UNIT-2

UNIT - II: BASIC COMPUTER ORGANIZATION AND DESIGN, MICROPROGRAMMED CONTROL

(09 periods)

Basic Computer Organization And Design: Instruction codes, Computer Registers, Computer instructions, Timing and control, Instruction cycle, Memory Reference Instructions, Input - Output and Interrupt.

Micro Programmed Control: Control memory, Address sequencing, Design of control unit, Hard wired control, Micro-programmed control.

2.1 Instruction Codes

A computer instruction is a binary code that specifies a sequence of microoperations for the computer. Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations. Every computer has its own unique instruction set. The ability to store and execute instructions, the stored program concept, is the most important property of a general-purpose computer.

An instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation. The most basic part of an instruction code is its operation code operation part. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer. The operation code must consist of at least 11 bits for a given 2^n (or less) distinct operations.

The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory. An instruction code must therefore specify not only the operation but also the registers or the memory words where the operands are to be found, as well as the register or memory word where the result is to be stored. Memory words can be specified in instruction codes by their address. Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2^k registers. There are many variations for arranging the binary code of instructions, and each computer has its own particular instruction code format. Instruction code formats are conceived by computer designers who specify the architecture of the computer.

Stored Program Organization

The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address. The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.

Figure 5-1 Stored program organization.

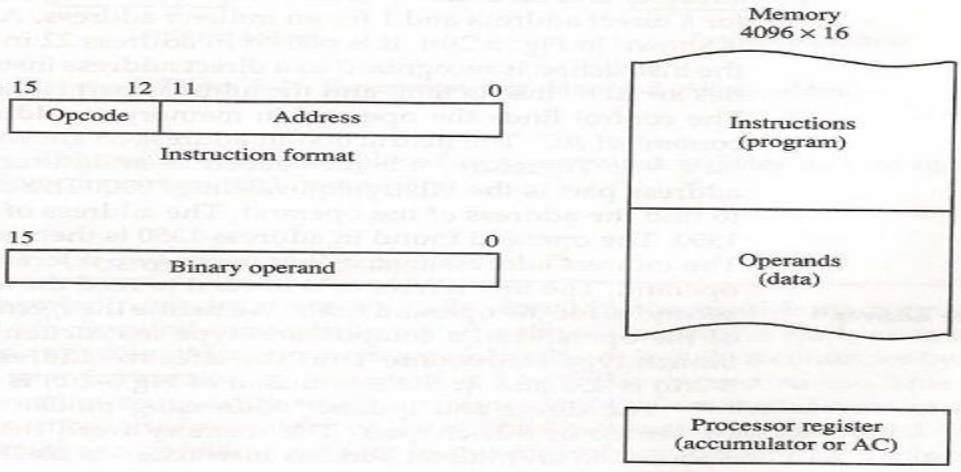


Figure 5-1 depicts this type of organization. Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits to specify an address since 2^{12} = 4096. If we store each instruction code in one 16-bit memory word, we have available four bits for the opcode Operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand. The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory. It then executes the operation specified by the operation code. Computers that have a single-processor register usually assign to it the name accumulator (AC) accumulator and label it AC. The operation is performed with the memory operand and the content of AC.

Indirect Address

It is sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand. When the second part of an instruction immediate code specifies an operand, the instruction is said to have an immediate instruction operand. When the second part specifies the address of an operand, the instruction is said to have a direct address. This is in contrast to a third possibility called indirect address, where the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found. One bit of the instruction code can be used to distinguish between a direct and an indirect address.

As an illustration of this configuration, consider the instruction code format shown in Fig. 5-2(a). It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I. The mode bit is 0 for a direct address and 1 for an indirect address. A direct address instruction is shown in Fig. 5-2(b). It is placed in address 22 in memory. The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457. The control finds the operand in memory at address 457 and adds it to the content of AC. The instruction in address 35 shown in Fig. 5-2(c) has a mode bit I = 1. Therefore, it is recognized as an indirect address instruction.

The indirect address instruction needs two references to memory to fetch an operand. The first reference is needed to read the address of the operand; the effective address second is for the operand itself. We define the effective address to be the address

of the operand in a computation-type instruction or the target address in a branch-type instruction. Thus the effective address in the instruction of Fig. 5-2(b) is 457 and in the instruction of Fig 5-2(c) is 1350.

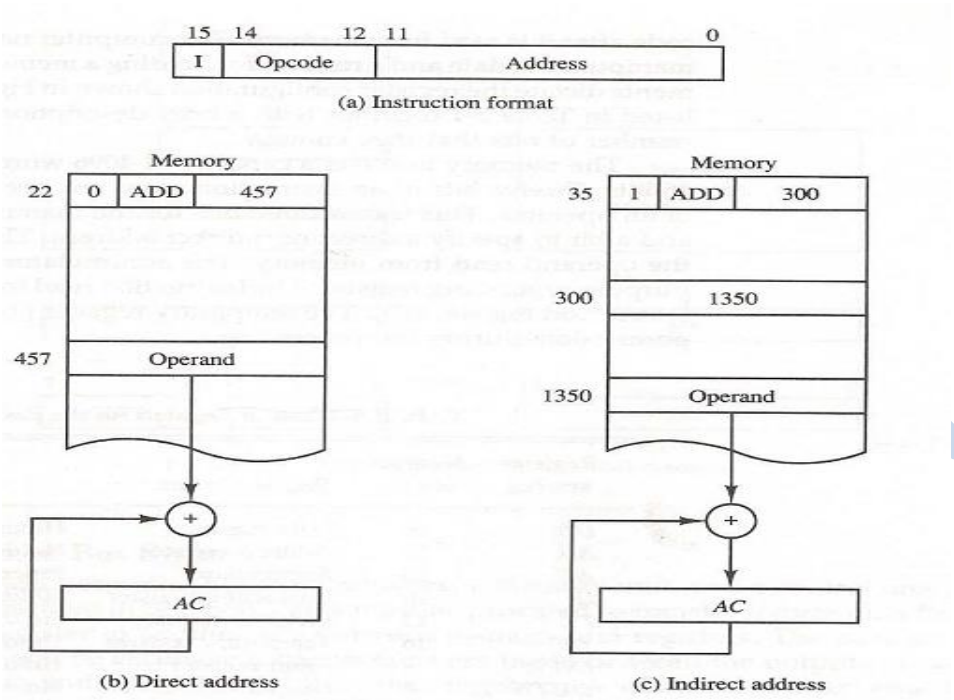


Figure 5-2 Demonstration of direct and indirect address.

2.2 Computer Registers

Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it, and so on. This type of instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed. It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory. The computer needs processor registers for manipulating data and a register for holding a memory address. These requirements dictate the register configuration shown in Fig. 5-3. The registers are also listed in Table 5-1 together with a brief description of their function and the number of bits that they contain.

TABLE 5-1 List of Registers for the Basic Computer

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

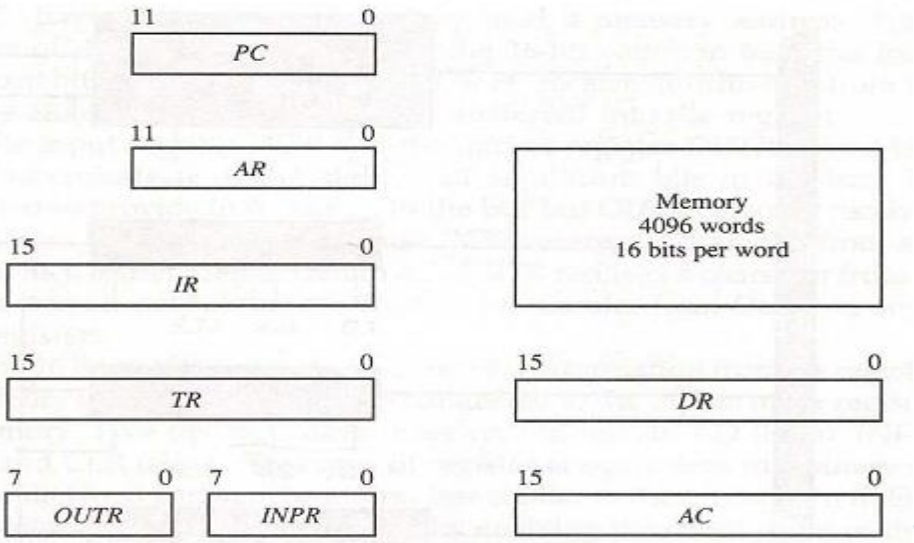


Figure 5-3 Basic computer registers and memory.

Common bus:

The basic computer has eight registers, a memory unit, and a control unit (to be presented in Sec. 5-4). Paths must be provided to transfer information from one register to another and between memory and registers. The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers. A more efficient scheme for transferring information in a system with many registers is to use a common bus. The connection of the registers and memory of the basic computer to a common bus system is shown in Fig. 5-4.

The outputs of seven registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0. The number along each output shows the decimal equivalent of the required binary selection. For example, the number along the output of DR is 3. The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0$= 011 since this is the binary value of decimal 3. The lines from the common bus are connected to the inputs of each register and the data inputs of the memory. The particular load (LD) register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition. The memory receives the contents of the bus when its write input is activated. The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0$ = 111.

Four registers, DR, AC, IR, and TR, have 16 bits each. Two registers, AR and PC, have 12 bits each since they hold a memory address. When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to 0'3. When AR or PC receives information from the bus, only the 12 least significant bits are transferred into the register.

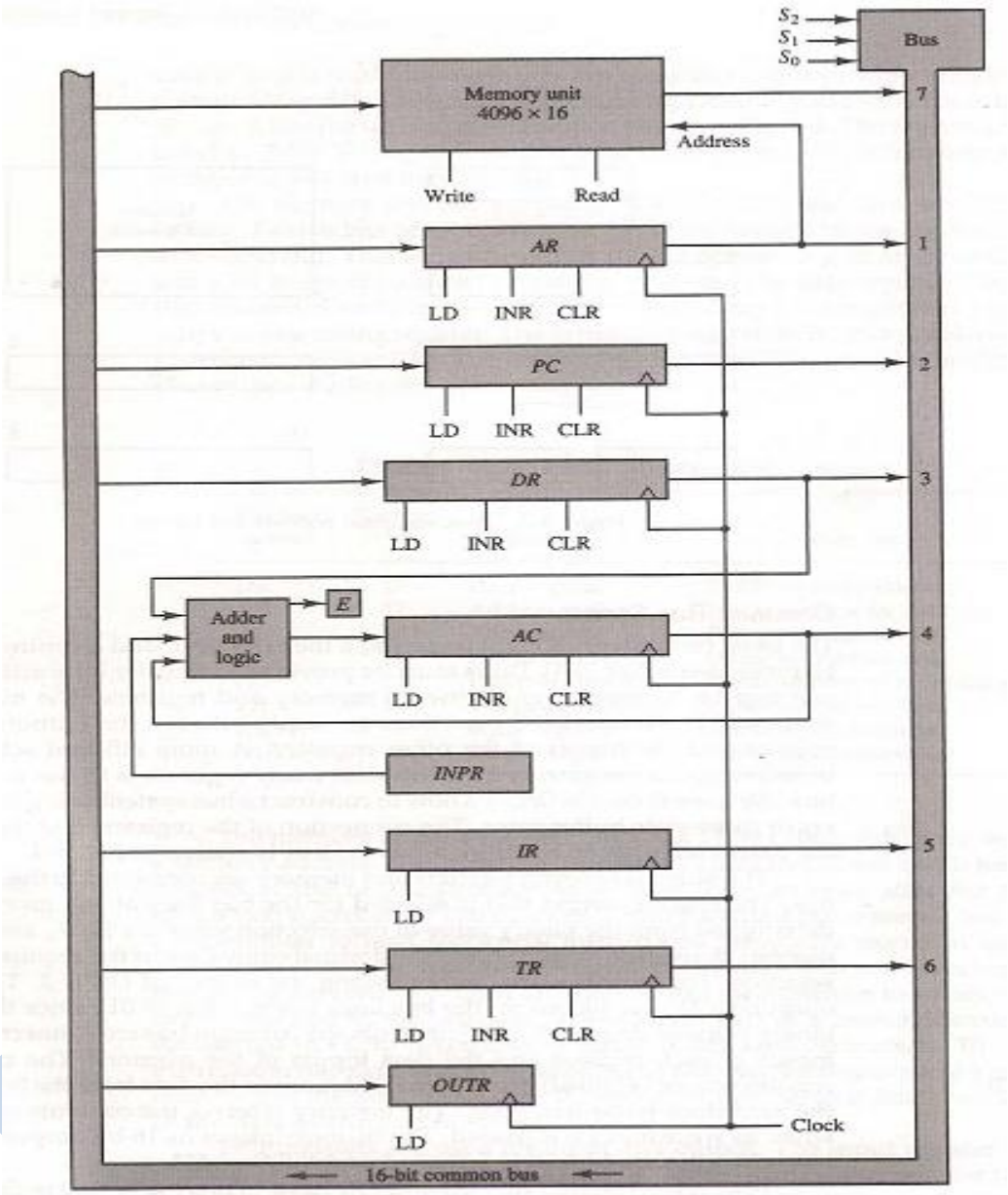


Figure 5-4 Basic computer registers connected to a common bus.

