BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2103

COMPUTER ORGANIZATION

Unit 13

Microprogramming

Table of Contents

1. INTRODUCTION

In the earlier units, you learned about the basic concepts of processors and how the computer executes the given command with the help of the instruction cycle. You also studied that the instructions are executed in a complex series of steps using many microprocessor components and various other components as well. Also in the previous unit, you learned about multiprocessing, different types of multiprocessors, and contention problems in the multiprocessor systems. The brain of the computer system is the central processing unit. CPU is made up of three main components i.e. ALU (Arithmetic Logic Unit), CU (Control Unit), and Registers. The CU or the control unit controls the entire operation of the computer. The Control unit decides the course of action when it receives an instruction. This control unit facilitates microprogramming. Microprogramming is the process of writing microcode for a microprocessor. Microcode is a layer of hardware-level instruction that translates the machine instructions into sequences of circuit-level operations. Microcode in a processor implementation is called a micro- program. The idea of microprogramming was formulated by Maurice Vincent Wilkes of Cambridge University in the 1950s. But the computer memories at that time were too slow to implement practically. In 1964, IBM first introduced its IBM 360 series of computers which were practically based on microprogramming.

In this unit, we will describe the concept of microprogramming and how a microprogrammed CU functions, microinstructions, microcode, and machine instructions.

Objectives:

After studying this unit, you should be able to:

- *Explain the concept of microprogramming*
- *State the role of the computer clock*
- *Describe the sequencing and execution of microinstructions*
- *Recognize the control path in microprogramming*
- *Explain the use of microcode*
- *List and explain various types of machine instructions*

2. BASIC PRINCIPLES

Let us now introduce the basic principles of microprogramming architecture. In a conventional computer i.e. in a hardwired computer, the control unit fetches the next instruction, decodes it, and then executes it. The control unit incorporates an execution circuit for each machine instruction. The decoding technique of the CU determines which circuit to utilize for the current instruction. Thus, the instructions are executed by hardware circuits, which is the fastest way the computer can run. Microprogramming is a completely different approach to fetching and executing instructions.

Microprogramming is an organized method of designing the control unit (CU) of a computer. In microprogramming, the control unit fetches and executes microinstructions instead of machine instructions. Micro-instructions are simple and function at a lower level i.e. they are close to the hardware as compared to the machine instructions. The complete set of steps required to process a machine instruction is called the microprogram.

2.1. Features of microprogramming

The main features of microprogramming are as follows:

i) It simplifies the hardware as the computer hardware components are not much expensive.
ii) It permits an orderly approach to control design. The micro-program is easy to debug and maintain.
iii) It makes the instruction set flexible and also the design of the control unit.
iv) It makes it possible to postpone detailed design decisions. New features can be added easily.
v) It is a slow process and acts as a disadvantage for the user when time available is limited.

The underlying concept in microprogramming is to have all the information on instruction execution stored in a special memory space termed the control store. This memory is not a part of the main memory of the computer but is a separate entity. The major difference between the control store and the main memory is given in table 1.

Table 1: Control Memory vs. Main Memory

Control Memory	Main Memory
1. It is much smaller in size. It requires space only for micro-programs or a few instructions.	1. It is large in size. It must have space tostore programs.
2. Control memory is read-only innature because it consists of microprograms that are writtenand stored in it permanently.	2. On the other hand, different types of programs are loaded into the main memory all the time.
3. It must be faster, as to run a machine instruction, several microinstructions have to be fetched and executed.	3. It need not necessarily be faster.

2.2. Hardwired vs. micro-programmed computers

Most of the computers used today are micro-programmed. The underlying reason behind this is the flexibility offered by the microprogramming. The hardwired CU is not as flexible as a micro-programmed CU. It is practically impossible to modify the instruction set and basic architecture of hardwired CU. But this could be done easily in a micro-programmed system. All one needs to do is to alter the micro-program stored in its control memory.

The control memory stores a microprogram corresponding to each opcode of conventional instruction. Figure 13.1 shows the Layout of a Micro- programmed Unit.

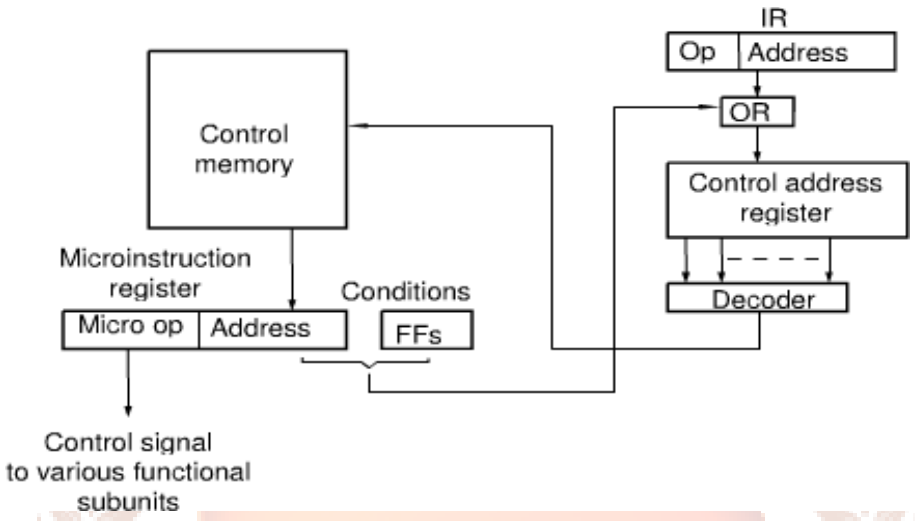


Figure 1: Layout of a Micro-programmed Unit

In figure 1 you can see that control memory stores a microprogram corresponding to each op-code of conventional instruction. Microinstructions are retrieved from control memory. The control address register locates the microinstruction to be retrieved from the control memory. The microinstruction register holds the retrieved microinstruction - micro opcode and address of the next microinstruction in the control memory. Current microinstruction is executed. The next microinstruction's address is entered into the control memory to retrieve the next microinstruction. If all microinstructions were executed, then store the next op-code of conventional instruction in the control address register, if not, executes the remaining microinstruction.

