Subject title: Allied 4: COMPUTER SYSTEM ARCHITECTURE

YEAR 2018-19 ONWARDS SEMESTER: IV

SUBJECT CODE: 18BIT44A

UNIT II: Basic computer organization and design: instruction codes-timing and control-computer register – instruction cycle – input – output and interrupt –Micro programmed control: control memory – address sequencing – design of control unit.

Prepared by : Mrs. P.Sundari

Instruction Codes

The internal organization of a digital system is defined by the sequence of micro operations it performs on data stored in its registers.

A computer instruction is a binary code that specifies a sequence of micro operations 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.

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 part. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.

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.

Stored program organization

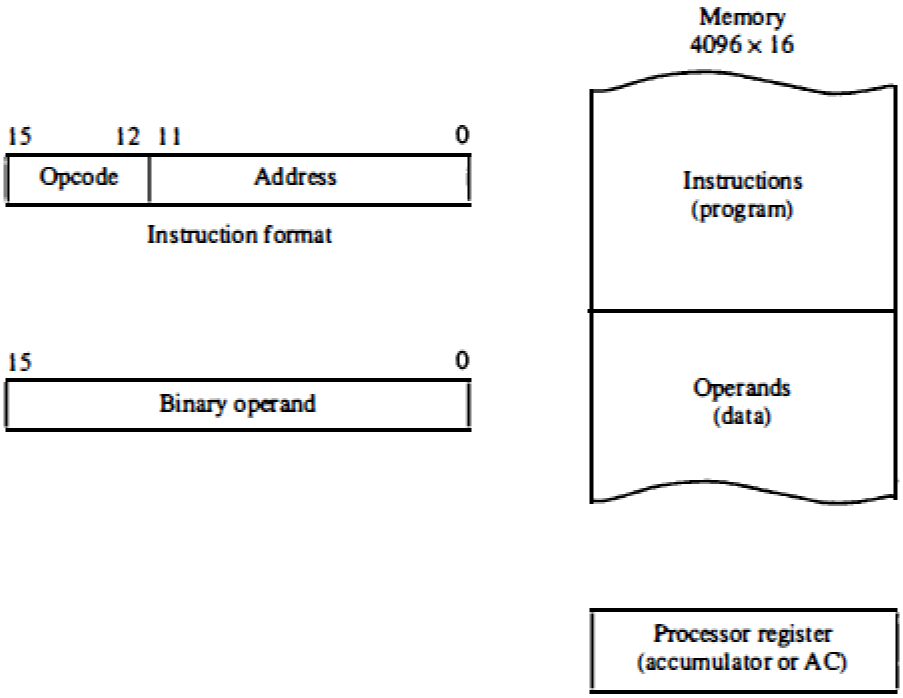


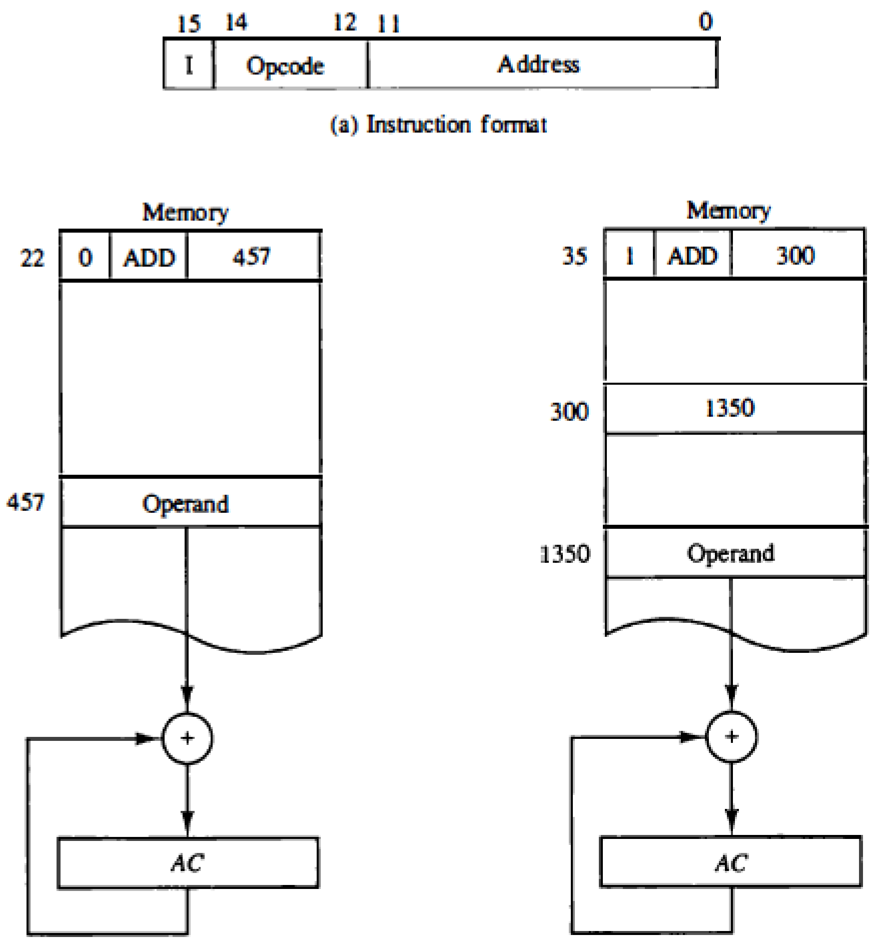
Figure depicts this type of organization. Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits to specify an address since 212 = 4096. If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated op code) 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 th operation code.

Computers that have a single-processor register usually assign to it the name 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 code specifies an operand, the instruction is said to have an immediate 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.