The input register INPR and the output register OUT R have 8 bits each and communicate with the eight least significant bits in the bus. INPR is connected to provide information to the bus but OUTR can only receive information from the bus. This is because INPR receives a character from an input device which is then transferred to AC. OUTR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers.

The input data and output data of the memory are connected to the common bus, but the memory address is connected to AR. Therefore, AR must memory address always be used to specify a memory address.

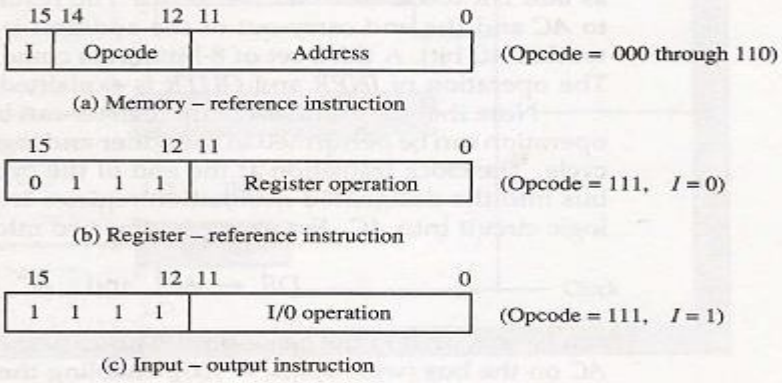
The 16 inputs of AC come from an adder and logic circuit. This circuit has three sets of inputs. One set of 16-bit inputs come from the outputs of AC. They are used to implement register microoperations such as complement AC and shift AC. Another set of 16-bit inputs come from the data register DR. The inputs from DR and AC are used for

arithmetic and logic microoperations, such as add DR to AC or AND DR to AC. The result of an addition is transferred to AC and the end carry-out of the addition is transferred to flip-flop E (ex- tended AC bit). A third set of 8-bit inputs come from the input register INPR.

2.3 Computer Instructions

The basic computer has three instruction code formats, as shown in Fig. 5-5. Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered. A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address (see Fig. 5-2). The register-reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed. Similarly, an input-output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.

Figure 5-5 Basic computer instruction formats.



The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction. If the three opcode bits in positions 12 though 14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I. If the 3-bit opcode is equal to 111, control then inspects the bit in position 15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input-output type.

TABLE 5-2 Basic Computer Instructions

Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Register-reference instructions use 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7. The other three hexadecimal digits give the binary equivalent of the remaining 12 bits. The input-output instructions also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F

Instruction Set Completeness

The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

1. Arithmetic, logical, and shift instructions
2. Instructions for moving information to and from memory and processor registers
3. Program control instructions together with instructions that check status conditions
4. Input and output instructions

Arithmetic, logical, and shift instructions provide computational capabilities for processing the type of data that the user may wish to employ. The bulk of the binary information in a digital computer is stored in memory, but all computations are done in processor registers. Therefore, the user must have the capability of moving information between these two units. Decision making capabilities are an important aspect of digital computers. For example, two numbers can be compared, and if the first is greater than the second, it may be necessary to proceed differently than if the second is greater than the first. Program control instructions such as branch instructions are used to change the

sequence in which the program is executed. Input and output instructions are needed for communication between the computer and the user. Programs and data must be transferred into memory and results of computations must be transferred back to the user.

2.4 Timing and Control

The timing for all registers in the basic computer is controlled by a master clock clock pulses generator. The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit. The clock pulses do not change the state of a register unless the register is enabled by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator. There are two major types of control organization: *hardwired control* and *microprogrammed control.*

Hardwired control

- In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation.
- A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has microprogrammed to be modified or changed.

Microprogrammed control.

- In the microprogrammed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.
- In the microprogrammed control, any required control changes or modifications can be done by updating the microprogram in control memory.

The block diagram of the control unit is shown in Fig. 5-6. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR).

The instruction register is shown again in Fig. 5-6, where it is divided into three parts: the I bit, the operation code, and bits 0 through 11. The operation code in bits 12 through 14 are decoded with a 3 X 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol 1. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in-binary from 0 through 15. The outputs of the timing signals counter are decoded into 16 timing signals T_0 through T_{15}.

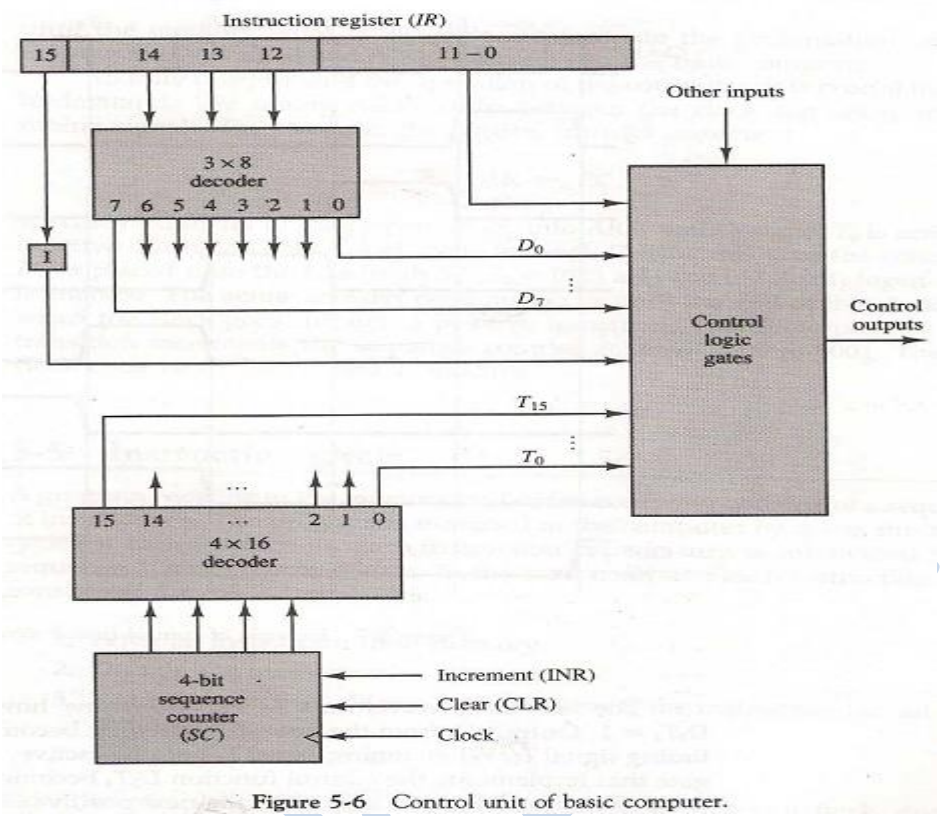


Figure 5-6 Control unit of basic computer.

Fig. 5-7 show how SC is cleared when $D_3T_4 = 1$. Output D3 from the operation decoder becomes active at the end of timing signal T2. When timing signal T4 becomes active, the output of the AND gate that implements the control function D3T4 becomes active. This signal is applied to the CLR input of SC. On the next positive clock transition (the one marked T4 in the diagram) the counter is cleared to 0. This causes the timing signal To to become active instead of T5 that would have been active if SC were incremented instead of cleared.

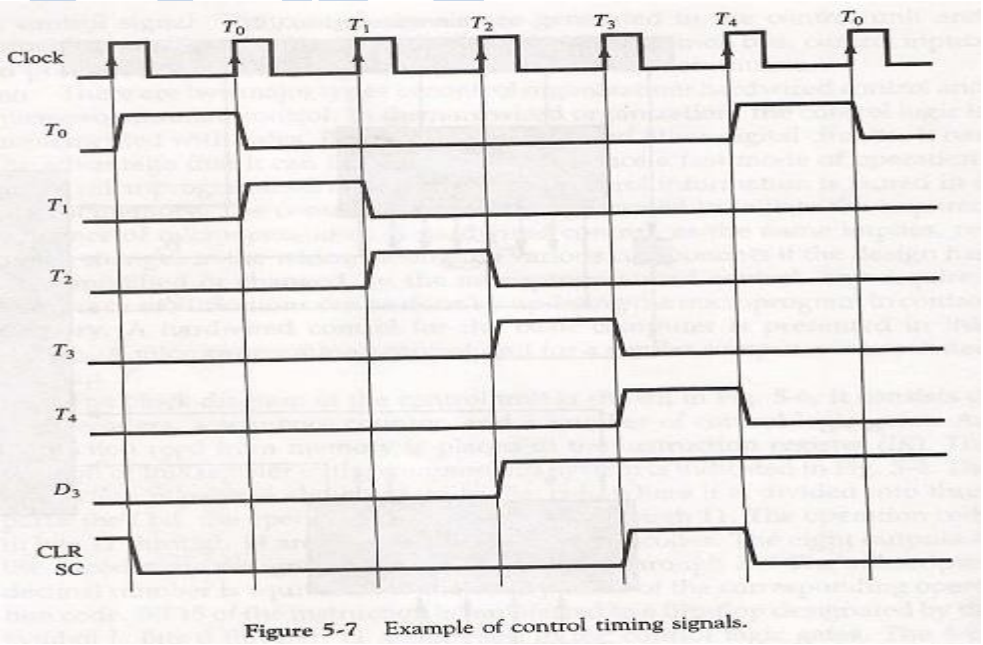


Figure 5-7 Example of control timing signals.

A memory read or write cycle will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, a memory read or write cycle initiated by a timing signal will be completed by the time the next clock goes through its positive transition. The clock transition will then be used to load the memory word into a register.
For example, the register transfer statement

T_o: AR ← P C

Specifies a transfer of the content of PC into AR if timing signal T_o is active.

2.5 Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases. In the basic computer each instruction cycle consists of the following phases:
1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Fetch and Decode

Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal To. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence To, T1, T2, and so on. The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

To: AR ← PC
T1: IR ←M[AR], PC ←PC + 1
T 2: Do, . . . , D7←Decode IR(12-14), AR ←IR(0-11), I ←IR(15)

Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal To. The instruction read from memory is then placed in the instruction register IR with the clock transition associated With timing signal T1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR. Note that SC is incremented after each clock pulse to produce the sequence To, T1, and T2.
Figure 5-8 shows how the first two register transfer statements are implemented in the bus system. To provide the data path for the transfer of PC to AR we must apply timing signal To to achieve the following connection:
1. Place the content of PC onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR.