Conditional jumps are implemented by letting the states of some conditional flip-flops modify the next microinstruction's address which has to be retrieved. For example, you have a four-bit opcode CU that allows 16 instructions and you may have five additional instructions inserted into its instruction set. This could be done easily by modifying the micro-program in micro-programmed architecture. But doing this similar task in a hardwired edition of your computer would need an entire remodel of the controller circuit hardware. Still, one more main benefit of utilizing micro-programmed control is the reality that it leads to simplification of the task of designing the computer. With the evolution of microprogramming, the process of identifying the instruction set and architecture is now

assigned to software rather than hardware. However, hardwired computers are still being used for certain types of applications, particularly where speed is a major consideration.

2.3 Applications and advantages of microprogramming

The various microprogramming applications are given below:

- Microprogramming is used for providing backward compatibility. Due to commercial reasons, new processors frequently are required to have the ability to run software that used to run on its predecessors.
- This technique has been utilized in minicomputers, mainframes, and also microprocessors.
- Microprogramming is used for developing special purpose machines that are planned for particular applications, such as real-time image analysis, radar processing, etc.
- In order to convert the selection of chips into a processor equipped with usable instructions, a micro-program controller can be utilized. This can be further used in the development of software for a fresh architecture prior to the production of the chip.

Intel 8080, Motorola 68000 and CISC (Complex Instruction set computers) CPUs are some examples of CPUs with micro-programmed CU.

2.4 Advantages

One main benefit of a micro-programmed control unit is that the content of the micro-program memory (also called the control store) can be altered and hence you can design your personal machine-level instructions. Moreover, it is absolutely possible to select a microprogram set which will execute the machine code of a completely different computer. In such a case, it is said that the computer has emulated one more computer.

This facility is helpful when you are converting your old computer into a new one that possesses a machine code which is mismatched with your old programs. Emulation can be applied to programs that occur in binary or object form on a disk or tape. As a result of writing micro-programs on the new machine in order to decode the machine code of the old machine, you have the advantage of utilizing the old software and yet enjoying the benefits of the new machine.

Emulation is the mutual software/hardware interpretation of the machine instruction of one machine by another. Target's machine architecture is mapped onto the host machine. Microprogramming supports effortless programming. Dynamic Microprogramming allows routines to be easily micro-programmed. In this, the computer is limited to represent any instruction vocabulary by the use of Writable Control Memory (WCM). It allows the instruction set of the machine to be changed and tailored to specific applications.

2.5 Limitations of microprogramming

The evolutions of powerful interpreters and compilers which are capable of creating low-level codes from high-level commands have superseded the advantages of micro-programming.CU based on microprogramming is slow. Also, there are many CPU design schemes that don't use micro- programming such as Transport Triggered Architecture (TTA) Processors, Superscalar Processors, and Reduced instruction set computing RISC Processors.

Self-Assessment Questions – 1

1. _____applies to programs which occur in binary or objectform on disk or tape.
2. CISC stands for_____.
3. In 1957, Moore proposed an alternative of CU design called as 'microprogramming'. (True/False)

Activity 1

There is an interesting question for you. Who was the first person to use the term microprogramming? When was the first micro-programmed computer tested and where? You can take the help of internet.

3. COMPUTER CLOCK

A computer clock defines the speed or the clock rate that characterizes the pace at which a microprocessor executes instructions and synchronizes the various computer tasks. The clock depicts a customary pattern of alternating low and high voltages on a wire. The various parts of a computer contain data and instructions. At regular intervals, they send this data through wires to the adjoining processing station. The computer has a clock pulse which coordinates with this activity. Computer clock helps in timing, synchronization, regulation of operation of processors, and generation of interrupts.

Clock pulses are constant, accurately placed changes in voltage. When to begin sending data on the wires is communicated to some circuits by the clock pulse. At the same time, it also tells other circuits about the time when the data from the earlier pulse should have already turned up. This clock pulse is distributed to the entire components of a processor which evokes them to perform their next operation. The clock speeds are expressed in Megahertz (MHz) and Gigahertz (GHz). Figure 13.2 shown below depicts a clock with four output lines, all carrying the same signal, but at different phases.

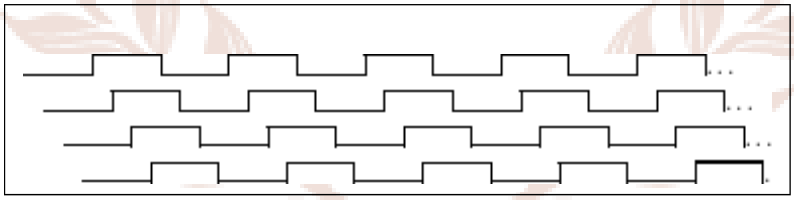


Figure 2: Computer Clock Cycle

These outputs segregate each clock cycle into four sub-cycles. These clock cycles may not be necessarily of the same size, and the four rising edges can be used to trigger the processor to start any four operations (fetch, decode, execute, output) in each clock cycle. It is worth noticing that the three of the clock outputs are generated as delayed copies of the first one, using delay lines.

