Computer Architecture ECS-355

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

(Week 9) Lecture 17-18

Learning objectives:

- Review of the last lecture
- Binary Microprogram
- Design of control unit
- Microprogram sequencer

(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:

Binary Microprogram

The symbolic microprogram is a convenient form for writing microprograms in a way that people can read and understand. But this is not the way that the microprogram is stored in memory. The symbolic microprogram must be translated to binary either by means of an assembler program or by the user if the microprogram is simple enough as in this example.

The equivalent binary form of the microprogram is listed in Table 7-3 (Book). The addresses for control memory are given in both decimal and binary. The binary content of each microinstruction is derived from the symbols and their equivalent binary values as defined in Table 7-1 (Book).

Note that address 3 has no equivalent in the symbolic microprogram since the ADD routine has only three microinstructions at addresses 0, 1, and 2. The next routine starts at address 4. Even though address 3 is not used, some binary value must be specified for each word in control

memory. We could have specified all 0' s in the word since this location will never be used. However, if some unforeseen error occurs, or if a noise signal sets CAR to the value of 3, it will be wise to jump to address 64, which is the beginning of the fetch routine.

The binary microprogram listed in Table 7-3 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. This fabrication is a hardware process and consists of creating a mask for the ROM so as to produce the l's and D's for each word. The bits of ROM are fixed once the internal links are fused during the hardware production. The ROM is made of IC packages that can be removed if necessary and replaced by other packages. To modify the instruction set of the computer, it is necessary to generate a new microprogram and mask a new ROM. The old one can be removed and the new one inserted in its place.

If a writable control memory is employed, the ROM is replaced by a RAM. The advantage of employing a RAM for the control memory is that the microprogram can be altered simply by writing a new pattern of l'.s and D's without resorting to hardware procedures. A writable control memory possesses the flexibility of choosing the instruction set of a computer dynamically by changing the microprogram under processor control. However, most microprogrammed systems use a ROM for the control memory because it is cheaper and faster than a RAM and also to prevent the occasional user from changing the architecture of the system.

Example:

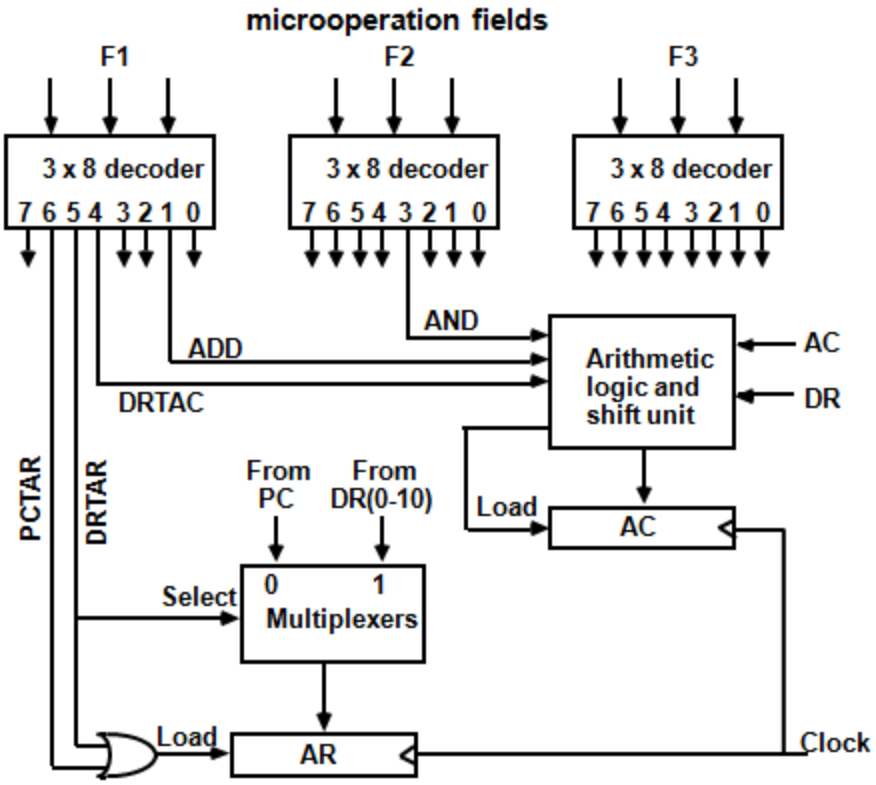
This example will explain the addition process of a variable which is stored in memory (indirect addressing mode). We need to analyze and understand how Control Unit will utilize the micro program for the said problem.

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

Design of Control Unit

The bits of the microinstruction are usually divided into fields, with each field defining a distinct, separate function. The various fields encountered in instruction formats provide control bits to initiate microoperations in the system, special bits to specify the way that the next address is to be evaluated, and an address field for branching. 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. This method reduces the size of the microinstruction bits but requires additional hardware external to the control memory. It also increases the delay time of the control signals because they must propagate through the decoding circuits. The

encoding of control bits was demonstrated in the programming example of the preceding section. The nine bits of the microoperation field are divided into three subfields of three bits each. The control memory output 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. Figure below shows the three decoders and some of the connections that must be made from their outputs. Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3 x 8 decoder to provide eight outputs. Each of these outputs must be connected to the proper circuit to initiate the corresponding microoperation as specified in Table 7-1. For example, when F1 = 101 (binary 5), the next clock pulse transition transfers the content of DR(0-10) to AR (symbolized by DRTAR in Table 7-1). Similarly, when F1 = 110 (binary 6) there is a transfer from PC toAR (symbolized by PCTAR). As shown in Fig. 7-7, 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 pulse transition only when output 5 or output 6 of the decoder are active. The other outputs of the decoders that initiate transfers between registers must be connected in a similar fashion.