The next clock transition initiates the transfer from PC to AR since To = 1. In order to implement the second statement.
T1: IR ←M[AR], PC ←PC + 1
it is necessary to use timing signal T1 to provide the following connections in the bus system.
1. Enable the read input of memory.

2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

The next clock transition initiates the read and increment operations since T1 = 1.

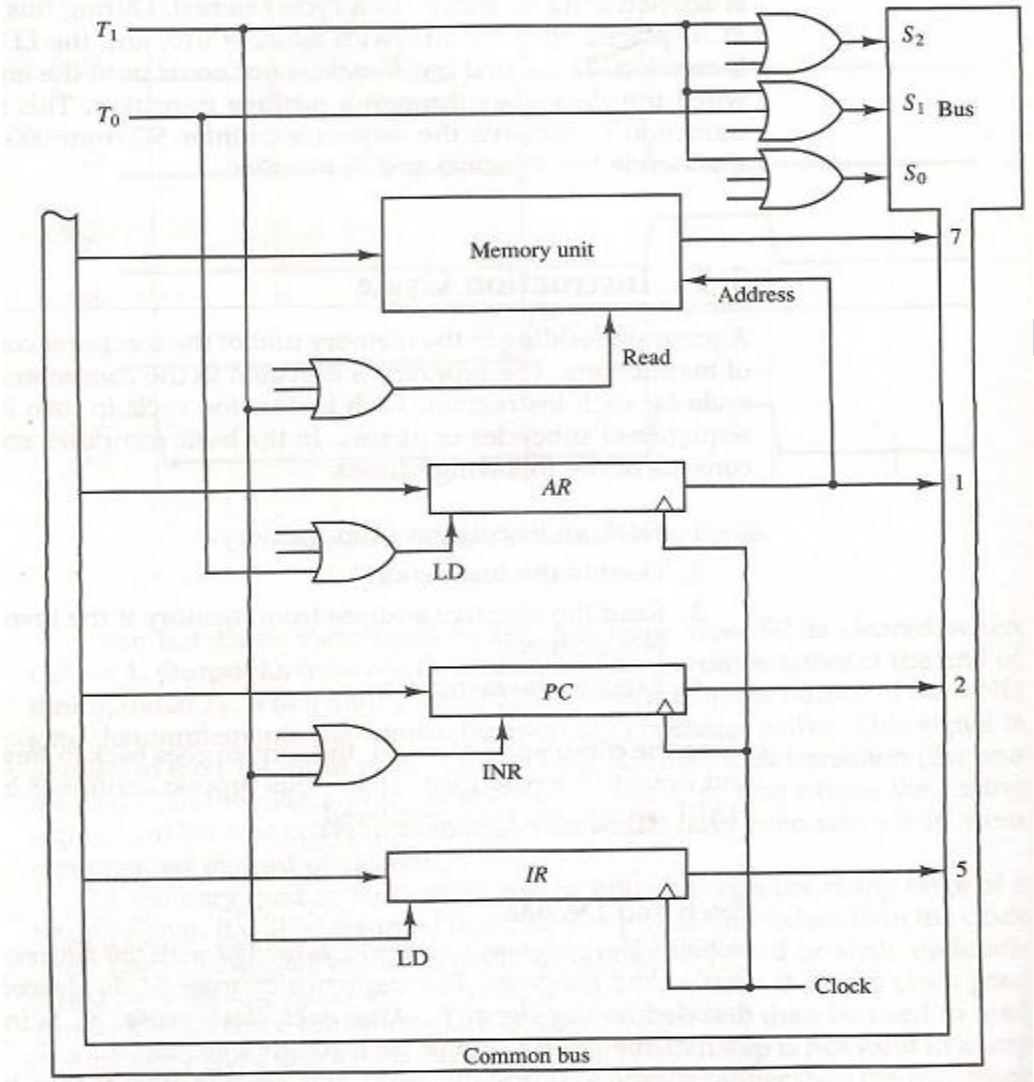


Figure 5-8 Register transfers for the fetch phase.

Determine the Type of Instruction

The timing signal that is active after the decoding is T3. During time T3, the control unit determines the type of instruction that was just read from memory. The flowchart of Fig. 5-9 presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.

Decoder output 07 is equal to 1 if the operation code is equal to binary 111. From Fig. 5-5 we determine that if D7 = 1, the instruction must be a register-reference or input-output type. If D7 = O, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction. Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I . If D7 = 0 and 1 = 1, we have a memory- reference instruction with an indirect address. It is then necessary to read the

effective address from memory. The microoperation for the indirect address indirect address condition can be symbolized by the register transfer statement
AR ←M [AR]
Initially, AR holds the address part of the instruction. This address is used during the memory read operation. The word at the address given by AR is read from memory and placed on the common bus. The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word.

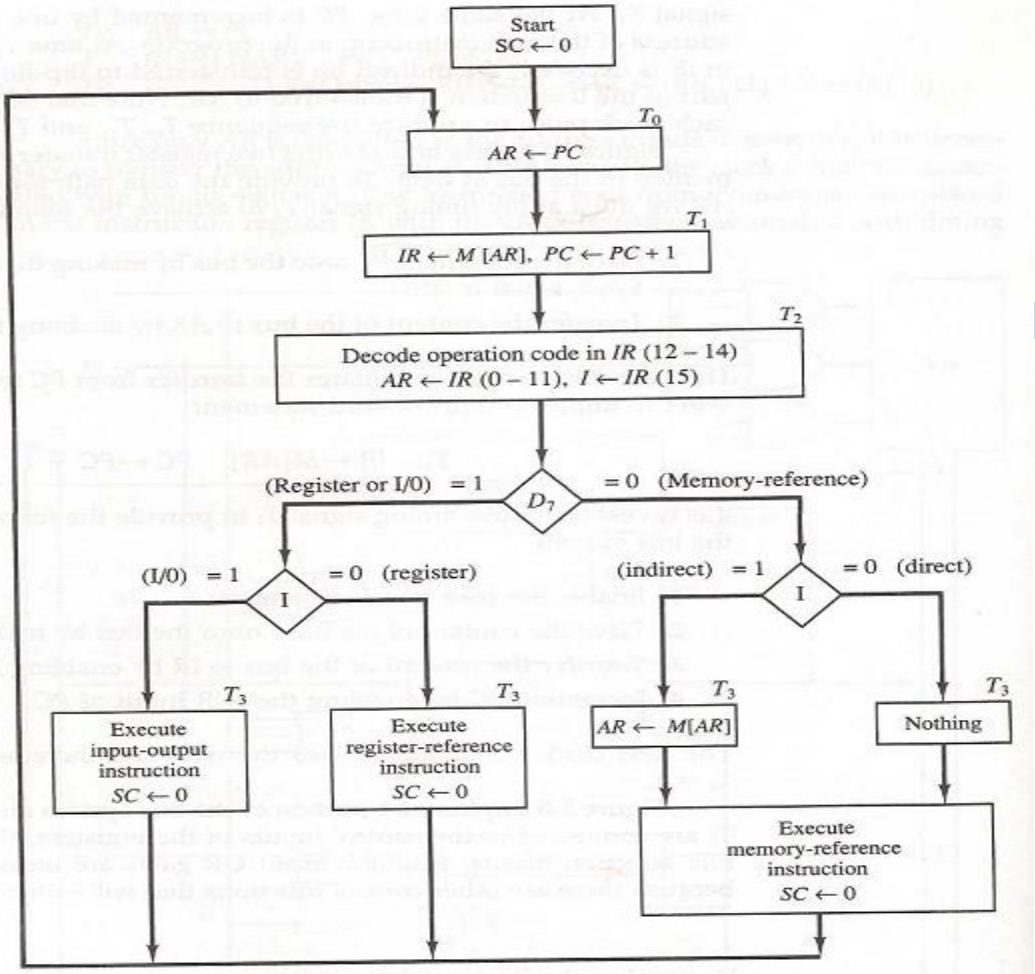


Figure 5-9 Flowchart for instruction cycle (initial configuration).

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T3. This can be symbolized as follows:

D7'IT3: AR ←M[AR]
D7'I'T3: Nothing
D7 I'T3: Execute a register-reference instruction
D7IT3: Execute an input-output instruction

When a memory-reference instruction with I = O is encountered, it is not necessary to do anything since the effective address is already in AR. However, the sequence counter SC must be incremented when D§T3 = 1, so that the execution of the memory-reference instruction can be continued with timing variable T4. A register-reference or input-output instruction can be executed with the clock associated with timing signal T 3. After the

instruction is executed, SC is cleared to 0 and control returns to the fetch phase with To=1.

Note that the sequence counter SC is either incremented or cleared to 0 with every positive clock transition. We will adopt the convention that if SC is incremented, we will not write the statement SC (-SC + 1, but it will be implied that the control goes to the next timing signal in sequence. When SC is to be cleared, we will include the statement SC <-0.

Register-Reference Instructions

Register-reference instructions are recognized by the control when D7=1 and I = 0. These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in IR(0-11). They were also transferred to AR during time T2.

The control functions and microoperations for the register-reference instructions are listed in Table 5-3. These instructions are executed with the clock transition associated with timing variable T3. Each control function needs the Boolean relation D7I' T3, which we designate for convenience by the symbol r.

TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

2.6 Memory-Reference Instructions

In order to specify the microoperations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely.

Table 5-4 lists the seven memory-reference instructions. The decoded output D_i for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs effective address to each instruction is included in the table. The effective address of the instruction is in the address register AR and was placed there during timing signal T; when I = 0, or during timing signal T3 when I = 1. The execution of the memory-reference instructions starts with timing signal T4.

TABLE 5-4 Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

We now explain the operation of each instruction and list the control functions and microoperations needed for their execution. A flowchart that summarizes all the microoperations is presented at the end of this section.

AND to AC

This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC. The microoperations that execute this instruction are:

D0T4: DR ←M[AR]

D0T5: AC ←AC /\ DR, SC ←0

ADD to AC

This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are

D1T4: DR ← M [AR]

D1T5: AC ← AC + DR, E ← C_{out}, SC ← 0

LDA: Load to AC

This instruction transfers the memory word specified by the effective address to AC. The microoperations needed to execute this instruction are

D2T4: DR← M[AR]

D2T5: AC ← DR, SC ← 0

STA: Store AC

This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation:

D3T4: M[AR] ← AC, SC ←0

BUN: Branch Unconditionally

This instruction transfers the program to the instruction specified by the effective address. Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle. PC is incremented at time T1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:

D4T4: PC ← AR, SC ← 0

The effective address from AR is transferred through the common bus to PC. Resetting SC to 0 transfers control to To. The next instruction is then fetched and executed from the memory address given by the new value in PC.