Self-Assessment Questions – 2

4. The clock is an oscillator that produces an output signal shaped like a wave.
5. The response time of a simple digital device is measured in microseconds. (True/False)

4. MICROINSTRUCTIONS AND ITS TIMING

Let us now understand what exactly a microinstruction is. A microinstruction is an instruction that controls data flow and instruction-execution sequencing in a processor at a more fundamental level than machine instructions. These microinstructions form the fetch-execute cycle. A microinstruction is a simple command that makes the hardware component work accurately. Each computer has a unique format of microinstruction. A microprogram-controlled computer has a control unit which is mainly a computer inside a computer. In simple terms, you may conclude that to execute a particular machine instruction, you will need a series of microinstructions. Figure 3 shows the basic data path in a micro-program.

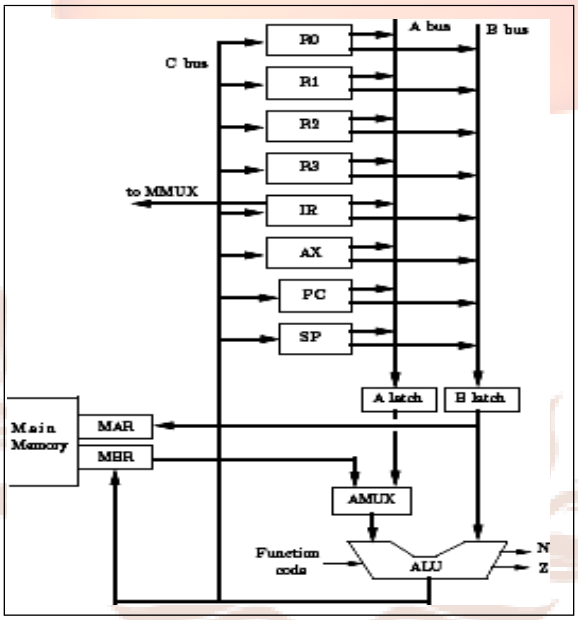


Figure 3: The Data Path in a Micro-program.

This is a very simple diagram which shows the various registers i.e. MAR (Memory Address Register), R0-R3,), IR (Instruction register), Accumulator (AX), PC (Program Counter), SP (Stack Pointer), and MBR (Memory Buffer Register). You can also see the three buses A, B, and C used in the data path. A and B are the two input buses and C is the output bus. In addition to these components, there are also many gates, decoders, control lines, and other hardware components which are not used in the execution of a micro- program.

Let us have a look at the operations carried out by micro-programmed computers shown in figure 3:

1) The set of buses A and B send data to the respective latches.
2) A 3-bit code can be sent to the ALU, requesting an ALU operation. Additionally, the data from the B latch can be sent to the MAR to load the MAR with an address and can also be sent to the ALU.
3) The ALU output can be sent to the C bus, to the MBR, to both places, or to none of them.
4) A memory operation (read or write) can be specified. The RD and WR fields can be set to 1 by any microinstruction that needs to read memory or write to it. Since memory can only do one thing at a time, it is invalid to set both RD and WR to 1 in the same microinstruction.

   Remember that our system is micro-programmed; therefore the control unit will not have the circuits to carry out the machine instructions. Instead, they will be executed by the microinstructions. Hence, the microinstruction should therefore be able to specify any of the four operations listed above. As you may recall that microinstructions are analogous to machine instructions but are simpler, hence several microinstructions (or a micro-program) are required to execute one machine instruction. Importantly the microinstructions in a micro- program must be able to jump to each other. This is the condition where the fifth operation is required.
5) The microinstruction in the control store should be able to jump to another during execution.

Every microinstruction is expected to be able to execute the five operations described above.

4.1 Microinstruction Format

Figure 4 shows the format of the microinstructions.

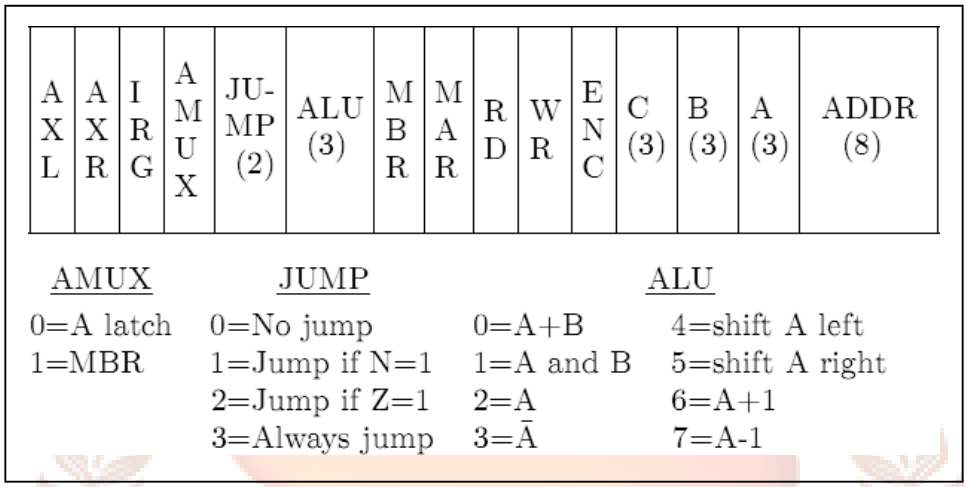


Figure 4: Microinstruction Format

Notice the following points:

1. All the fields where no width is indicated are 1-bit wide.
2. You may also note that the microinstructions don't have an opcode, unlike machine instructions. Therefore, they are simpler than machine instructions. This is the reason why a microinstruction can be fetched from the control store and can be executed immediately, without the need for any opcode decoding.