(a) Instruction format

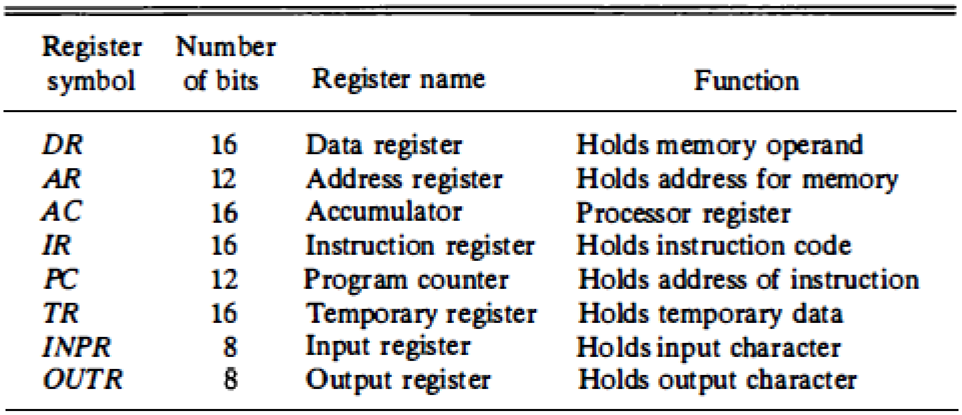
(b) Direct address

(c) Indirect address

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. The computer needs processor registers for manipulating data and a register for holding a memory address.

List of registers



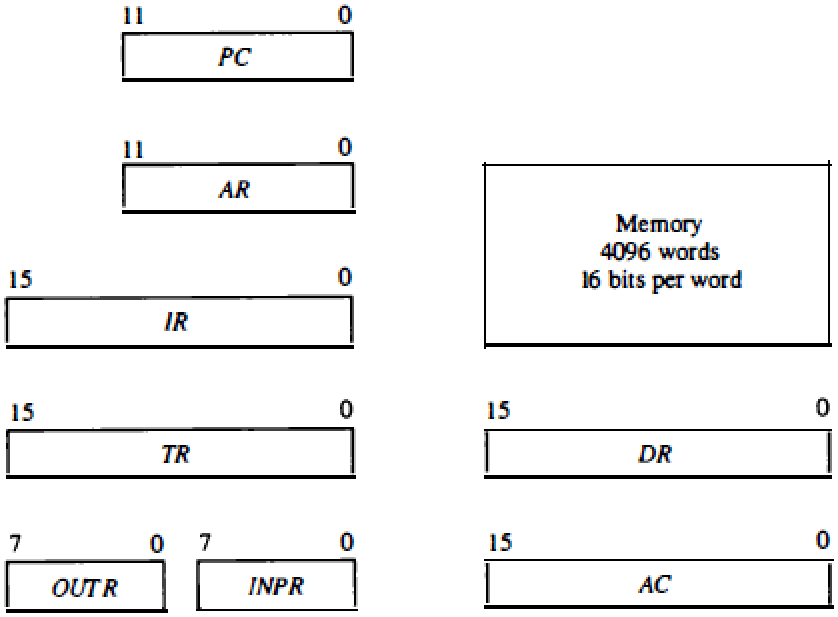
Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register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

The memory address register (AR) has 12 bits since this is the width of a memory address. The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed. The PC goes through a counting sequence and causes the computer to read sequential instructions previously stored in memory.

Instruction words are read and executed in sequence unless a branch instruction is encountered. A branch instruction calls for a transfer to a nonconsecutive instruction in the program. The address part of a branch instruction is transferred to PC to become the address of the next instruction. To read an instruction, the content of PC is taken as the address for memory and a memory read cycle is initiated. PC is then incremented by one, so it holds the address of the next instruction in sequence.

Two registers are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device.

Computer registers and Memory



Timing and Control

The timing for all registers in the basic computer is controlled by a master clock 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 micro operations for the accumulator.