BSA: Branch and Save Return Address

This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the

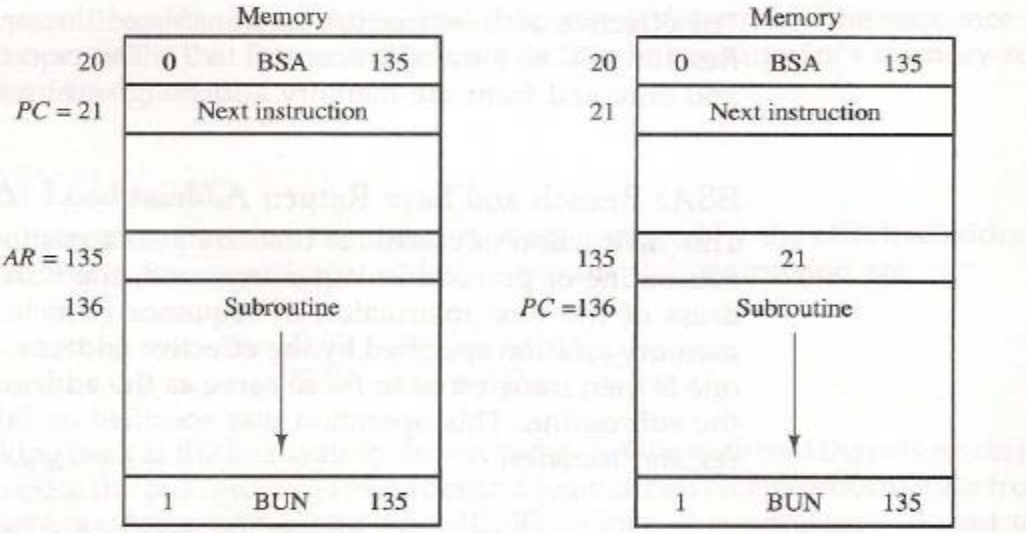
next instruction in sequence (which is available in PC) into a memory location specified by the effective address. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine. This operation was specified in Table 5-4 with the following register transfer:

M[AR] ← PC, PC ← AR + 1

Subroutine call

The BSA instruction performs the function usually referred to as a sub- routine call. The indirect BUN instruction at the end of the subroutine performs the function referred to as a subroutine return. In most commercial computers, the return address associated with a subroutine is stored in either a processor register or in a portion of memory called a stack.

Figure 5-10 Example of BSA instruction execution.



(a) Memory, *PC*, and *AR* at time T_4 (b) Memory and *PC* after execution

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer. To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two microoperations:

D5T4 : M[AR]←PC, AR← AR + 1

D5T5: PC← AR, SC← 0

Timing signal T4 initiates a memory write operation, places the content of PC onto the bus, and enables the INR input of AR. The memory write operation is completed and AR is incremented by the time the next clock transition occurs. The bus is used at T5 to transfer the content of AR to PC.

ISZ: Increment and Skip if Zero

This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by one in order to skip the next instruction in the program.

Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory. This is done with the following sequence of microoperations:

D6T4: DR←M[AR]

D6T5: DR←DR + 1

D6T6: M[AR] ← DR, if (DR = 0) then (PC <- PC + 1), SC ← 0

Control Flowchart

A flowchart showing all microoperations for the execution of the seven memory-reference instructions is shown in Fig. 5-11. The control functions are indicated on top of each box. The microoperations that are performed during time T4, T5, or T6 depend on the operation code value. This is indicated in the flowchart by six different paths, one of which the control takes after the instruction is decoded. The sequence counter SC is cleared to 0 with the last timing signal in each case. This causes a transfer of control to timing signal T_0 to start the next instruction cycle.

Note that we need only seven timing signals to execute the longest instruction (ISZ). The computer can be designed with a 3-bit sequence counter. The reason for using a 4-bit counter for SC is to provide additional timing signals for other instructions that are presented in the problems section.

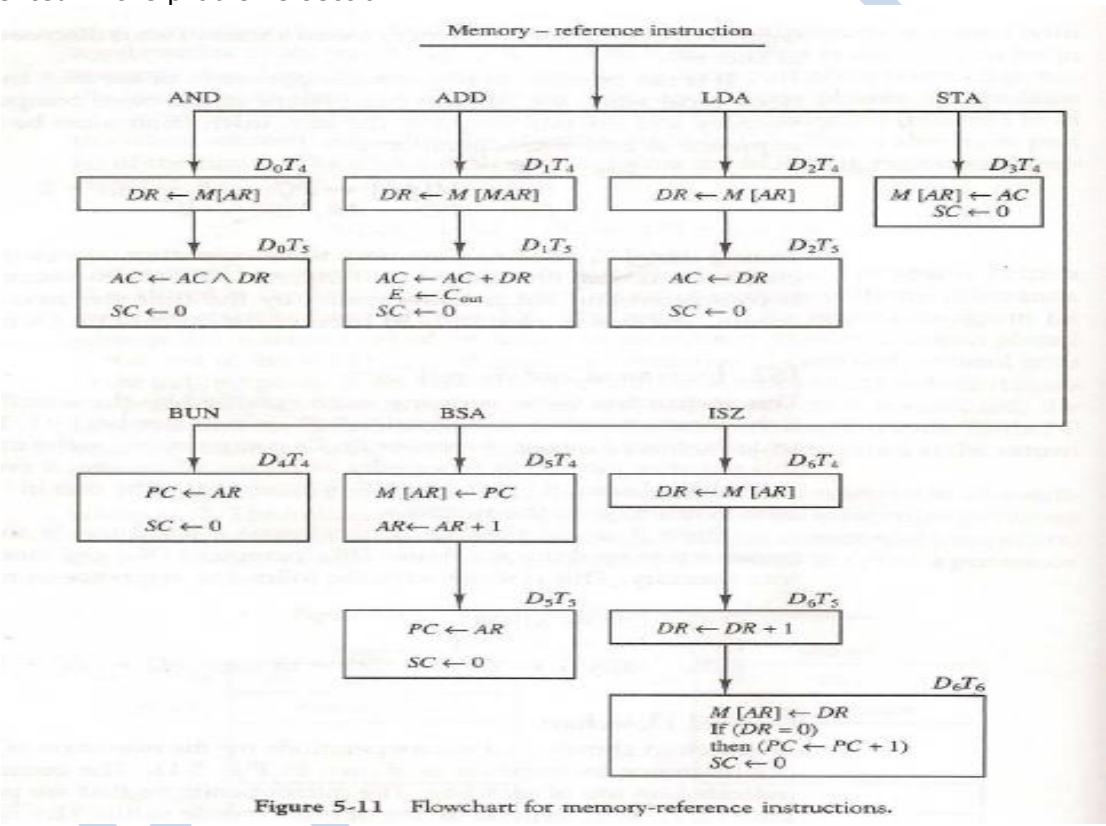


Figure 5-11 Flowchart for memory-reference instructions.

2.7 Input-Output and Interrupt.

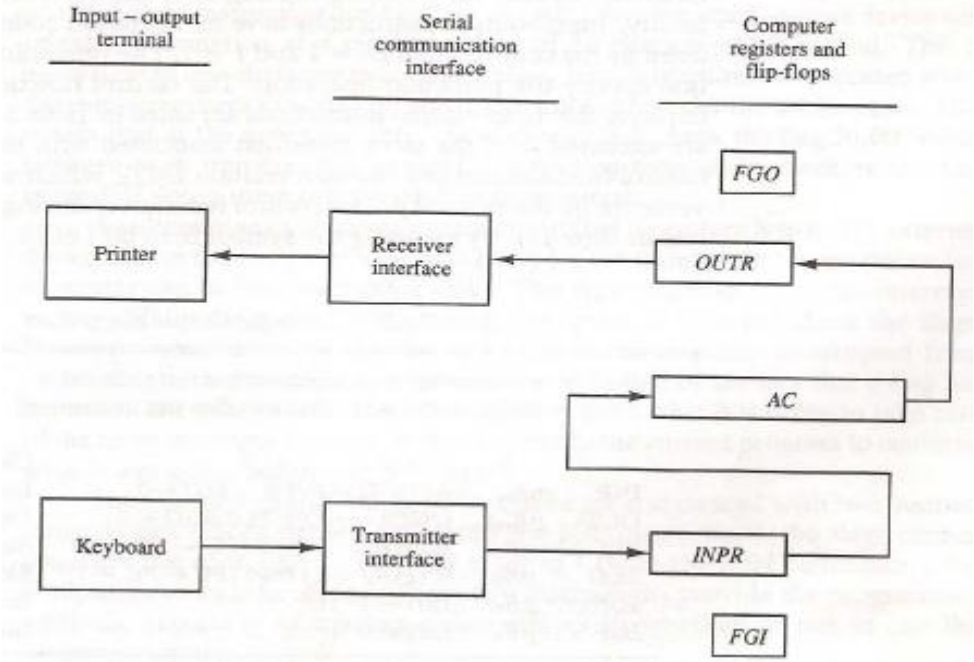
A computer can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device. Commercial computers include many types of input and output devices. To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer.

Input-Output Configuration

The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code. The serial information from the keyboard is shifted into the input register INPR. The serial information for the printer is stored in the output register OUTR. These two registers communicate with a communication interface serially and with the AC in parallel. The input-output configuration is shown in Fig. 5-12. The transmitter interface receives serial information from the keyboard and transmits it to

INPR. The receiver interface receives information from OUTR and sends it to the printer serially.

Figure 5-12 Input-output configuration.



The input register INPR consists of eight bits and holds an alphanumeric input information. The 1-bit input flag PGI is a control flip-flop. The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer. The flag is needed to synchronize the timing rate difference between the input device and the computer. The process of information transfer is as follows. Initially, the input flag PGI is cleared to 0. When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag PGI is set to 1. As long as the flag is set, the information in INPR cannot be changed by striking another key. The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and PGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key.

The output register OUTR works similarly but the direction of information flow is reversed. Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to O. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1. The computer does not load a new character into OUT R when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

Input-Output Instructions

Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility. Input-output instructions have an operation code 1111 and are recognized by the control when D7 = 1 and I = 1. The remaining bits of the instruction specify the particular operation. The control functions and microoperations for the input-output instructions are listed in Table 5-5.

TABLE 5-5 Input-Output Instructions

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

Program Interrupt

In programmed control transfer. The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer. The difference of information flow rate between the computer and that of the input-output device makes this type of transfer inefficient. To see why this is inefficient, consider a computer that can go through an instruction cycle in 1 μs. Assume that the input-output device can transfer information at a maximum rate of 10 characters per second. This is equivalent to one character every 100,000 μs. Two instructions are executed when the computer checks the flag bit and decides not to transfer the information. This means that at the maximum rate, the computer will check the flag 50,000 times between each transfer. The computer is wasting time while checking the flag instead of doing some other useful processing task.

An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer. In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility. While the computer is running a program, it does not check the flags. However, when a flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that a flag has been set. The computer deviates momentarily from what it is doing to take care of the input or output transfer. It then returns to the current program to continue what it was doing before the interrupt.

The interrupt enable flip-flop IEN can be set and cleared with two instructions. When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer. When IEN is set to 1 (with the ION instruction), the computer can be interrupted. These two instructions provide the programmer with the capability of making a decision as to whether or not to use the interrupt facility.