Now we'll discuss the meaning of the individual fields:

- The AXL (Auxiliary left) and AXR (Auxiliary Right) fields direct the way the AX register is loaded from the ADDR field (address field).
- The IRG bit controls the loading of the opcode from the leftmost 4 bits of the IR (Instruction Register) to the MPC.
- The AMUX (A-multiplexer) field controls the setting of the A-multiplexer. In order to flip the A-multiplexer to the left (MBR), it is essential to have a 1 bit in this field.
- The JUMP field is 2-bit wide. It specifies the type of jump the microinstruction requires. There are two types of conditional jumps, depending on the values of the status flags Z and N. One is the unconditional jump, and the other one is no jump. ADDR field contains the jump address.
- The ALU specifies the following eight possible operations. These are:
  - **Code 0= A+B**; The code 0 adds the value of A and B

- **Code 1=A ANDB**; This code passes the AND result of A and B
- **Code 2=A**; This code means that the ALU will simply pass the: value of the A latch through, and disregard the input from the B latch
- **Code3 = Ā**; This Code is similar to code 2, but in this, the value being passed is 1s complemented while going through the ALU.
- **Code 4= shift A left**
- **Code 5= shift A right**
- **Code 6= A+1;** Code 6 increments the A input by 1.
- **Code 7=A-1;** This code decrements the A input by 1

4.2 Microinstruction Timing

Timing plays a very vital role in the microinstruction execution. Although any microinstruction can perform all five operations given above in section 13.4, it does not make sense for it to perform them simultaneously. It is essential to give time to stabilize the voltages at the latches after selecting the two registers and sending them to the A and B latches. This ensures that the ALU can operate them reliably. It is crucial to provide ALU with enough time to finish the operations. Only after the completion, the ALU results can be moved further. It is because of this that the timing is important. In our example, we have assumed that each microinstruction is fetched and is fully executed in one clock cycle only. It has been previously mentioned that the clock cycle is divided into 4 sub-cycles (fetch cycle, decode cycle, execute cycle, output cycle).

Now we will study what happens during each sub-cycle.

1. In sub-cycle 1 or the Fetch cycle, the microinstruction is fetched from the control store. The MPC (microprogram counter) now will point to the next microinstruction as it would have been updated in the previous sub-cycle 4. Also, the control store tries to output the next microinstruction to the MIR (Microinstruction counter). The clock output line 1 opens the gates between the control store and the MIR, moving the next microinstruction into the MIR (Figure 13.5). It is important to notice that all the fields of the MIR are active at the end of this sub-cycle. Those fields that are used in later sub-cycles should thus be blocked with the help of gates.

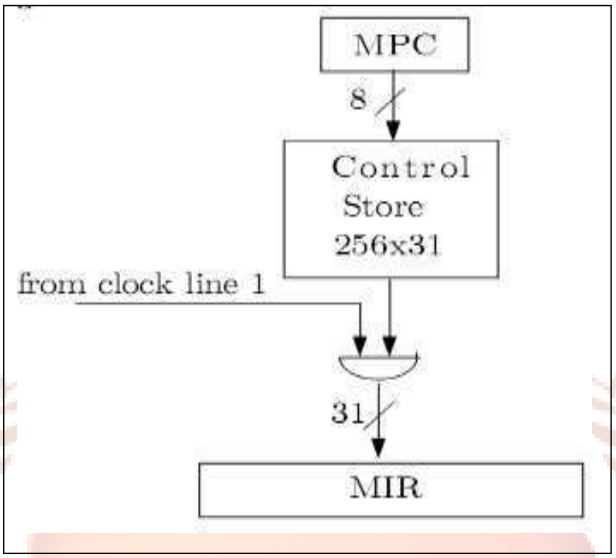


Figure 5: Fetching a Microinstruction

2. In the next sub-cycle, the two input registers namely A and B are selected. They are then sent to the A and B buses and thereafter to the two latches. At the time the voltages at the latches are given the time to stabilize. The 3-bit 'A' field in the MIR becomes the input of the A- decoder. Only one of the 8 decoder outputs can be high, and that one opens a gate (actually, 16 gates) that moves the selected register to the A bus and from there, in sub-cycle 2, to the A latch. Figure 13.6 shows the details of moving data (16 bits) from the MBR to the AMUX. This is controlled by 16 AND gates which are opened when the 'MBR' field in the MIR is 1. This field is updated in sub-cycle 1 so, if it is 1, the MBR will move to the AMUX and from there, (if the AMUX is switched to the left) to the ALU, during sub-cycle 2. At the end of the sub-cycle-2, ALU has stable inputs.
3. Sub-cycle 3 (or the execute cycle) is devoted to the ALU. In this sub-cycle, the ALU is always active. You will note that there are no gates at the ALU inputs and hence, there is no way to enable or disable it. The ALU is always active. It always looks at its inputs and uses them to generate the correct output. The ALU disregards its output until the moment it is believed to be correct. At the moment the ALU inputs are right it marks the start of sub-cycle
4. Sub-cycle 4: In this sub-cycle, the ALU output is moved to its destination. There can be two destinations for this one is the C bus and the other is the MBR.

Figure 6 shows the details of moving the ALU to the MBR.