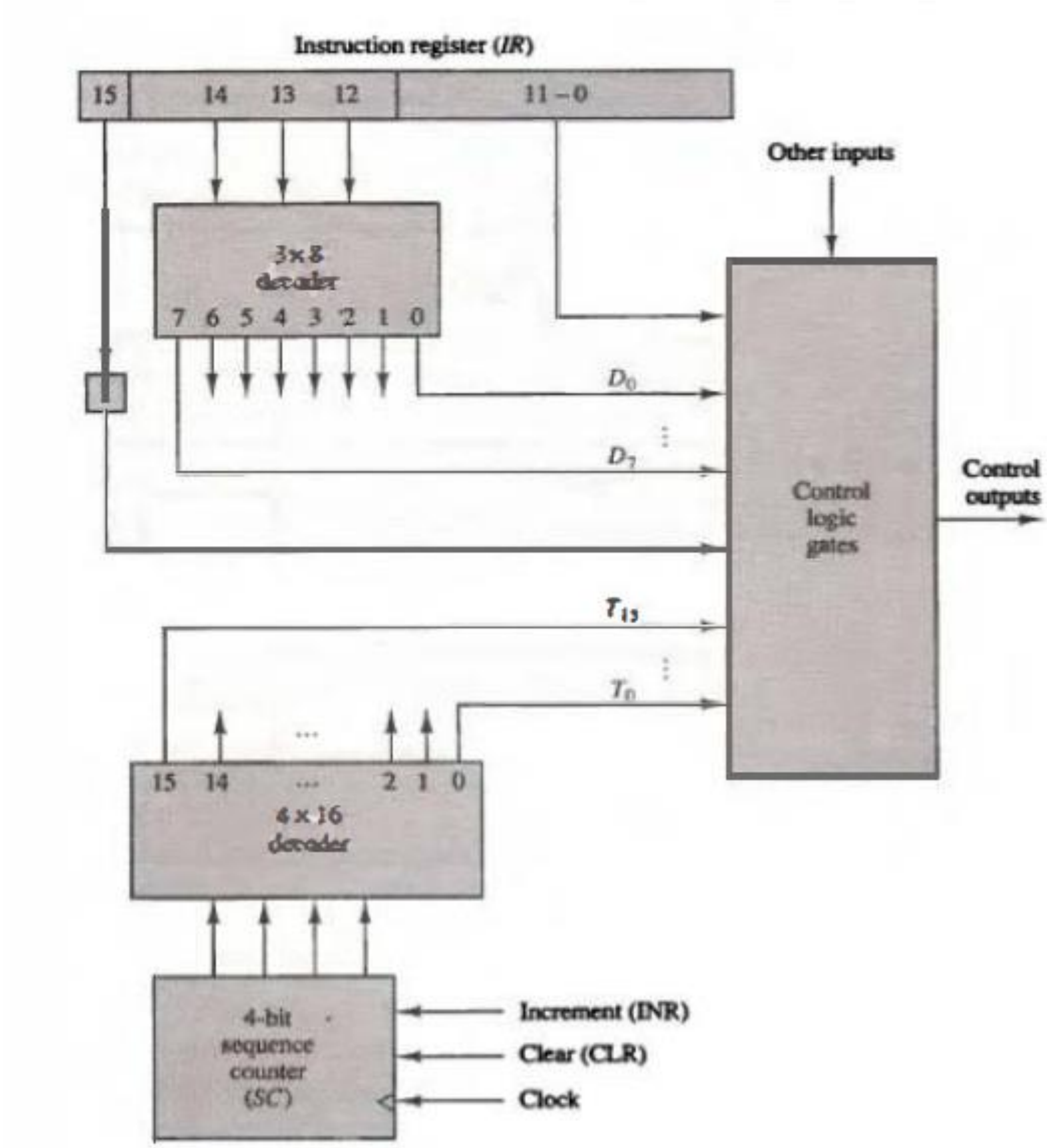
There are two major types of control organization: hardwired control and micro programmed control. In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.

It has the advantage that it can be optimized to produce a fast mode of operation.

In the micro programmed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro operations.

A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. In the micro programmed control, any required changes or modifications can be done by updating the micro program in control memory.

Control unit



The block diagram of the control unit is shown in Fig. 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 divided into three parts: the I bit, the operation code, and bits 0 through 1 1. 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 D0 through D7• 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 I. 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 counter are decoded into 16 timing signals T0 through T15•

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 sub cycles 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 T0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on. The micro operations for the fetch and decode phases can be specified by the following register transfer statements.