The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig. 5-13. An interrupt flip-flop R is included in the computer. When R = 0, the computer goes through an instruction cycle. During the execute phase of the instruction cycle IEN is checked by the control. If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle. If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flap R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

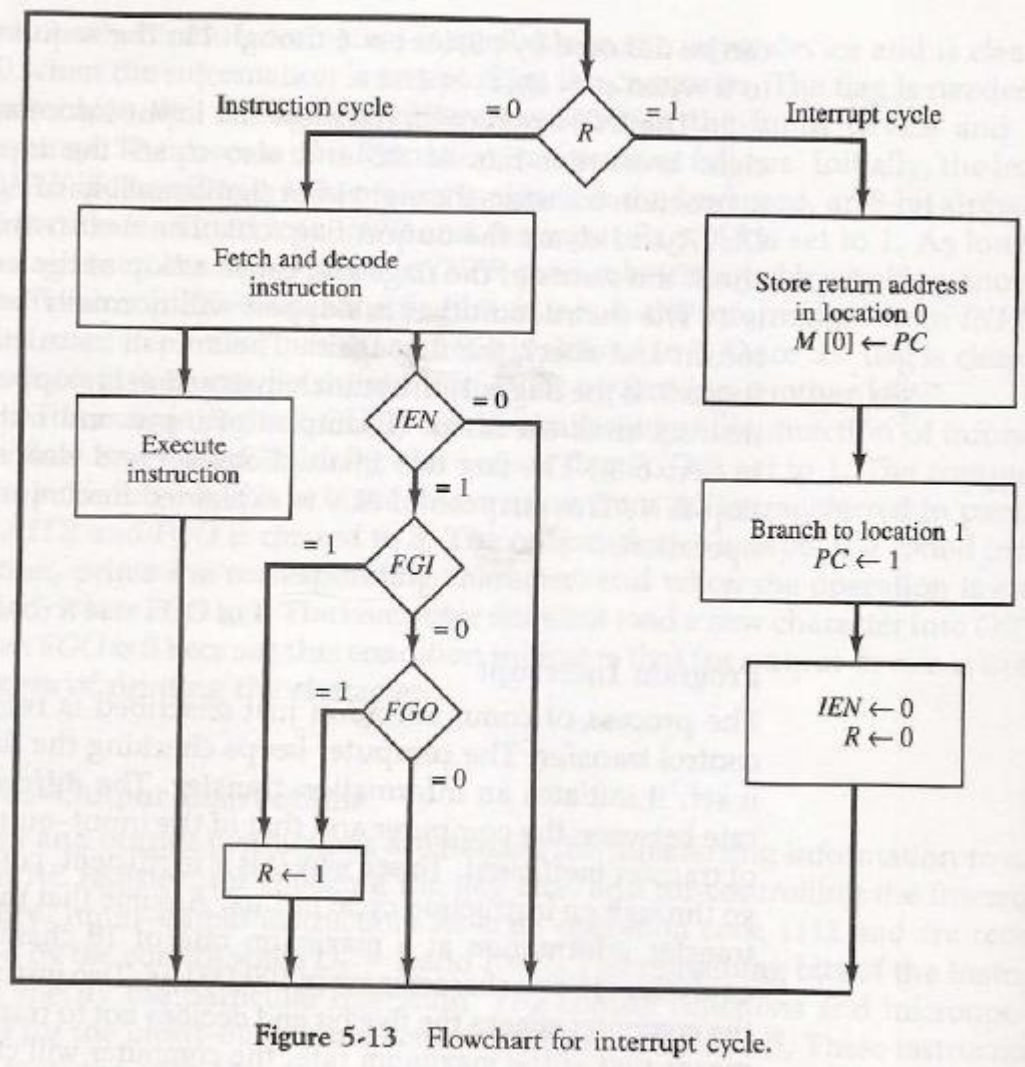


Figure 5-13 Flowchart for interrupt cycle.

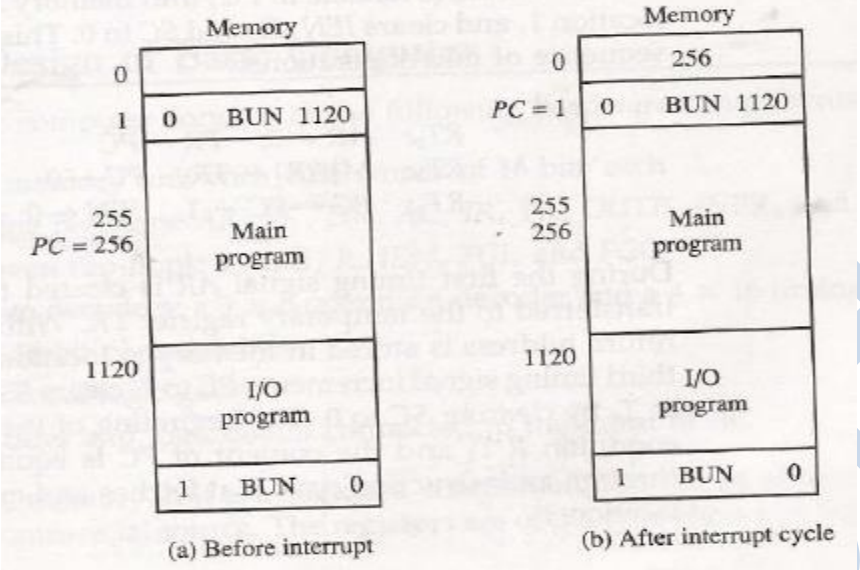
The interrupt cycle is a hardware implementation of a branch and save return address operation. The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor register, a memory stack, or a specific memory location. Here we choose the memory location at address 0 as the place for storing the return address. Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.

An example that shows what happens during the interrupt cycle is shown in Fig. 5-14. Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255. At this time, the return address 256 is in PC. The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Fig. 5-14(a).

When control reaches timing signal To and finds that R = 1, it proceeds with the interrupt cycle. The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120. This program checks the flags, determines which flag is set, and then transfers the required input or output information. Once this is done, the instruction ION is executed to

set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted. This is shown in Fig. 5-14(b).

Figure 5-14 Demonstration of the interrupt cycle.



The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0. This instruct tion is placed at the end of the I/0 service program. After this instruction is read from memory during the fetch phase, control goes to the indirect phase (because I = 1) to read the effective address. The effective address is in location 0 and is the return address that was stored there during the previous interrupt cycle. The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

Interrupt Cycle

We are now ready to list the register transfer statements for the interrupt cycle. The interrupt cycle is initiated after the last execute phase if the interrupt flip-flop R is equal to 1. This flip-flop is set to 1 if IEN = 1 and either PGI or FGO are equal to 1. This can happen with any clock transition except when timing signals To, T1, or T2 are active. The condition for setting flip-flop R to 1 can be expressed with the following register transfer statement:

$T_6T_1T_5$(IEN)(FGI + PGO): R ←1

The symbol + between FGI and FGO in the control function designates a logic OR operation. This is ANDed with IEN and $T'_0T'_1T'_2$.

MICRO-PROGRAMMED CONTROL: Control Memory, Address Sequencing, Micro-program Example, Design of Control Unit, Hardwired Control, Micro-programmed Control, Nanoprogramming.

2.8 Control Memory

Introduction

- The function of a control unit in a digital computer is to initiate sequences of finite micro-operations
- The complexity of any digital system is related to the number of microoperations performed.

2 types of control units:

1. "Hardwired control unit" is implemented using conventional logic design techniques.
2. "Microprogramming" is another implementation for building control units.

- The control unit initiates a series of sequential microoperations. During a given time certain microoperations needed to be initiated while others remain idle.

Control word

- is a series of 0's and 1's that can be programmed to perform different operations on components of the system

Now

- The control unit whose binary control words are stored in memory is called "microprogrammed control unit". Such that each word in control memory contains a number of microinstructions
- Microinstruction specifies one or more microoperations of the system

Microprogram

- Program stored in memory that generates all the control signals required to execute the instruction set correctly
- Consists of microinstructions

Microinstruction

- Contains a control word and a sequencing word
  - Control Word -All the control information required for one clock cycle
  - Sequencing Word -Information needed to decide the next microinstruction address

Control Memory (Control Storage: CS)

- Storage in the microprogrammed control unit to store the microprogram
- Holds fixed microprogram that cannot be altered by user
- Microprogram consists of microinstructions that specifies internal control signals for register microoperations
- Microinstruction generates microoperations to fetch instructions from main memory, evaluate effective address, execute the operation, and return control to fetch phase again to start new cycle.

The general configuration of a microprogrammed control unit is demonstrated in the block diagram of Fig. 7-1. The control memory is assumed to bea ROM, within which all control information is permanently stored.

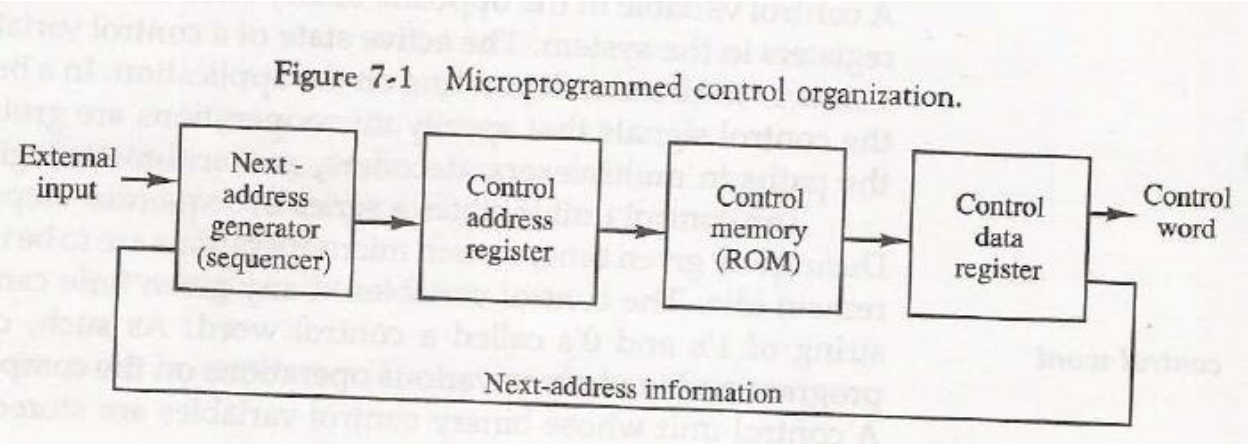


Figure 7-1 Microprogrammed control organization.

Control Address Register

- Specifies address of microinstructions, and control data register holds microinstruction
- Microinstruction contains control word that specifies one or more microoperations
- Once these operations are executed, control must determines next address
- Microinstructions contain bits for initiating microoperations and bits that determines the address sequence for the control memory

Sequencer

- Next address generator is called microprogram sequencer
- Will do the following jobs which can be called next address calculation that comes from different sources:
    - In-line Sequencing
    - Branch
    - Conditional Branch
    - Subroutine
    - Loop
    - Instruction OP-code mapping

Pipeline register

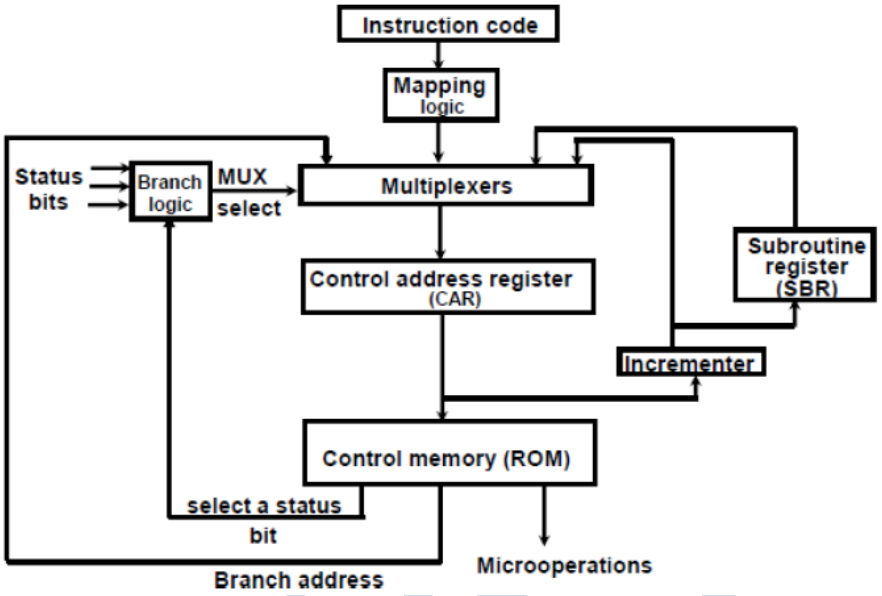
The control data register holds the present microinstruction while thenext address is computed and read from memory. The data register is some pipeline register times called a pipeline register. It allows the execution of the microoperationsspecified by the control word simultaneously with the generation of the nextmicroinstruction. This configuration requires a two-phase clock, with one clockapplied to the address register and the other to the data register.