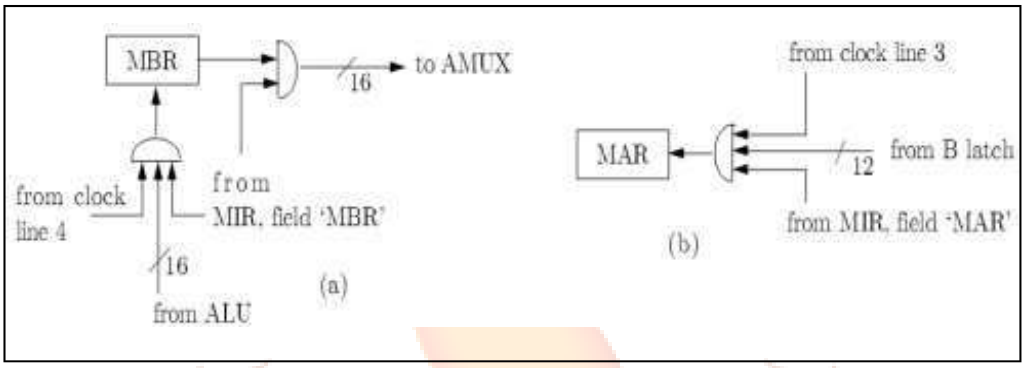


Figure 6: Details at the MAR and MBR

This move is controlled by 16 gates which open only when the 'MBR' field in the MIR is 1 and it is sub-cycle 4. During sub-cycle 4, a microinstruction jump also occurs. A jump in a microprogram is done by resetting the MPC similar to resetting the PC in the case of a program. There is always an increment by 1 in the value of the MPC and it is sent back to the MMUX multiplexer, from which it goes back to the MPC in sub-cycle 4.

You can recall that a microinstruction can jump, conditionally or unconditionally to another microinstruction based on the value of the 'JUMP' field and on the values of the two status flags Z (Zero) and N(Sign flag that denotes Negative). Extra hardware is therefore required, to observe the 2 bits of the 'JUMP' field (which are labeled L (left) and R (right)) and the status flags, to make a decision whether or not a jump should take place. If the jump is required then the MMUX is switched to the right. This special set of hardware is called the **Micro Sequencing Logic** (See figure 7).

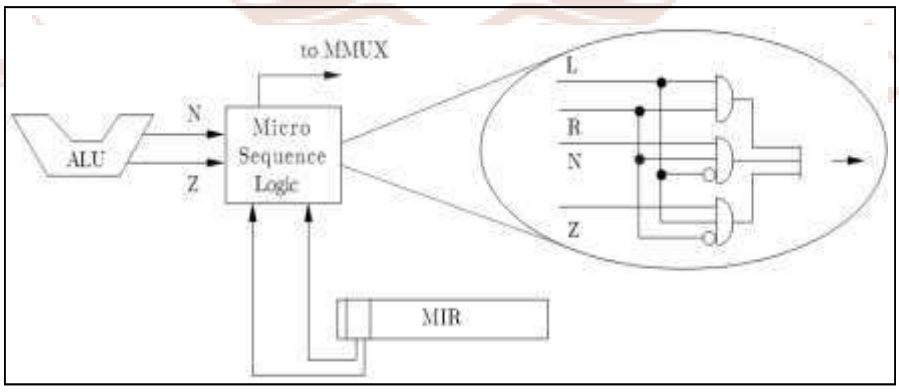


Figure 7: Micro Sequencing Logic

This micro sequencing logic is very simple. It consists of 3AND gates and 1 OR gate. A jump should take place if any of the following three conditions are met:

(1) The 'JUMP' field equals 3 (L=1, R=1);

(2) 'JUMP'=1 (L=0, R=1) AND N=1;

(3) 'JUMP'=2 (L=1, R=0) AND Z=1.

Self-Assessment Questions –3

6. _____field contains the jump address.
7. The JUMP field is 2-bit wide.(True/False)

Activity 2

Consider the 3 function input lines to the ALU. They tell the ALU which of its 8 operations to perform. Demonstrate when does the ALU look at these inputs?

5. CONTROL PATH

In this section, you will learn a few more features that improve the operations of a computer. One of these features is that it deals with decoding the current machine instruction. A real hardwired computer comprises an opcode decoder whose inputs are the bits of the opcode. As shown in the figure. 13.8 (a), each output of the opcode decoder activates one of the individual execution circuits. Depending on the current opcode, one of the decoder's outputs goes high, which activates the corresponding execution circuit. There are no execution circuits in a real micro-programmed computer. Once the next machine instruction is obtained, the control unit has to perform its micro-program. Therefore, it needs to know where that micro-program initiates in the control store. Generally, this is done with the help of a small, separate ROM that contains the start addresses of all the micro-programs in the control store. Once the control unit tracks the opcode of the current instruction from the IR, it sends it to the address bus of that ROM. While sending an address to a ROM, the consequent data comes out on the data bus of the ROM (sending a 'read' signal is not required as a ROM is non-writable). The control unit transfers the data (the start address of the right micro-program) to the MPC (Micro program counter) (Figure13.8 (b) depicts an example of such a ROM.

We assume a hypothetical computer that has instructions with opcodes 0, 1, 3, and 5 and whose micro-programs start at control store locations 25, 56, 74, and 129, respectively. You will notice that not all locations in the ROM are used, as a computer may not utilize every available opcode. For example, a computer with 8-bit opcodes need not always have 256 instructions. In such cases, it is sometimes cheaper to use a Programmable Logic Array (PLA) instead of a ROM. A PLA is quite similar to a ROM but is manufactured such that only locations that are actually used exist. Any memory location that is not used or not going to be used is simply not made-up.