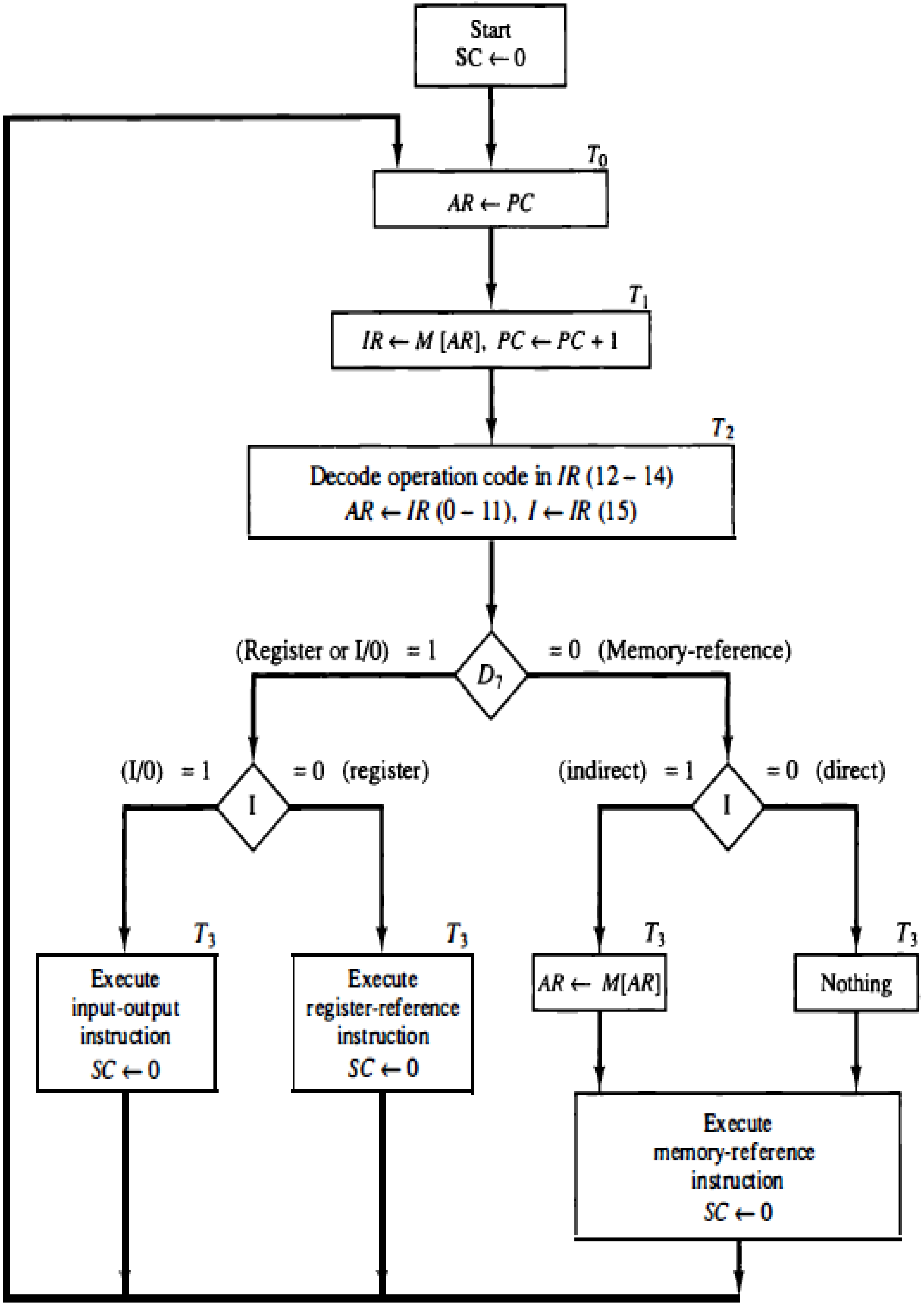
T0: AR ← PC

T,: IR ←M[AR], PC ← PC + 1

T2: D0, • • • , D7 ← Decode IR(12-14), AR ← IR(0-11), 1 ← IR(l5)

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. The flowchart of Fig. presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding. Decoder output D, is equal to 1 if the operation code is equal to binary 1ll. If D7 =1, the instruction must be a register-reference or input-output type. If D7 = 0, 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 I = 1, we If D7 = 0 and I = 1, we have a memory reference instruction with an indirect address.


Start
SC ← 0
T_0
$AR \leftarrow PC$
T_1
$IR \leftarrow M[AR],\ PC \leftarrow PC + 1$
T_2
Decode operation code in IR (12 – 14)
$AR \leftarrow IR(0-11),\ I \leftarrow IR(15)$
(Register or I/0) = 1
D_7
= 0 (Memory-reference)
(I/0) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T_3
Execute input-output instruction $SC \leftarrow 0$
T_3
Execute register-reference instruction $SC \leftarrow 0$
T_3
$AR \leftarrow M[AR]$
T_3
Nothing
Execute memory-reference instruction $SC \leftarrow 0$

Input –output and Interrupt

Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device.

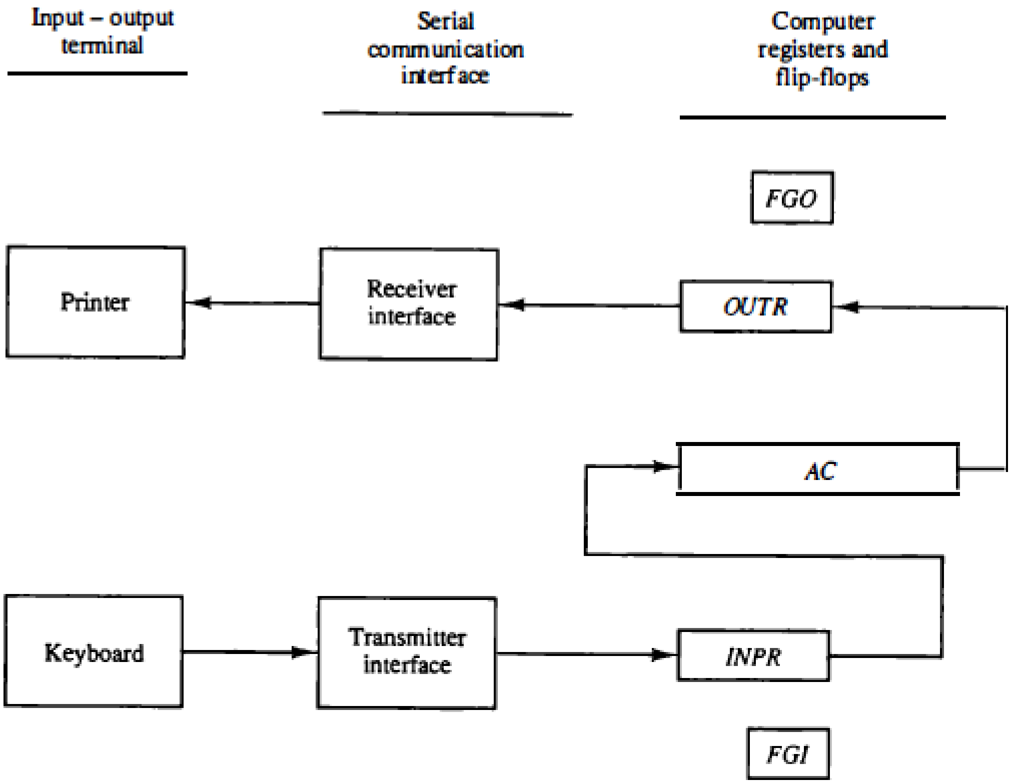
Input-Output Configuration

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.

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.