2.9 Address Sequencing

- We need a transformation from instruction code bits into address in control memory. Then the microinstruction that executes the instruction may be sequenced by incrementing the control address register (CAR).
- Sometimes the sequence of microoperations depend on values of certain status bits.
- Address sequence capability required in control memory are:
    - Incrementing CAR
    - Unconditional-Conditional branching depending on status bits

- Mapping from bits of instruction to address in control memory
- Subroutine call-return facility

- See next Figure that shoes control memory and how next address is selected

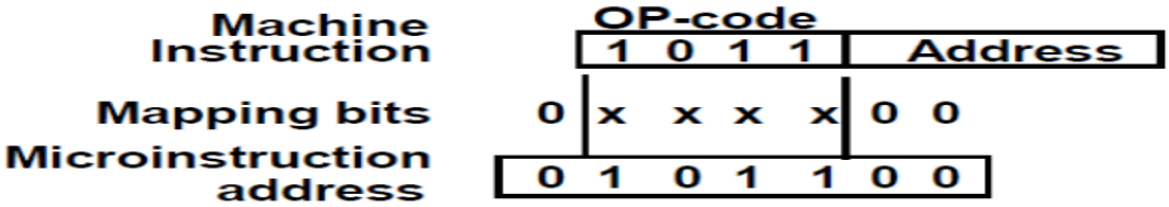


Conditional Branching

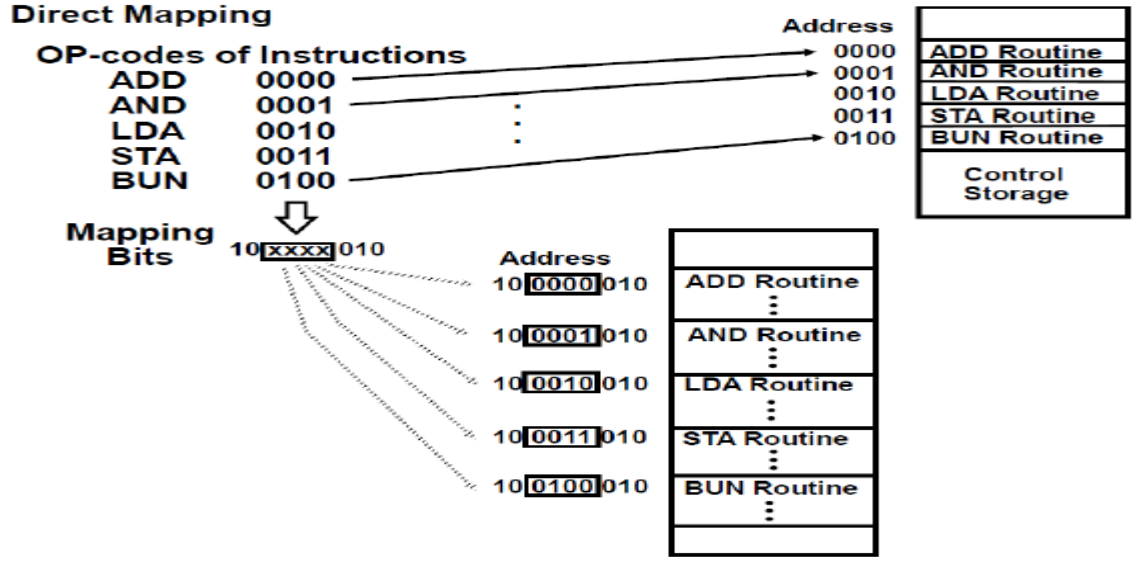
- Branch logic of above Figure provides decision making capability in control unit.
- Status conditions provide parameter data such as carry bit, sign bit, mode bit, and IO status bits. Those can be tested as 0/1
- In branch logic circuit, the specified condition is tested and a branch to indicated address is carried on if condition is true; otherwise, ACR is incremented.
- Conditions to Test: O(overflow), N(negative), Z(zero), C(carry), etc.
- In Unconditional Branch, the value of one status bit at the input of the multiplexer is fixed to 1, to ensure always branching.

Instruction Mapping

- We assume an operation code of 4 bits that can specify up to 16 distinct instructions. Assume control memory has 128 words which requires 7 address bits
- For each operation code we can reach to an address in control memory where we can start executing microinstruction of a routine
- A simple mapping converts the 4 bits in op code to 7 bits address for control memory as shown next
- This mapping places 0 in the most significant bit of the address, transferring the 4 bits for op-code, and clearing the 2 least significant bits of CAR.
- This will give a micro routine with capacity of 4 microinstructions



MAPPING OF INSTRUCTIONS



- Mapping can be implemented using ROM concept: the bits of the instruction specify the address of ROM and the content (data) of ROM gives the address of CAR. By this way the Microprogram routine can be placed in any location in control memory.
- Mapping sometimes can be implemented by programmable logic devices or PLD. The mapping function can be expressed in terms of Boolean expression that easily be implanted conveniently with PLD.

Subroutines

Subroutines are programs that are used by other routines to accomplish aparticular task. A subroutine can be called from any point within the main bodyof the microprogram. Frequently, many microprograms contain identical sections of code. Microinstructions can be saved by employing subroutines thatuse common sections of microcode. For example, the sequence of microoperations needed to generate the effective address of the operand for an instruction is common to all memory reference instructions. This sequence could bea subroutine that is called from within many other routines to execute theeffective address computation.

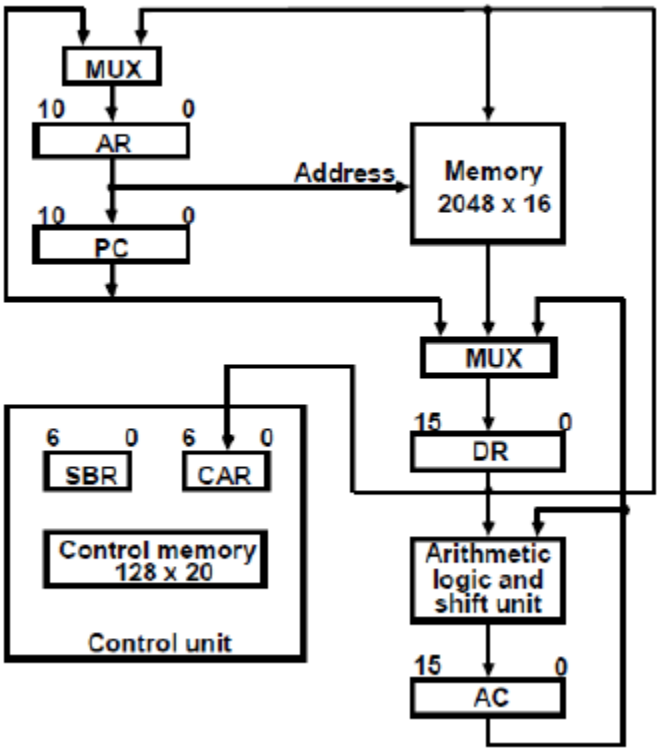
Microprograms that use subroutines must have a provision for storing thereturn address during a subroutine call and restoring the address during asubroutine return. This may be accomplished by placing the incrementedsubroutine register output from the control address register into a subroutine register and branching to the beginning of the subroutine. The subroutine register can then become the source for transferring the address for the return to the mainroutine. The best way to structure a register file that stores addresses forsubroutines is to organize the registers in a last-in, first-out (LIFO) stack.

Microprogram Example

- Next Figure shows 2 memory units: main unit for storing instructions and data, and a control memory for storing microprogram

- 4 registers associated with processor unit (PC, AD, DR, and AC) and 2 associated withcontrol memory (CAR control address register and SBR subroutine register).
- Multiplexers are used to transfer data between registers rather than a bus.
- DR receives data from AC, PC, or memory
- AR receives data from PC or DR
- PC receives data from AR only
- ALU and shift unit perform its function on AC and DR and places results in AC
- Memory address always comes from AR, Data written to memory comes from DR, and data read from memory goes to DR

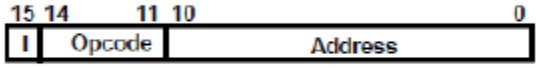
Computer Configuration



Instruction format

- A next Figure shows, it consists of indirect bit, 4 op-code bits, and 11 address bits .

Machine instruction format



- Next Figure lists 4 out of 16 possible memory reference instructions

Sample machine instructions

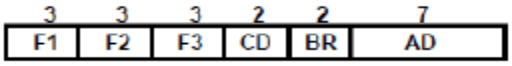
Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	If (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

Microinstruction Format

- Microinstruction format of control memory is shown in next Figure. It shows 20 bit of microinstruction divided to:
  - F1, F2, F3 that specify 3 microoperations executed in same clock cycle
  - CD field for specifying branch status bit condition
  - BR field that determines type of branch.
  - AD which is 7-bit address field that can a location of 128 different addresses

Microinstruction Format



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

Microinstruction Field F1-F2-F3

- The 3 bits field encoded to specify one of 7 distinct microoperation
- If fewer than 3 microinstruction is needed then unneeded field(s) will be encoded with "000"
- Not allowed to encode 2 or more conflicting operations

Condition Field CD

- Consists of 2 bits encoded to specify 4 status bit condition as shown in next figure.
  - First condition is always "1", CD = "00"
  - I bit, CD = "01"
  - Z bit (of AC register), CD = "11"
  - S bit (MSB of AC register), CD = "10"

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

Symbols and Binary Code for microinstruction Fields.

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	

Branch Field BR

- Consists of 2 bits and it is used with conjunction of address bits AD to decide and choose address of next microinstruction
  - When BR = "00" then control performs JMP
  - When BR = "01" it performs a CALL to a subroutine. Stores return address in SBR register
  - When BR = "10" it performs RET operation. Moves return address from SBR to CAR
  - When BR = "11" it performs mapping from operation code bits of instruction into CAR address

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

Symbolic Microinstructions

- Each line of assembly line Microprogram defines a symbolic microinstruction. Each symbolic microinstruction is divided into 5 fields:

label, microoperations, CD, BR, and AD. The fields specify the followinginformation.

1. The label field may be empty or it may specify a symbolic address. Alabel is terminated with a colon (z).
2. The microoperations field consists of one, two, or three symbols, separated by commas. There may be nomore than one symbol from each F field. The NOP symbol is used when themicroinstruction has no microoperations. This will be translated bythe assembler to nine zeros.
3. The CD field has one of the letters U, I, S, or Z.
4. The BR field contains one of the four symbols address field

5. The AD field specifies a value for the address field of the microinstruction in one of three possible ways:

a. With a symbolic address, which must also appear as a label.

b. With the symbol NEXT to designate the next address in sequence.

c. When the BR field contains a RET or MAP symbol, the AD field isleft empty and is converted to seven zeros by the assembler.

ORG

ORG is used to define the origin, or firstaddress, of a micr0program routine. Thus the symbol ORG 64 informs theassembler to place the next microinstruction in control memory at decimaladdress 64, which is equivalent to the binary address 1000000.

The Fetch Routine

- Control memory has 128 locations each with 20 bits inside
- The first 64 words(0 to 63) are dedicated for routines for the 16 macro instructions (16 * 4)
- The last 64 words can be used for other functions
- From location 64 we will find routine for fetch cycle

Fetch and Decode Routine

- During a fetch routine, an instruction is read from memory, decoded, and PC is updated
- See next figure to identify microinstructions for fetch cycle.