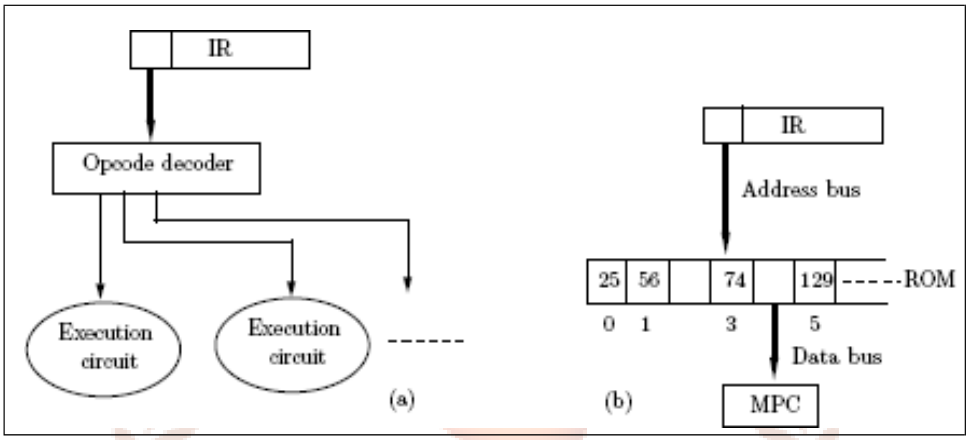
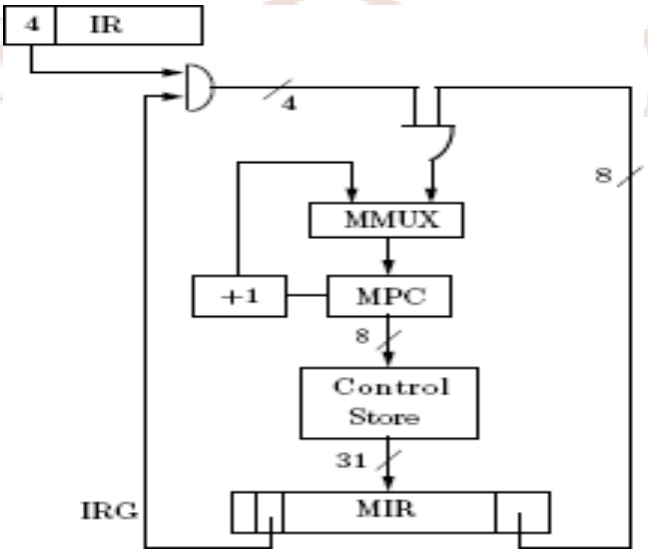


Figure 8: Decoding the Opcode

In our example computer, no ROM has been used. In fact, the 4-bit opcode is moved from the IR (Instruction Register) to the MPC each time the next instruction is obtained from the main memory. Our opcodes, depicted in table 2, have values 1 through 12, which imply that the next microinstruction will be obtained from one of these locations (locations 1 through 12) in the control store. The only thing required is to place 'goto' microinstructions at the locations going to the start of the right micro- programs. The PUSH instruction looks different in table 2 as it has a 6-bit opcode. However, you will learn in section 6 that how the last 2 bits of this opcode are isolated and analyzed by microinstructions. Therefore, the control unit can treat this instruction as if it had the 4-bit opcode 1011.

Figure 9 depicts the details of the movement of the opcode of the current machine instruction from the IR to the MPC through the MMUX (multi-channel multiplexer).

Figure 9: Moving the Opcode to the MPC



The four leftmost bits of the IR are moved to the MMUX, through AND gates followed by OR gates. The microinstruction which does that has to meet the following three conditions:

1) It has to open the AND gates,
2) It should have an ADDR field of all zeros, so as not to send anything through the OR gate, and
3) It has to make sure that the MMUX is turned to the right.

All three conditions can be realized by the simple microinstruction 'irg; goto 0'. With a little thought, a reader can be convinced that a 'goto 0;' combined with an 'irg' does not affect a jump to location zero of the control store. Instead, it moves the eight bits 0000pppp (where pppp are the opcode bits from the IR) to the MMUX and from there, to the MPC. Thus, the MPC is reset to 0000pppp which causes the next microinstruction to be obtained from the location pppp of the control store. As pppp is a 4-bit number, it can be between 0 and 15 (in fact, it should be between 1 and 15). Therefore, locations 1 through 15 of the control store should be set aside for special use. Location L should include the microinstruction 'goto P', where P depicts the start address inside the microprograms control store for the machine instruction which has an opcode called L. Location 0 of the control store is set aside for a 'goto' microinstruction which starts the fetch sequence of the first machine instruction.

The next feature deals with the use of constants. While writing machine instructions, many computers offer the immediate mode, making it easy to use constants. Our microinstructions do not make use of any addressing modes and can only use the existing registers and gates. As constants are helpful, one more feature has been added to our microinstructions, making it feasible to create a 16-bit constant in the AX register. We can use the two fields AXL and AXR of every microinstruction to move the 8-bit ADDR field to the left or right halves of the AX register. Remember that the ADDR field usually contains a jump address; now it also has a second use as a constant. Figure 10 depicts the details of the gates involved in this move.

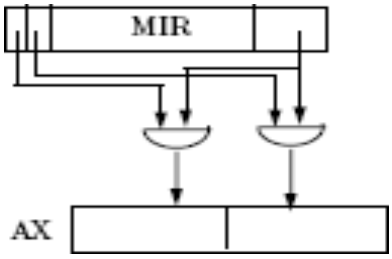


Figure 10: Moving a constant to the AX register.

Self-Assessment Questions –4

8. A real hardwired computer comprises an _____decoder whoseinputs are the bits of the opcode.
9. There are no _____circuits in a real micro-programmedcomputer.