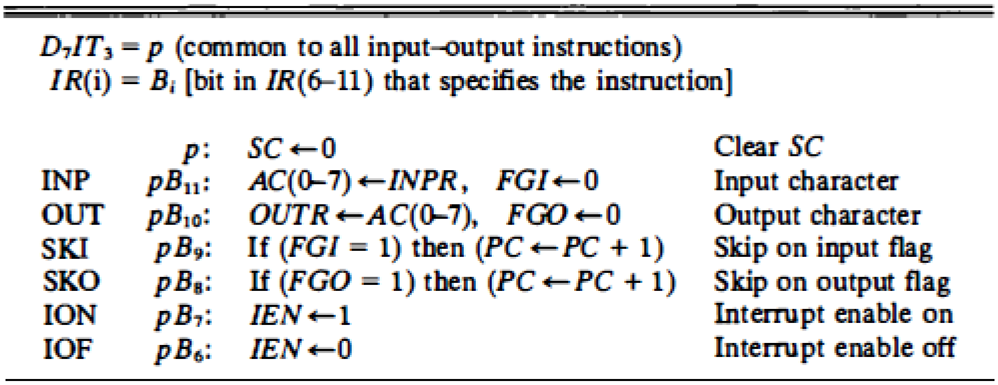
The input register INPR consists of eight bits and holds an alphanumeric input information. The 1-bit input flag FGI is a control flip-flop. The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer. The flag is 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 FGI 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 FGI 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 FGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key.

Input output configuration



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 0. 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 OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

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

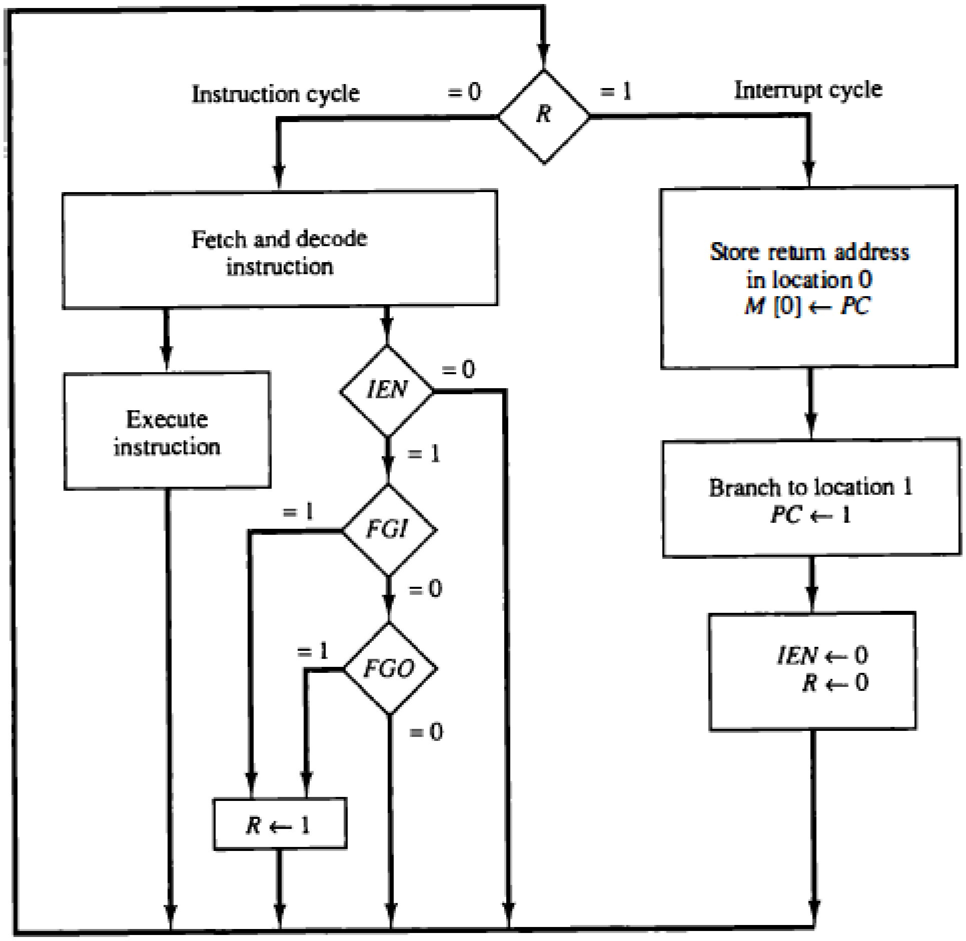
Program Interrupt

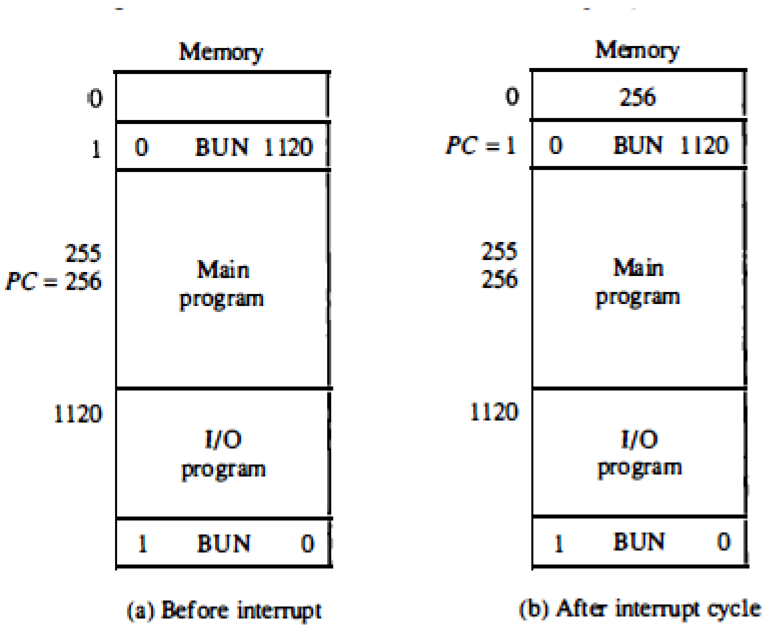
The interrupt enable flip-flop IEN can be set and cleared with two instructions. When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer. When IEN is set to 1 (with the ION instruction), the computer can be interrupted.

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-flop 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.