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 execution of the third MAP microinstruction in the fetch routine results in a branch to address 0XXXX00, where XXXX are the four bits of operation code. This gives 4 words in control memory for each routine

Indirect Routine

- We need to calculate the indirect procedure for accessing the address of operand in all memory-reference instructions

- So INDRCT routine has been placed in a subroutine that can be called in every memory-reference instruction
- It is only can be called if I=1 then a branch to NDRCT occurs (See Table)
- So in this cycle memory is accessed to get the effective address of operand

Execution of ADD instructions

- For ADD instruction, the microinstructions in locations 1 and 2 will carry out.
- Reads operand from memory into DR
- Add contents of DR to AC
- Jumps to fetch routine

Execution of BRANCH instructions

- It causes a branch to effective address of instruction if AC < 0 which means S = 1
- Starts by checking the value of S. if S=0 then no branch occurs and next microinstruction causes a jump to fetch routine
- If S=1 then control goes to location OVER where a call to INDRCT takes place first if I=1 then control goes to where AR register points to

Execution of STORE instructions

- Uses INDRCT routine if I=1
- Content of AC is transferred to DR then memory write operation is initiated to store DR into MEM(AR)

Execution of EXCHANGE instructions

- Exchange content of MEM(AR) with AC
- Jumps to INDRCT routine to get the effective operand address
- Reads from memory the operand and direct it to DR
- AC and DR are exchanged in third microinstruction
- DR is written to memory MEM(AR)

Next Figure shows partial symbolic microprogram for selected instructions

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	

2.10 Design of Control Unit:

The number of control bits that initiate microoperations can be reduced by grouping mutually exclusive variables into fields and encode k bits into 2k microoperations

F-fields decoding

- Next figure shows 3 decoders of 3X8 type, so each field gives 8control lines
- Each of these 24 outputs will be connected into proper circuit to initiated microoperations as specified in Table 7.1.

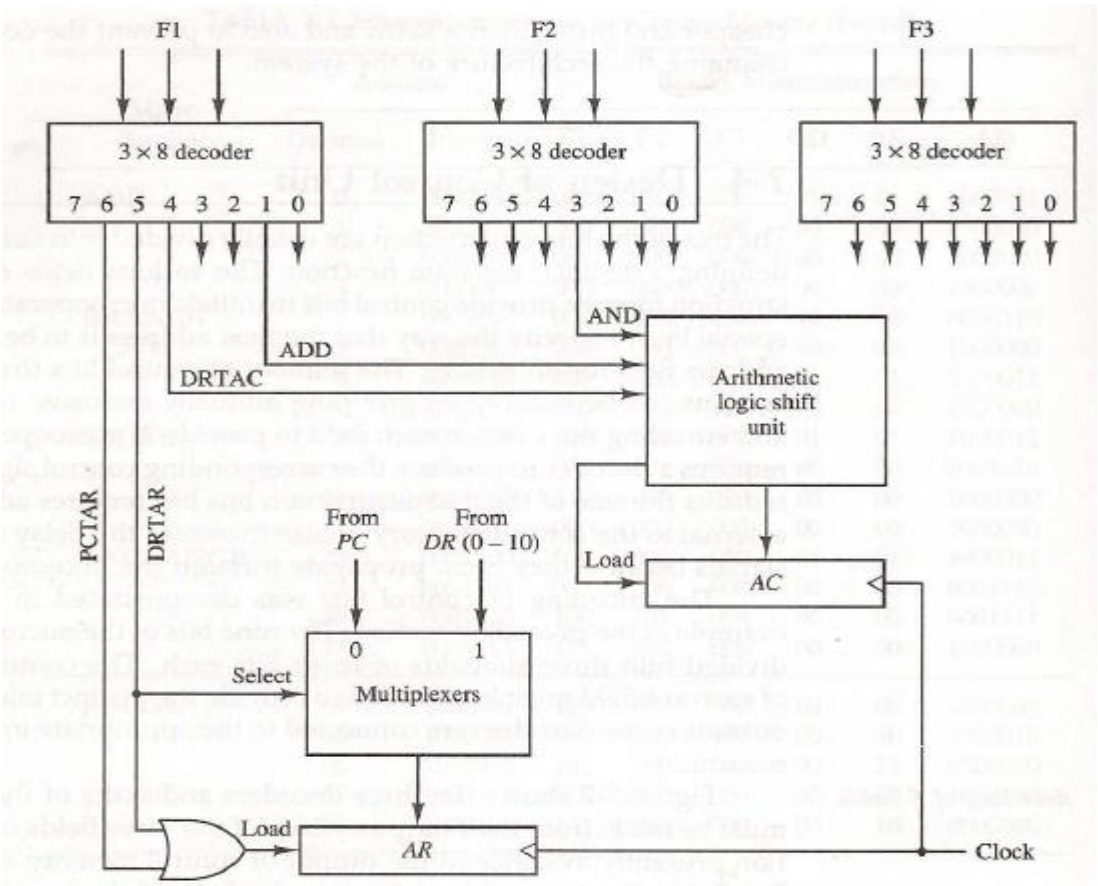


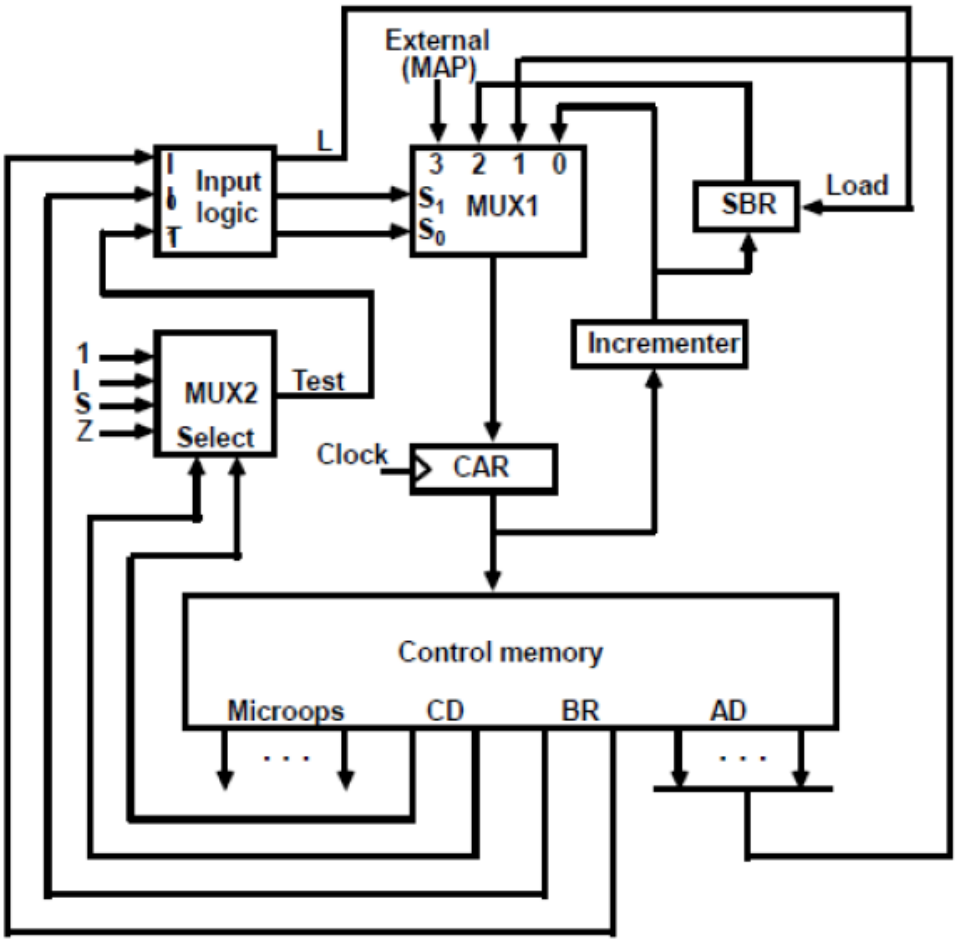
Figure 7-7 Decoding of microoperation fields.

Arithmetic-Logic-shift Unit

Instead of using gates to generate the control signals marked by the symbols AND, ADD, and DR in Fig. 5-19, these inputs will now come from the outputs of the decoders associated with the symbols AND, ADD, and DRTAC, respectively. 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.

Microprogram Sequencer

- Address selection part is called microprogram sequencer
- The purpose of the sequencer is to present address of control memory so to fetch next microinstruction to be executed
- The choice of address source is controlled by next address information bits from present microinstruction
- Next figure shows a microprogram sequencer for control memory
    - 2 multiplexers, one to select address from one of four address sources to CAR, and the second for testing value for selected status bit
    - CAR always provide address to control memory



The input logic circuit in Fig. 7-8 has three inputs, I0, I1, and T, and threeoutputs, So, S1, and L. Variables So and 51 select one of the source addressesfor CAR. Variable L enables the load input in SBR. The binary values of the twoselection variables determine the path in the multiplexer. For example, withS1 So = 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 outputof MUX 1 contains a 7-bit address.

The truth table for the input logic circuit is shown in Table 7-4. Inputs 11and Io are identical to the bit values in the BR field. The function listed in eachentry was defined in Table 7-1. The bit values for S1 and So are determined fromthe stated function and the path in the multiplexer that establishes the requiredtransfer. The subroutine register is loaded with the incremented value of CARduring a call microinstruction (BR = 01) provided that the status bit conditionis satisfied (T = 1). The truth table can be used to obtain the simplified Booleanfunctions for the input logic circuit:

$$S_1 = I_1$$

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

$$L = I_1' I_0 T$$

TABLE 7-4 Input Logic Truth Table for Microprogram Sequencer

BR Field	Input			MUX 1		Load *SBR*
	I_1	I_0	T	S_1	S_0	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 aninverter.

2.11 Hardwired Control

The control units use fixed logic circuits to interpret instructions and generate control signals from them.

The fixed logic circuit block includes combinational circuit that generates the required control outputs for decoding and encoding functions.

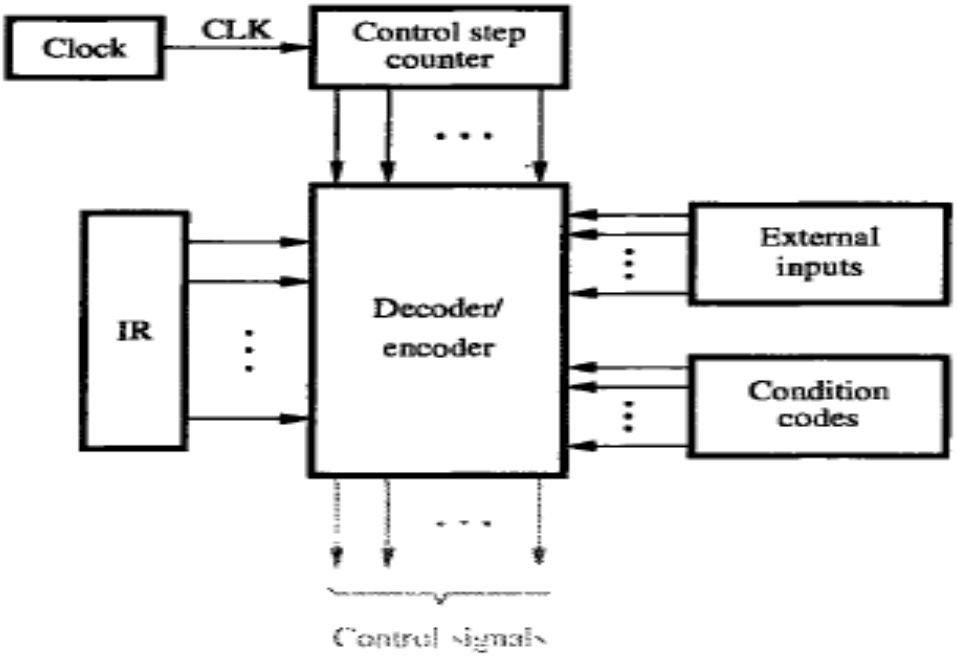


Figure 7.10 Control unit organization.

