Computer Architecture ECS-355

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

(Week 8) Lecture 15-16

Learning objectives:

- Review of the last lecture
- Microinstruction format
- Fetch Routine
- Symbolic microprogram

(Note: It is important to note that we are considering "Basic computer architecture" processor which we were discussing in the class and it's the continuation of the same concepts. Another important point is that brief part of the last lecture is repeated. Audio clip will be sent separately for every diagram.)

Resources: Beside these lecture handouts, this lesson will draw from the following

Text Book: Computer System Architecture by Morris Mano (3rd Edition) and **Reference book:Computer Architecture, by William Stallings (4th Edition)**.

Lecture:

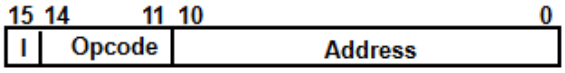
Microinstruction format:

This section introduces microinstruction of the control unit micro program. For the reference User program instruction 'Machine instruction' format is also presented which is different from the Microinstruction.

The microinstruction format for the control memory is shown in Figure below. The 20 bits of the microinstruction are divided into four functional parts. The three fields F1, F2, and F3 specify microoperations for the computer. The CD field selects status bit conditions. The BR field specifies the type of branch to be used. The AD field contains a branch address. The address field is seven bits wide, since the control memory has $128 = 2^7$ words. The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations as listed in Table 7-1. 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.

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

Machine instruction format

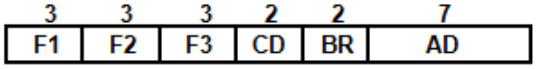


Sample machine instructions

Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

Microinstruction Format



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

The nine bits of the microoperation fields will then be 000 100 101. It is important to realize that two or more conflicting microoperations cannot be specified simultaneously. For example, a microoperation field 010 001 000 has no meaning because it specifies the operations to clear AC to 0 and subtract DR from AC at the same time.

Microinstructions fields:

Each microoperation in Table presented belows is defined with a register transfer statement and is assigned a symbol for use in a symbolic microprogram. All transfer-type microoperations symbols use five letters. The first two letters designate the source register, the third letter is always a T, and the last two letters designate the destination register. For example, the microoperation that specifies the transfer AC <- DR (F1 = 100) has the symbol DRTAC, which stands for a transfer from DR to AC. The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table below.

Microinstruction Field Descriptions

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

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	

Condition and Branching fields:

The first condition is always a 1, so that a reference to CD = 00 (or the symbol U) will always find the condition to be true. When this condition is used in conjunction with the BR (branch) field, it provides an unconditional branch operation. The indirect bit I is available from bit 15 of DR after an instruction is read from memory. The sign bit of AC provides the next status bit. The zero value, symbolized by Z, is a binary variable whose value is equal to 1 if all the bits in AC are equal to zero. We will use the symbols U, I, S, and Z for the four status bits when we write microprograms in symbolic form. The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction. As shown in Table, when BR = 00, the control performs a jump GMP) operation (which is similar to a branch), and when BR = 01, it performs a call to subroutine (CALL) operation. The two operations are identical except that a call microinstruction stores the return address in the subroutine register SBR. The jump and call operations depend on the value of the CD field. If the status bit condition specified in the CD field is equal to 1, the next address in the AD field is transferred to the control address register CAR_ Otherwise, CAR is incremented by 1. The return

from subroutine is accomplished with a BR field equal to 10. This causes the transfer of the return address from SBR to CAR. The mapping from the operation code bits of the instruction to an address for CAR is accomplished when the BR field is equal to 11. This mapping is as depicted in Figure below. 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.

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

Fetch Routine:

The control memory has 128 words, and each word contains 20 bits. To microprogram the control memory, it is necessary to determine the bit values of each of the 128 words. The first 64 words (addresses 0 to 63) are to be occupied by the routines for the 16 instructions. The last 64 words may be used for any other purpose. A convenient starting location for the fetch routine is address 64. The microinstructions needed for the fetch routine are

AR <-PC

DR <- M[AR], PC<-PC + 1

AR <- DR(0-10), CAR(2-5) <- DR(11-14), CAR(0,1,6) <-0

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

Sequence of microoperations in the fetch cycle:

```
AR ← PC
DR ← M[AR], PC ← PC + 1
AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 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 three microinstructions that constitute the fetch routine have been 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.

Symbolic Microprogram:

When the third MAP microinstruction in the fetch routine is executed, it branches to address 0xxxx00, where xxx represents the four bits of the operation code. For example, if the operation code for an ADD instruction 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 initial addresses for the BRANCH and STORE routines are 000100 (decimal 4) and 000200 (decimal 8), respectively, with the first addresses for the other 13 routines being at 12, 16, 20, ...60, providing four words in control memory for each routine.

Each routine must include microinstructions for evaluating the effective address and executing the instruction. The indirect address mode, relevant to all memory-reference instructions, can save control memory space if the microinstructions for indirect addressing are stored as a subroutine called INDRCT, located right after the fetch routine (as shown in Table 7-2). This table also displays the symbolic microprogram for the fetch routine and the microinstruction routines for executing four computer instructions.

To illustrate how transfer and return from the indirect subroutine work, assume the MAP microinstruction at the end of the fetch routine causes a branch to address 0, where the ADD routine is stored. The first microinstruction in the ADD routine calls the INDRCT subroutine, based on status bit I. If I = 1, it branches to INDRCT and saves the return address (T) in the subroutine register SBR. The INDRCT subroutine has two microinstructions:

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

TABLE 7-2 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	

In indirect addressing, the address part of the instruction indicates where the effective address is stored, rather than being the operand's address. Thus, the memory must be accessed to retrieve the effective address, which is then transferred to AR. The return from the subroutine (RET) transfers the address from SBR to CAR, resuming execution at the second microinstruction of the ADD routine.

The ADD instruction execution occurs through microinstructions at addresses 1 and 2. The first microinstruction reads the operand from memory into DR, while the second adds the contents of DR and AC and then jumps back to the beginning of the fetch routine.

The BRANCH instruction should direct control to the effective address if AC < 0. This condition is detected when status bit 5 is 1, indicating a negative value. The BRANCH routine first checks the value of S; if S = 0, no branch occurs, and the next microinstruction jumps back to the fetch routine without changing the PC. If S = 1, control transfers to location 0\VER, where the microinstruction calls the INDRCT subroutine if I = 1. The effective address is then moved from AR to PC, and the program jumps back to the fetch routine.

The STORE routine also utilizes the INDRCT subroutine if I = 1. Here, the content of AC is transferred to DR, initiating a memory write operation to store DR's content at the effective address specified in AR.

In the EXCHANGE routine, the operand from the effective address is read into DR, and the contents of DR and AC are swapped in the third microinstruction. This interchange is feasible when the registers are of the edge-triggered type (see Fig. 1-23). Finally, the original content of AC, now in DR, is stored back in memory.

Note that Table 7-2 shows only a partial list of the microprogram, covering four out of 16 possible computer instructions. Additionally, control memory words from locations 69 to 127 remain unused. Instructions like multiply and divide, which require longer sequences of microoperations, will need more than four microinstructions for execution. These unused control memory words can accommodate such requirements.