FLOW CHART FOR INTERRUPT CYCLE





Micro programmed control

Control Memory

The function of the control unit in a digital computer is to initiate sequences of micro operations. When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.

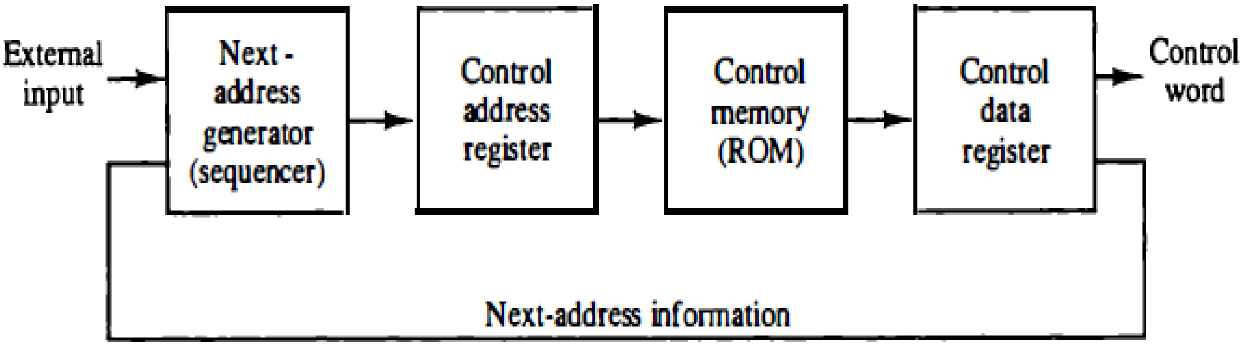
Microprogramming is a second alternative for designing the control unit of a digital computer. The control unit initiates a series of sequential steps of micro operations. During any given time, certain micro operations are to be initiated, while others remain idle. The control variables at any given time can be represented by a string of l's and O's called a control word.

A control unit whose binary control variables are stored in memory is called a micro programmed control unit. Each word in control memory contains within it a micro instruction. The microinstruction specifies one or more micro operations for the system. A sequence of microinstructions constitutes a micro program.

A memory that is part of a control unit is referred to as a control memory.

A computer that employs a micro programmed control unit will have two separate memories: a main memory and a control memory. The main memory is available to the user for storing the programs. The contents of main memory may alter when the data are manipulated and every time that the program is changed. The user's program in main memory consists of machine instructions and data. In contrast, the control memory holds a fixed micro program that cannot be altered by the occasional user. The micro program consists of microinstructions that specify various internal control signals for execution of register micro operations. Each machine instruction initiates a series of microinstructions in control memory. These microinstructions generate the micro operations to fetch the instruction from main memory; to evaluate the effective address, to execute the operation specified by the instruction, and to return control to the fetch phase in order to repeat the cycle for the next instruction.

The control memory is assumed to be a ROM, within which all control information is permanently stored.



Micro programmed control organization

The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.

The microinstruction contains a control word that specifies one or more micro operations for the data processor. Once these operations are executed, the control must determine the next address. The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory.

The next address generator is sometimes called a micro program sequencer, as it determines the address sequence that is read from control memory.

The control data register holds the present microinstruction while the next address is computed and read from memory. The data register is sometimes called a pipeline register.

The main advantage of the micro programmed control is the fact that once the hardware configuration is established, there should be no need for further hardware or wiring changes. If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory. The hardware configuration should not be changed for different operations; the only thing that must be changed is the micro program residing in control memory.

Address Sequencing

Microinstructions are stored in control memory in groups, with each group specifying a routine. Each computer instruction has its own micro program routine in control memory to generate the micro operations that execute the instruction. The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another.

An initial address is loaded into the control address register when power is turned on in the computer. This address is usually the address of the first micro instruction that activates the instruction fetch routine. The fetch routine may be sequenced by incrementing the control address register.

The control memory next must go through the routine that determines the effective address of the operand.

Mapping

The next step is to generate the micro operations that execute the instruction fetched from memory. The micro operation steps to be generated in processor registers depend on the operation code part of the instruction. Each instruction has its own micro program routine stored in a given location of control memory. The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a mapping process. A mapping procedure is a rule that transforms the instruction code into a control memory address.