The arithmetic logic shift unit can be designed as in Figs. 5-19 (chapter 5) and 5-20 (chapter 5). Instead of using gates to generate the control signals marked by the symbols AND, ADD, and DR in Fig. 5-19 (chapter 5), these inputs will now come from the outputs of the decoders associated with the symbols AND, ADD, and DRTAC, respectively , as shown in Figure above. The other outputs of the decoders that are associated with an AC operation must also be connected to the arithmetic logic shift unit in a similar fashion.

Micro 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. A microprogram sequencer can be constructed with digital functions to suit a particular application. However, just as there are large ROM units available in integrated circuit packages, so are general-purpose sequencers suited for the construction of microprogram control units. To guarantee a wide range of acceptability, an integrated circuit sequencer must provide an internal organization that can be adapted to a wide range of applications.

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

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 choice of the address source is guided by the next-address information bits that the sequencer receives from the present microinstruction. Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during microprogram looping and subroutine calls. Some sequencers provide an output register which can function as the address register for the control memory. To illustrate the internal structure of a typical microprogram sequencer we will show a particular unit that is suitable for use in the microprogram computer example developed in the preceding section. The block diagram of the microprogram sequencer is shown in figure below. 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 a 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 number 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction. Although the diagram shows a single subroutine register, a typical sequencer will have a register stack about four to eight levels deep. In this way, a number of subroutines can be active at the same time. A push and pop operation, in conjunction with a stack pointer, stores and retrieves the return address during the call and return microinstructions. 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 (test) variable is equal to 1; otherwise, it is equal to 0. The T value together with the 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. Typical sequencer operations are: increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations. With three inputs, the sequencer can provide up to eight address

sequencing operations. Some commercial sequencers have three or four inputs in addition to the T input and thus provide a wider range of operations.

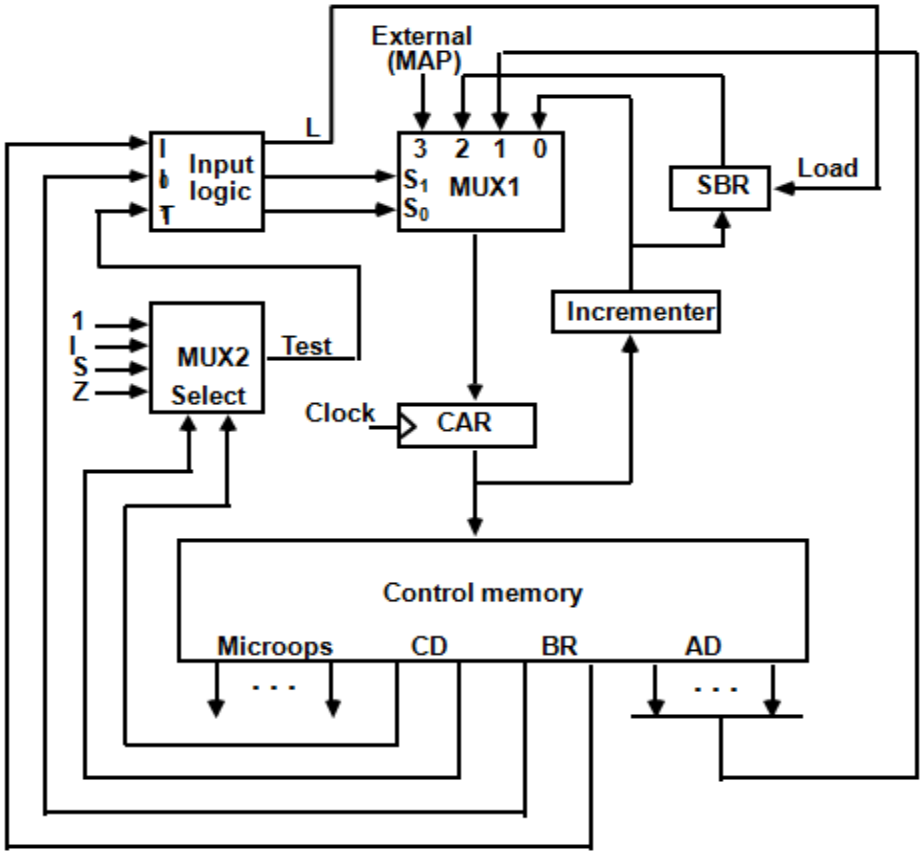


Figure: Microprogram sequencer

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

The input logic circuit in figure below has three inputs, I_0, I_1, and T, and three outputs, S0, S1, and L. Variables So and S, select one of the source addresses for CAR. Variable L enables the load input in SBR. The binary values of the two selection variables determine the path in the multiplexer. For example, with S1 S0 = 10, multiplexer input number 2 is selected and establishes a transferpath from SBR to CAR. Note that each of the four inputs as well as the output of MUX 1 contains a 7-bit address. The truth table for the input logic circuit is shown in Table 7-4. Inputs 11 and 10 are Identical to the bit values in the BR field. The function listed in each entry was defined in Table 7·1. The bit values for S1 and So 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:

$$S_1 = I_1$$

$$S_0 = I_1.I_0 + I_1' T$$

$$L = I_1'I_0T$$

TABLE 7-4 Input Logic Truth Table for Microprogram Sequencer

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

The circuit can be constructed with three AND gates, an OR gate, and an inverter. Note that the incrementer circuit in the sequencer of figure (microprogram 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 circuits. The output carry from one stage must be used 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.