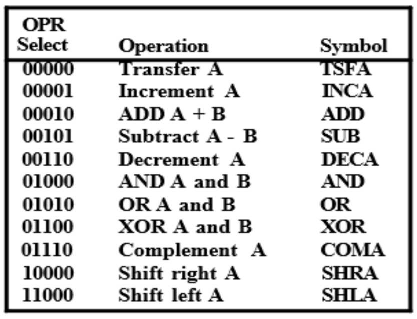
Encoding of register selection fields



Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

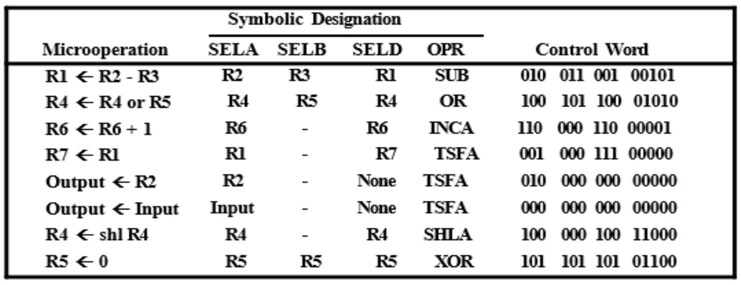
ALU CONTROL

Encoding of ALU operations

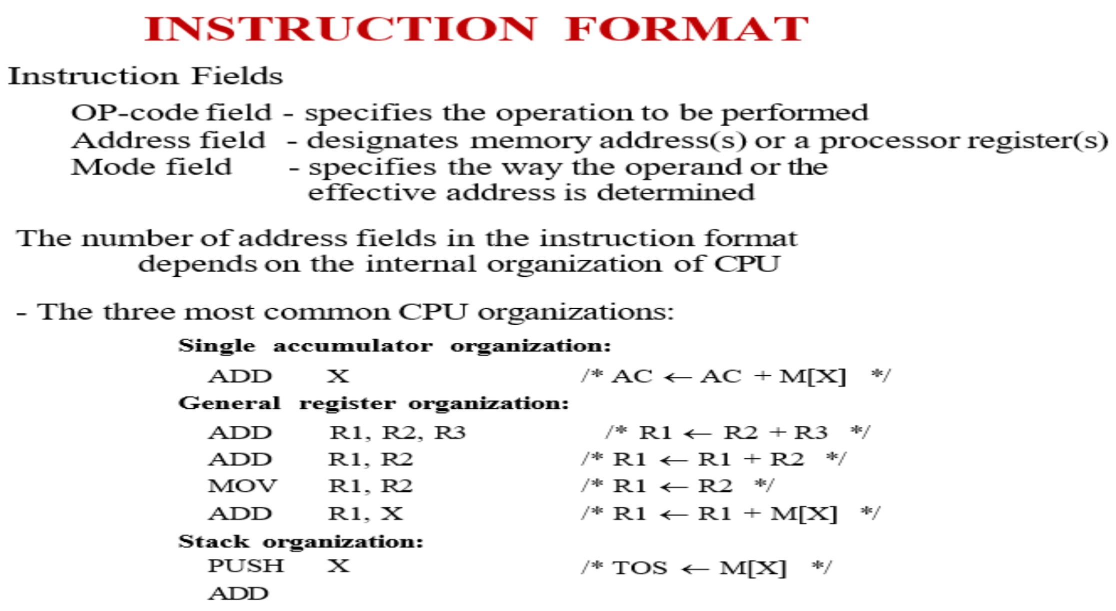


OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Examples of ALU Microoperations



	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
R1 $\leftarrow$ R2 - R3	R2	R3	R1	SUB	010 011 001 00101
R4 $\leftarrow$ R4 or R5	R4	R5	R4	OR	100 101 100 01010
R6 $\leftarrow$ R6 + 1	R6	-	R6	INCA	110 000 110 00001
R7 $\leftarrow$ R1	R1	-	R7	TSFA	001 000 111 00000
Output $\leftarrow$ R2	R2	-	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	-	None	TSFA	000 000 000 00000
R4 $\leftarrow$ shl R4	R4	-	R4	SHLA	100 000 100 11000
R5 $\leftarrow$ 0	R5	R5	R5	XOR	101 101 101 01100



INSTRUCTION FORMAT

Instruction Fields

OP-code field - specifies the operation to be performed
Address field - designates memory address(s) or a processor register(s)
Mode field - specifies the way the operand or the effective address is determined

The number of address fields in the instruction format depends on the internal organization of CPU

- The three most common CPU organizations:

Single accumulator organization:

```
ADD    X             /* AC ← AC + M[X]  */
```

General register organization:

```
ADD    R1, R2, R3     /* R1 ← R2 + R3  */
ADD    R1, R2        /* R1 ← R1 + R2  */
MOV    R1, R2        /* R1 ← R2  */
ADD    R1, X         /* R1 ← R1 + M[X]  */
```

Stack organization:

```
PUSH   X             /* TOS ← M[X]  */
ADD
```

Addressing Modes:

- * Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced)
- * Variety of addressing modes
  - to give programming flexibility to the user
  - to use the bits in the address field of the instruction efficiently

Types of Addressing Modes:

Implied Mode

Address of the operands are specified implicitly in the definition of the instruction

- No need to specify address in the instruction
- EA = AC, or EA = Stack [SP**],** **EA: Effective Address.**