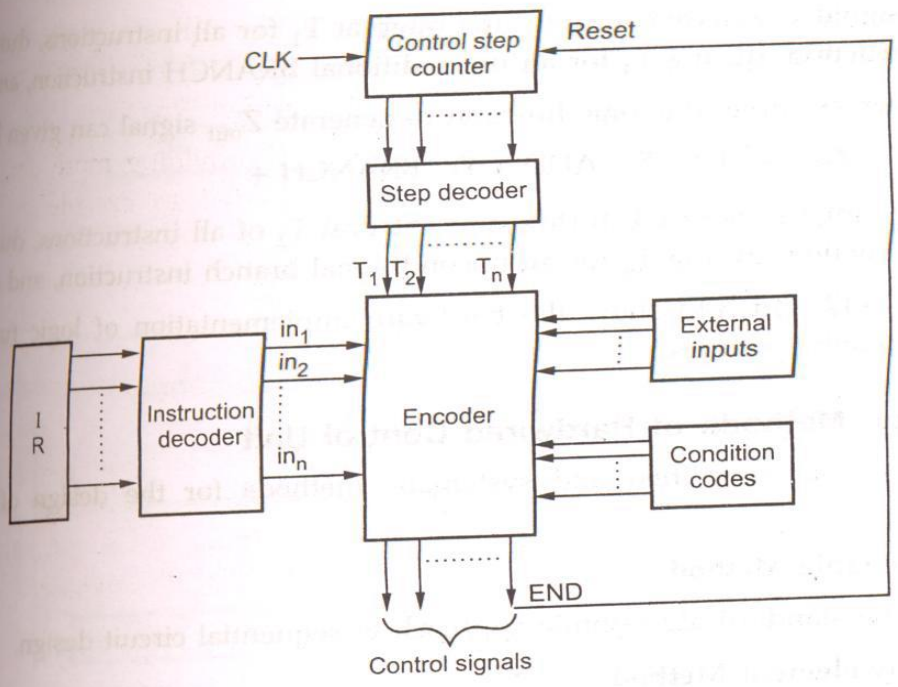
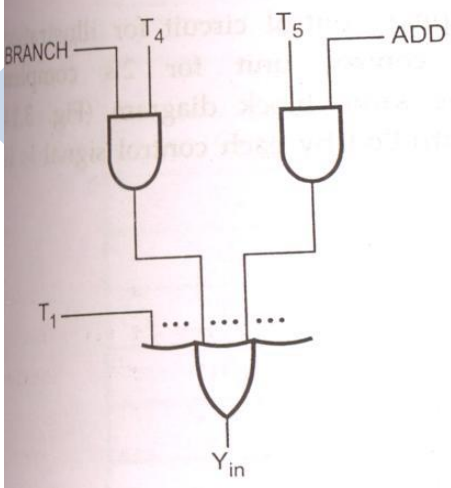


Fig. 3.11 Detail block diagram for hardwired control unit

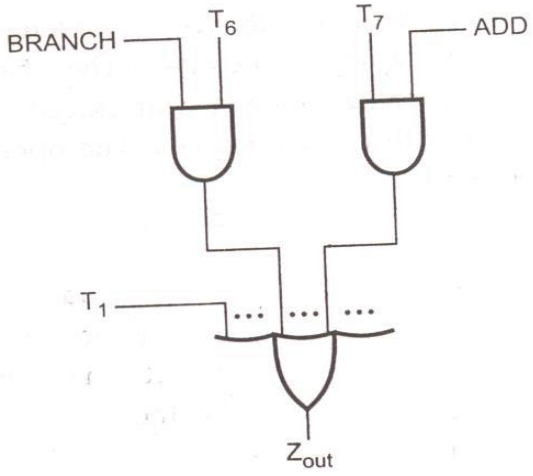
Let us see how the encoder generates signal for single bus processor organisation wn in Fig. 3.12 Y_{in}. The encoder circuit implements the following logic function to erate Y_{in}.

$$Y_{in} = T_1 + T_5 \cdot ADD + T_4 \cdot BRANCH + \ldots$$



Generation of the Y_{in} control signal

Fig. 3.12



Generation of the Z_{out} control signal

Fig. 3.13

Instruction decoder
It decodes the instruction loaded in the IR.
If IR is an 8 bit register then instruction decoder generates 28(256 lines); one for each instruction.
According to code in the IR, only one line amongst all output lines of decoder goes high (set to 1 and all other lines are set to 0).
Step decoder
It provides a separate signal line for each step, or time slot, in a control sequence.
Encoder
It gets in the input from instruction decoder, step decoder, external inputs and condition codes.
It uses all these inputs to generate the individual control signals.
After execution of each instruction end signal is generated this resets control step counter and make it ready for generation of control step for next instruction.
The encoder circuit implements the following logic function to generate Yin
Yin = T1 + T5 . Add + T . BRANCH+…
The Yin signal is asserted during time interval T1 for all instructions, during T5 for an ADD instruction, during T4 for an unconditional branch instruction, and so on.
As another example, the logic function to generate Zout signal can given by
Zout = T2 + T7 . ADD + T6 . BRANCH +….
The Zout signal is asserted during time interval T2 of all instructions, during T7 for an ADD instruction, during T6 for an unconditional branch instruction, and so on.
A Complete processor
It consists of

- Instruction unit
- Integer unit
- Floating-point unit
- Instruction cache
- Data cache
- Bus interface unit
- Main memory module
- Input/Output module.

Instruction unit- It fetches instructions from an instruction cache or from the main memory when the desired instructions are not available in the cache.
Integer unit – To process integer data
Floating unit – To process floating –point data
Data cache – The integer and floating unit gets data from data cache
The 80486 processor has 8-kbytes single cache for both instruction and data whereas the Pentium processor has two separate 8 kbytes caches for instruction and data.

The processor provides bus interface unit to control the interface of processor to system bus, main memory module and input/output module.

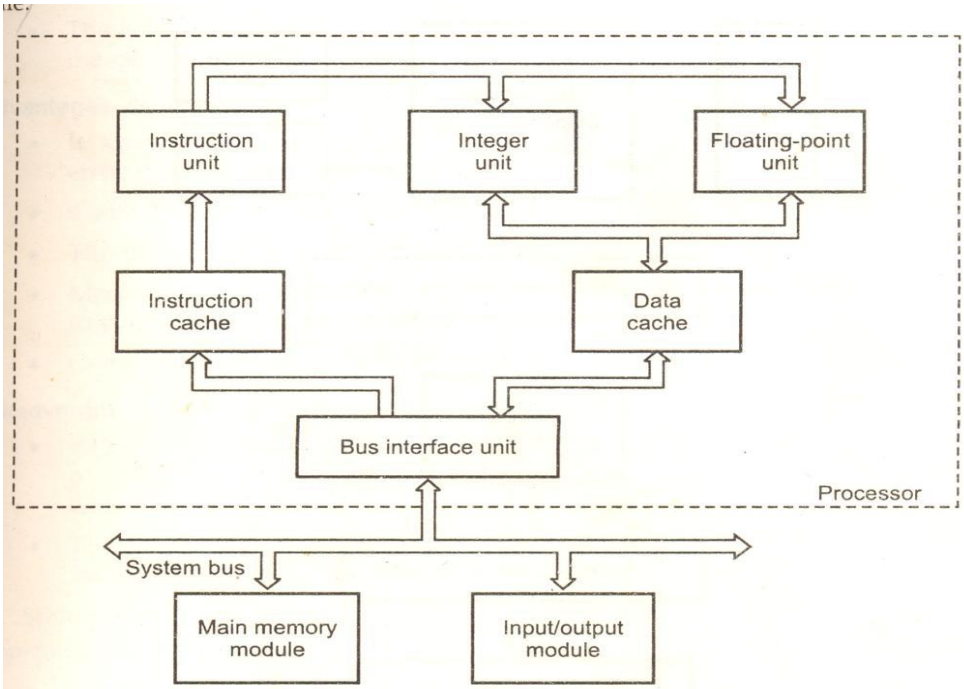


Fig. 3.14 Block diagram of a complete processor

2.12 Microprogrammed Control

- Every instruction in a processor is implemented by a sequence of one or more sets of concurrent micro operations.
- Each micro operation is associated with a specific set of control lines which, when activated, causes that micro operation to take place.
- Since the number of instructions and control lines is often in the hundreds, the complexity of hardwired control unit is very high.
- Thus, it is costly and difficult to design. The hardwired control unit is relatively inflexible because it is difficult to change the design, if one wishes to correct design error or modify the instruction set.
- Microprogramming is a method of control unit design in which the control signal memory CM.
- The control signals to be activated at any time are specified by a microinstruction, which is fetched from CM.
    - A sequence of one or more micro operations designed to control specific operation, such as addition, multiplication is called a micro program.
    - The micro programs for all instructions are stored in the control memory.

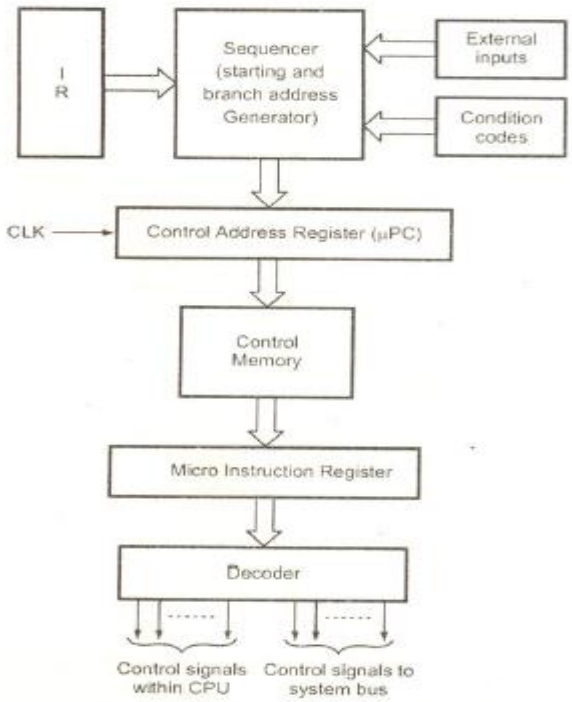


Fig. 3.15 Microprogrammed control unit

- The address where these microinstructions are stored in CM is generated by microprogram sequencer/microprogram controller.
- The microprogram sequencer generates the address for microinstruction according to the instruction stored in the IR.
- The microprogrammed control unit,
  - control memory
  - control address register
  - micro instruction register
  - microprogram sequencer
  - The components of control unit work together as follows:
- The control address register holds the address of the next microinstruction to be read.
- When address is available in control address register, the sequencer issues READ command to the control memory.
- After issue of READ command, the word from the addressed location is read into the microinstruction register.
- Now the content of the micro instruction register generates control signals and next address information for the sequencer.
- The sequencer loads a new address into the control address register based on the next address information.

Advantages of Microprogrammed control

- It simplifies the design of control unit. Thus it is both, cheaper and less error phrone implement.
- Control functions are implemented in software rather than hardware.
- The design process is orderly and systematic
- More flexible, can be changed to accommodate new system specifications or to correct the design errors quickly and cheaply.

- Complex function such as floating point arithmetic can be realized efficiently.

Disadvantages

- A microprogrammed control unit is somewhat slower than the hardwired control unit, because time is required to access the microinstructions from CM.
- The flexibility is achieved at some extra hardware cost due to the control memory and its access circuitry.

Comparison between Hardwired and Microprogrammed Control

Attribute	Hardwired Control	Microprogrammed Control
Speed	Fast	Slow
Control functions	Implemented in hardware	Implemented in software
Flexibility	Not flexible to accommodate new system specifications or new instructions	More flexible, to accommodate new system specification or new instructions redesign is required
Ability to handle large/complex instruction sets	Difficult	Easier
Ability to support operating systems and diagnostic features	Very difficult	Easy
Design process	Complicated	Orderly and systematic
Applications	Mostly RISC microprocessors	Mainframes, some microprocessors
Instructionset size	Usually under 100 instructions	Usually over 100 instructions
ROM size	-	2K to 10K by 20-400 bit microinstructions
Chip area efficiency	Uses least area	Uses more area