6. MICROCODE

The microinstructions work at a lower level in the computer; they are close to the hardware. However, it is also convenient to write them using a high-level notation, similar to PASCAL or C. This enables the reduction in the number of bugs present in the microinstructions and also makes the microcode simplified to be read. We can also think of a micro-assembler reading microinstruction in this notation and translating each into a 31-bit number because our microinstruction format is 31-bit long (it is 31 based on the bit length of the various fields of the microinstruction as shown in figure 4). We use the principle that each microinstruction is written on one line, with an individual phrase separated from the other with the help of a semicolon. The notation used for various operations performed by a microinstruction is given below:

The operations of selecting 2 registers, specifying an ALU operation, and routing the result to another register are specified by phrases such as:

'r1:=r1+r0;',

'r2:=band(r3,r1);', 'r4:=r3;',

'r0:=inv(r3);', and

'r3:= lshift(inv(r3));',

where band stands for "Boolean AND", inv, for "inverse" and l shift, for "left shift."

The phrase 'alu:=r1;' means selecting R1, moving it to the A latch and, from there, to the ALU, where it is passed through without any processing (ALU function-code 2) and then discarded.

Nothing is moved to either the C bus or the MBR. The outcome of such an operation is new values of the status flags Z and N, so such a phrase is generally combined in the microinstruction with a conditional jump phrase.

Name	Mnemonic	Opcode	Operand	Description
Load direct	LODD R1	0001	12-bit x	R1:=M[x]
Load relative	LODR R1	0010	"	R1:=M[x+PC]
Load indirect	LODN R1	0011	"	R1:=M[M[x]]
Load immediate	LODI R1	0100	"	R1:=x
Unconditional Jump	JMP	0101	"	PC:=x
Indirect jump	JMP R1	0110	none	PC:=R1
Conditional jump	JZER R1	0111	12-bit x	if R1=0, PC:=x
Conditional jump	JNEG R1	1000	"	if R1< 0, PC:=x
Procedure call	CALL	1001	"	SP:=SP−1; M[SP]:=PC; PC:=x
Procedure return	RET	1010	none	PC:=M[SP]; SP:=SP+1
Push register	PUSH Ri	1011ii*	"	SP:=SP−1; M[SP]:=Ri
Increment SP	INSP	1100	8-bit y	SP:=SP+y

Table 2: The Machine Instructions

6.1 Fully Horizontal Microcode

A single bit in the control word can be allocated to every control signal. This is shown in figure 11. It seems to be the simplest solution.

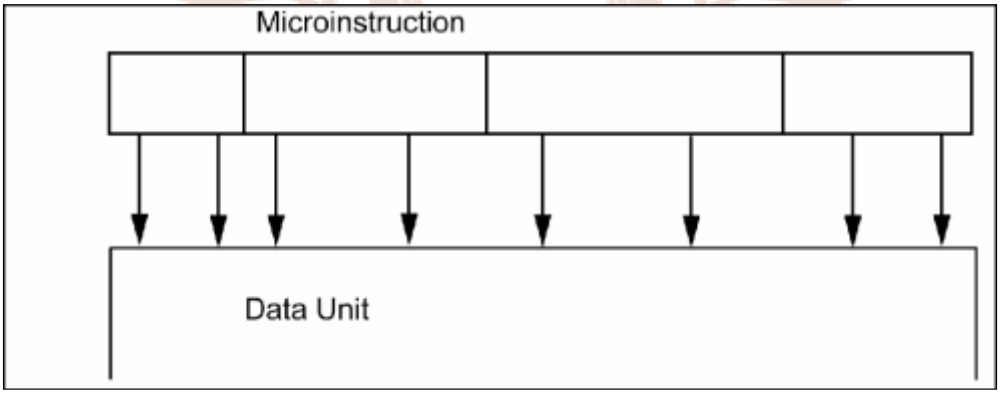


Figure 11: Fully-Horizontal Microcode

Figure 11 represents a fully-horizontal microcode. This is clearly at a disadvantage as all control signals lying on the data path will never get activated at the same time. Therefore, it would need a great deal wider microcode words than that is required. However, the Microcode words are generally wide that is of 100 bits or more. In case, a data path has plenty of control signals, the horizontal microcode will make it a lot wider.

But, it also has a couple of advantages, viz,

1. Several data path signals are able to be activated at the same time, which boosts potential parallelism; and
2. Since there is the absence of extra hardware between the data path and controller which could reduce the speed of the processor, the horizontal microcode promises to be fast.

6.2 Fully Vertical Microcode

The control signals happen to be fully encoded in the control word in a fully- vertical microcode. This signifies that if a data path possessed 16 control signals, these would be programmed as four bits. This has been shown in figure 12.

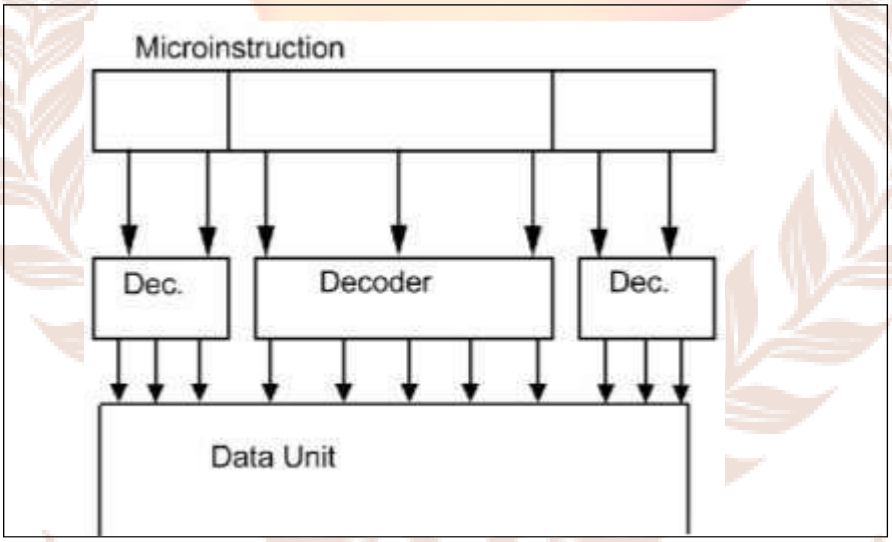


Figure 12: Fully-Vertical Microcode

Thus, the microcode memory required is saved. However, any chance for parallelism is eliminated and a decoder is added between the data path and the controller.