Immediate Mode

Instead of specifying the address of the operand, operand itself is specified

- No need to specify address in the instruction
- However, operand itself needs to be specified
- Sometimes, require more bits than the address
- Fast to acquire an operand

Register Mode

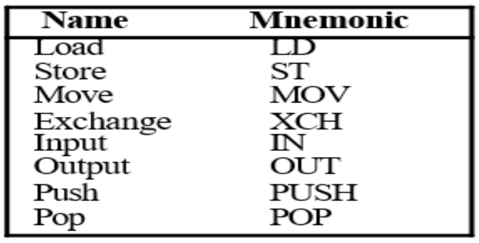
Address specified in the instruction is the register address

- Designated operand needs to be in a register
- Shorter address than the memory address
- Saving address field in the instruction
- Faster to acquire an operand than the memory addressing
- EA = IR(R) (IR(R): Register field of IR)
- **Register Indirect Mode**
- Instruction specifies a register which contains the memory address of the operand
- Saving instruction bits since register address
- is shorter than the memory address
- Slower to acquire an operand than both the register addressing or memory addressing
- EA = [IR(R)] ([x]: Content of x)
- **Auto-increment or Auto-decrement features:**
- Same as the Register Indirect, but:
- When the address in the register is used to access memory, the
- value in the register is incremented or decremented by 1 (after or before
- the execution of the instruction)
- **Direct Address Mode**
- Instruction specifies the memory address which can be used directly to the physical memory
- Faster than the other memory addressing modes
- Too many bits are needed to specify the address for a large physical memory space
- EA = IR(address), (IR(address): address field of IR)
- **Indirect Addressing Mode**
- The address field of an instruction specifies the address of a memory location that contains the address of the operand
- When the abbreviated address is used, large physical memory can be
- addressed with a relatively small number of bits
- Slow to acquire an operand because of an additional memory
- access
- EA = M[IR(address)]
- **Relative Addressing Modes**
- The Address fields of an instruction specifies the part of the address (abbreviated address) which can be used along with a

- designated register to calculate the address of the operand
- ***PC Relative Addressing Mode(R = PC)***
- EA = PC + IR(address)
- Address field of the instruction is short
- Large physical memory can be accessed with a small number of address bits
- ***Indexed Addressing Mode***
- XR: Index Register:
- EA = XR + IR(address)
- ***Base Register Addressing Mode***
- BAR: Base Address Register:
- EA = BAR + IR(address

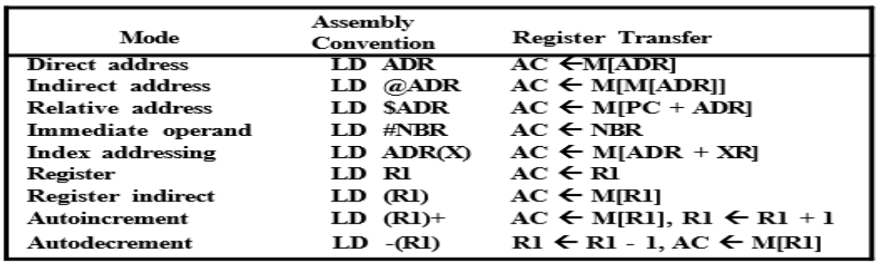
DATA TRANSFER INSTRUCTIONS:

Typical Data Transfer Instructions:



Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Transfer Instructions with Different Addressing Modes



Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	AC ←M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Autoincrement	LD (R1)+	AC ← M[R1], R1 ← R1 + 1
Autodecrement	LD -(R1)	R1 ← R1 - 1, AC ← M[R1]

DATA MANIPULATION INSTRUCTIONS

Three Basic Types:
Arithmetic instructions
Logical and bit manipulation instructions
Shift instructions

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negate(2's Complement)	NEG

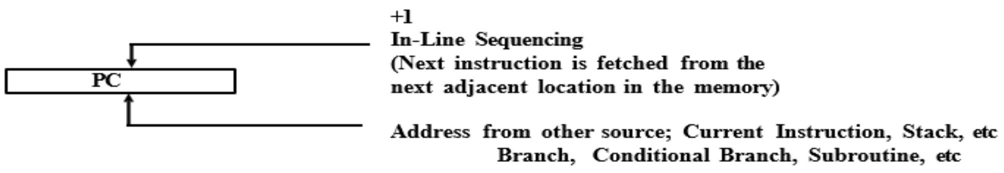
Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right thru carry	RORC
Rotate left thru carry	ROLC

PROGRAM CONTROL INSTRUCTIONS

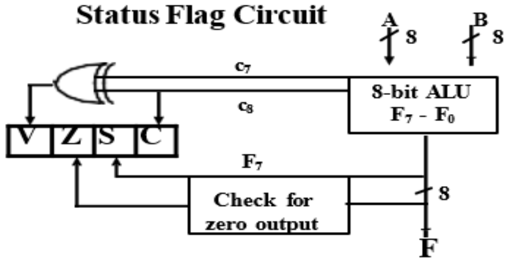

PC
+1
In-Line Sequencing
(Next instruction is fetched from the next adjacent location in the memory)
Address from other source; Current Instruction, Stack, etc
Branch, Conditional Branch, Subroutine, etc

Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by -)	CMP
Test (by AND)	TST

* CMP and TST instructions do not retain their results of operations(- and AND, respectively). They only set or clear certain Flags.

Status Flag Circuit



CONDITIONAL BRANCH INSTRUCTIONS

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
	Unsigned compare conditions (A - B)	
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
	Signed compare conditions (A - B)	
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$