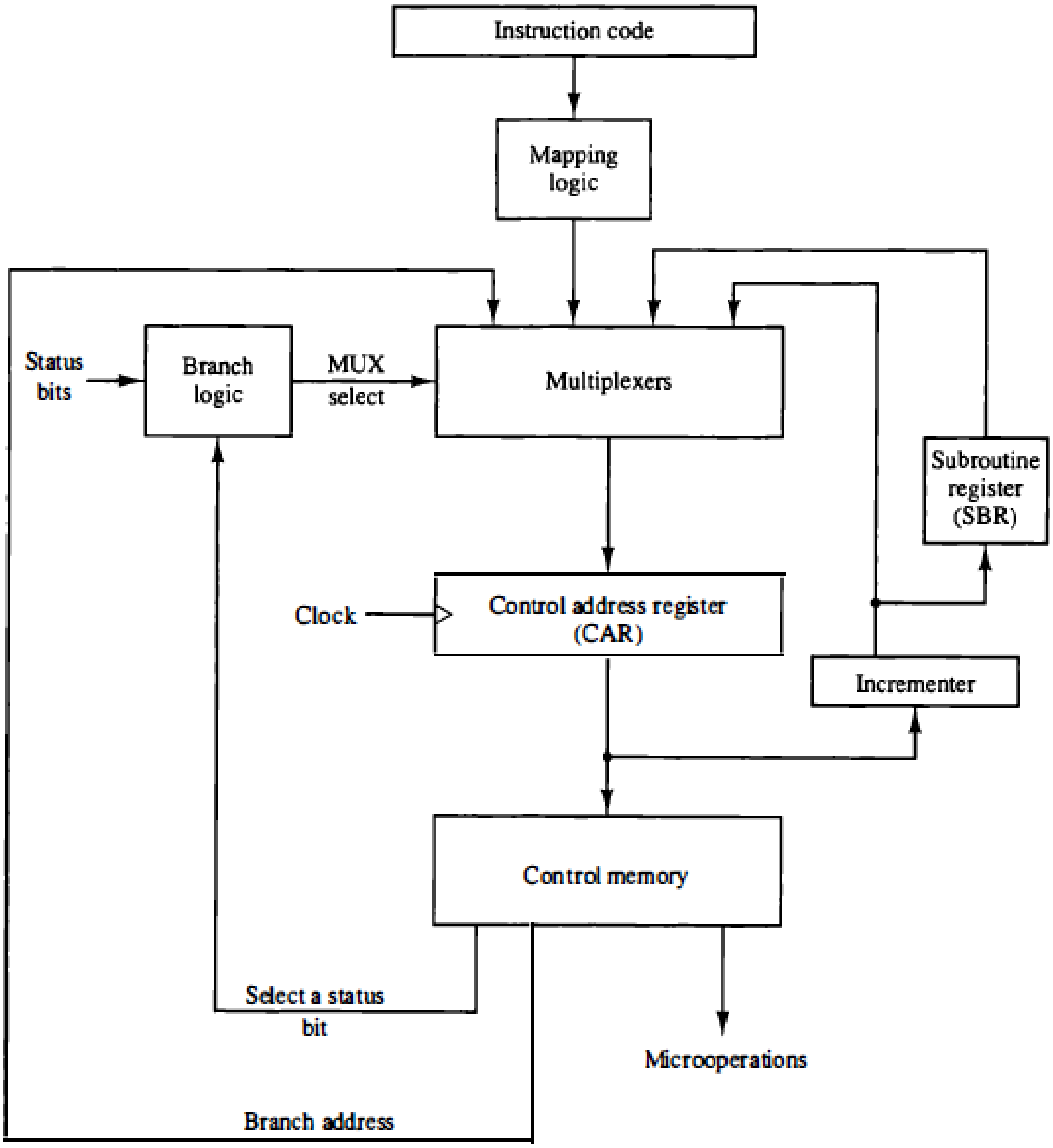
When the execution of the instruction is completed, control must return to the fetch routine. This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine. In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.

2. Unconditional branch or conditional branch, depending on status bit conditions.

3. A mapping process from the bits of the instruction to an address for control memory.

4. A facility for subroutine call and return.



Selection of address for control memory

Figure shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.

The microinstruction in control memory contains a set of bits to initiate micro operations in computer registers and other bits to specify the method by which the next address is obtained. The diagram shows four different paths from which the control address register (CAR) receives the address. The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence.

Branching is achieved by specifying the branch address in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition. An external address is transferred into control memory via a mapping logic circuit. The return address for a subroutine is stored in a special register whose value is then used when the micro program wishes to return from the subroutine.

Conditional branching

The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions. Information in these bits can be tested and actions initiated based on their condition: whether their value is 1 or 0. The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.

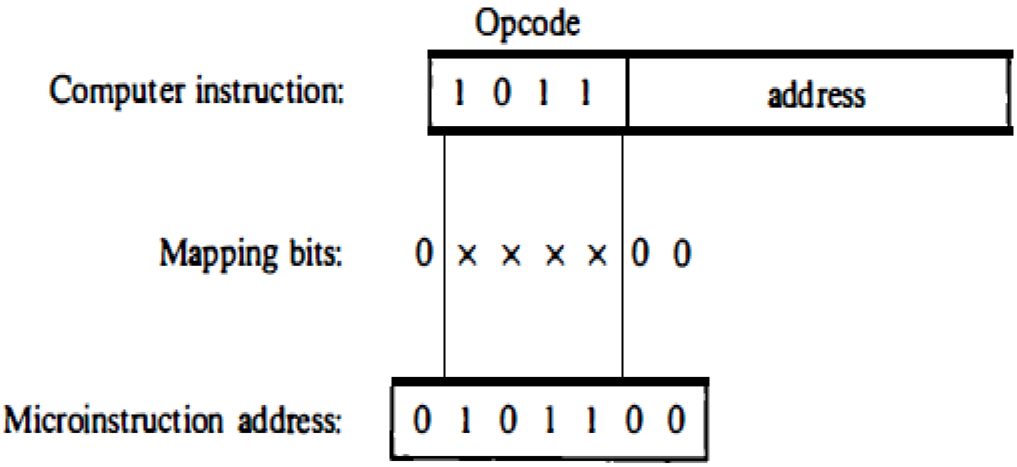
The branch logic hardware may be implemented in a variety of ways. The simplest way is to test the specified condition and branch to the indicated address if the condition is met; otherwise, the address register is incremented.

An unconditional branch microinstruction can be implemented by loading the branch address from control memory into the control address register.

This can be accomplished by fixing the value of one status bit at the input of the multiplexer, so it is always equal to 1. A reference to this bit by the status bit select lines from control memory causes the branch address to be loaded into the control address register unconditionally.

Mapping of Instruction

A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a micro program routine for an instruction is located. The status bits for this type of branch are the bits in the operation code part of the instruction.



Mapping of instruction code to micro instruction address

For each operation code there exists a micro program routine in control memory that executes the instruction. One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in Fig.