The decoder will utilize a few of the space/resources saved by decreasing the microcode memory size. Additionally, it will also help in delaying the signals. This illustrates a time-space trade-off.

Self-Assessment Questions –5

10. The microinstructions work at a low level; they are close to the hardware. (True/False)
11. In _____ microcode, the control word fully encoded files control signals.

7. MACHINE INSTRUCTIONS

A computer should have a set of machine instructions that includes all the important types of the instruction set. Our computer is just an example, so it has been assigned a small, incomplete instruction set, specifically designed to illustrate the micro-programs. All instructions are 16-bits long but one has a 4-bit opcode. A few, simple addressing modes are used.

Table 2 shows that some instructions use register R1. On a computer, each of those instructions should be able to use any of the 4 general-purpose registers. Important instructions, such as STORE, ADD, SUBTRACT, and SHIFT are missing since their micro-programs are similar to the ones for LOAD. Another important point to notice is that the machine instructions can use the 4 general-purpose registers and the SP, but they cannot use the PC, IR, or AX. The latter registers are used only by the microinstructions. This is another aspect of the microinstructions being low-level. They can access hardware components that even machine instructions cannot.

Self-Assessment Questions –6

12. A computer should have a set of machine instructions that includes all the important types of _____ set.
13. Machine instructions can use the 4 general-purpose registers and the SP, but they cannot use the PC, IR or AX. (True/False)

8. SUMMARY

In this unit, we discussed about the concepts of microprogramming, Computer Clock, Microinstructions and its timing, control path, microcode, and machine instructions. Let us recapitulate the important concepts discussed in this unit:

- Microprogramming was proposed by Maurice Vincent Wilkes of Cambridge University.
- Microprogramming is an organized method of designing the control unit (CU) of a computer. In this, the control unit fetches and executes microinstructions instead of machine instructions. The main underlying concept beneath microprogramming is to have all the information on instruction execution stored in a special memory space termed as the control store.
- Microinstructions are simple and function at a lower level i.e. they are close to the hardware as compared to the machine instructions. The complete set of steps required to process a machine instruction is called the microprogram.
- The clock is an oscillator that produces an output signal shaped like a square wave.
- A microinstruction is an instruction that controls data flow and instruction-execution sequencing in a processor at a more fundamental level than machine instructions.
- It is essential to give time to stabilize the voltages at the latches after selecting the two registers and sending them to the A and B latches. This ensures that the ALU can operate them reliably.

9. GLOSSARY

- **CISC:** Complex Instruction set computers
- **Computer clock:** A Computer clock is an oscillator that produces an output signal shaped like a square wave.
- **Control store:** A special memory space to store all microcode.
- **Microprogram:** A collection of microinstructions for the particular machine instruction.
- **Microprogramming:** Microprogramming is an orderly method of designing the control unit of a conventional computer
- **PLA:** Programmable Logic array
- **RISC:** Reduced Instruction set computers

10. TERMINAL QUESTIONS

1. Explain the meaning of microprogramming? Who gave the concept of microprogramming?
2. What is a microinstruction? Where is it used?
3. Describe briefly the history of microprogramming.
4. Discuss the role of computer clock in microprogramming.
5. Describe the format for a microinstruction.
6. Write a short note on microinstruction timing.
7. Describe the control path for executing a micro-program.
8. Describe a microcode.
9. What do you understand by fully vertical and fully horizontal microcode? Explain using an example.
10. Write a short note on machine instructions.

11. ANSWERS

Self-Assessment Questions

1. Emulation
2. Complex Instruction set computers
3. False
4. Square
5. False
6. ADDR
7. True
8. Opcode
9. Execution
10. True
11. Fully vertical
12. Instruction
13. True

Terminal Questions

1. The principle of microprogramming is to have the control unit fetch and execute by use of microinstructions instead of machine instructions. Refer Section 2.
2. The microinstructions are similar to machine instructions but are simpler and operate at a lower level (they are closer to the hardware). Refer Section 2.
3. Maurice Vincent Wilkes of Cambridge University gave rise to the concept of microprogramming'. Refer Section 2.
4. Computer clock plays a vital role in carrying out signals in a microinstruction execution. Refer Section 3.
5. Microinstructions are components of microcode. Refer Section 4.
6. Timing plays a very vital role in the microinstruction execution. Refer Section 4.
7. A real hardwired computer comprises of an opcode decoder whose inputs are the bits of the opcode. Refer Section 5.
8. The entire content of the control store is sometimes referred to as the *microcode.* Refer Section 6.

9. A single bit in the control word can be allocated to every control signal in fully horizontal microcode. Refer Section 6.
10. Machine instructions are high level instructions. Refer Section 7.

References

- Sajjan G. Shiva; Computer Design and Architecture; Marcel Dekker
- Silvia Melitta Mueller, Wolfgang J. Paul; Computer Architecture; Springer
- Joseph D. Dumas II; Computer Architecture; CRC Press
- Nicholas P. Carter; Schaum's outline of computer Architecture; Mc. Graw-Hill Professional

E-references

- http://cnx.org/content/m29711/latest/
- http://euler.mat.uson.mx/~havillam/ca/CS323/0708.cs-323003.html
- http://doit.ort.org/course/cont_frame.htm
- http://www.davidsalomon.name/