This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register. This provides for each computer instruction a micro program routine with a capacity of four microinstructions. If the routine needs more than four microinstructions, it can use addresses 1000000 through 1 1 1 1 1 1 1. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

Subroutines

Subroutines are programs that are used by other routines to accomplish a particular task. A subroutine can be called from any point within the main body of the micro program. Frequently, many micro programs contain identical sections of code. Microinstructions can be saved by employing subroutines that use common sections of microcode.

For example, the sequence of micro operations needed to generate the effective address of the operand for an instruction is common to all memory reference instructions. This sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Micro programs that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return. This may be accomplished by placing the incremented 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 main routine. The best way to structure a register file that stores addresses for subroutines is to organize the registers in a last-in, first-out (LIFO) stack.

Design of Control Unit

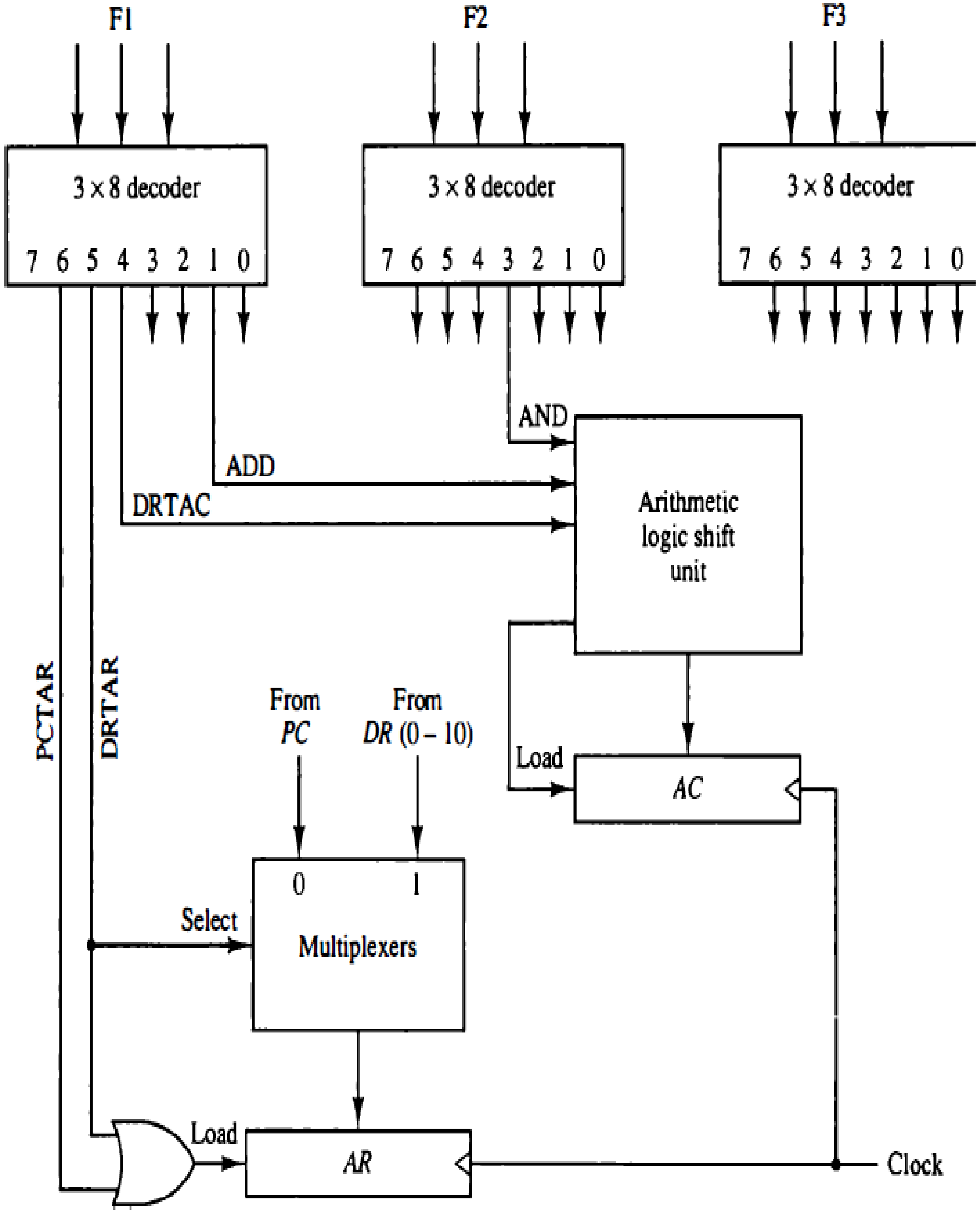
The bits of the microinstruction are usually divided into fields, with each field defining a distinct, separate function.

The various fields encountered in instruction formats provide control bits to initiate micro operations in the system, special bits to specify the way that the next address is to be evaluated, and an address field for branching.

The number of control bits that initiate micro operations can be reduced by grouping mutually exclusive variables into fields and encoding the k bits in each field to provide 2^k micro operations. Each field requires a decoder to produce the corresponding control signals. This method reduces the size of the microinstruction bits but requires additional hardware external to the control memory. It also increases the delay time of the control signals because they must propagate through the decoding circuits.

The encoding of control bits was demonstrated in the programming example of the preceding section. The nine bits of the micro operation field are divided into three subfields of three bits each. The control memory output of each subfield must be decoded to provide the distinct micro operations.

The outputs of the decoders are connected to the appropriate inputs in the processor unit.



Decoding of micro operation fields

Figure shows the three decoders and some of the connections that must be made from their outputs.

Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3 x 8 decoder to provide eight outputs.

Micro program Sequencer

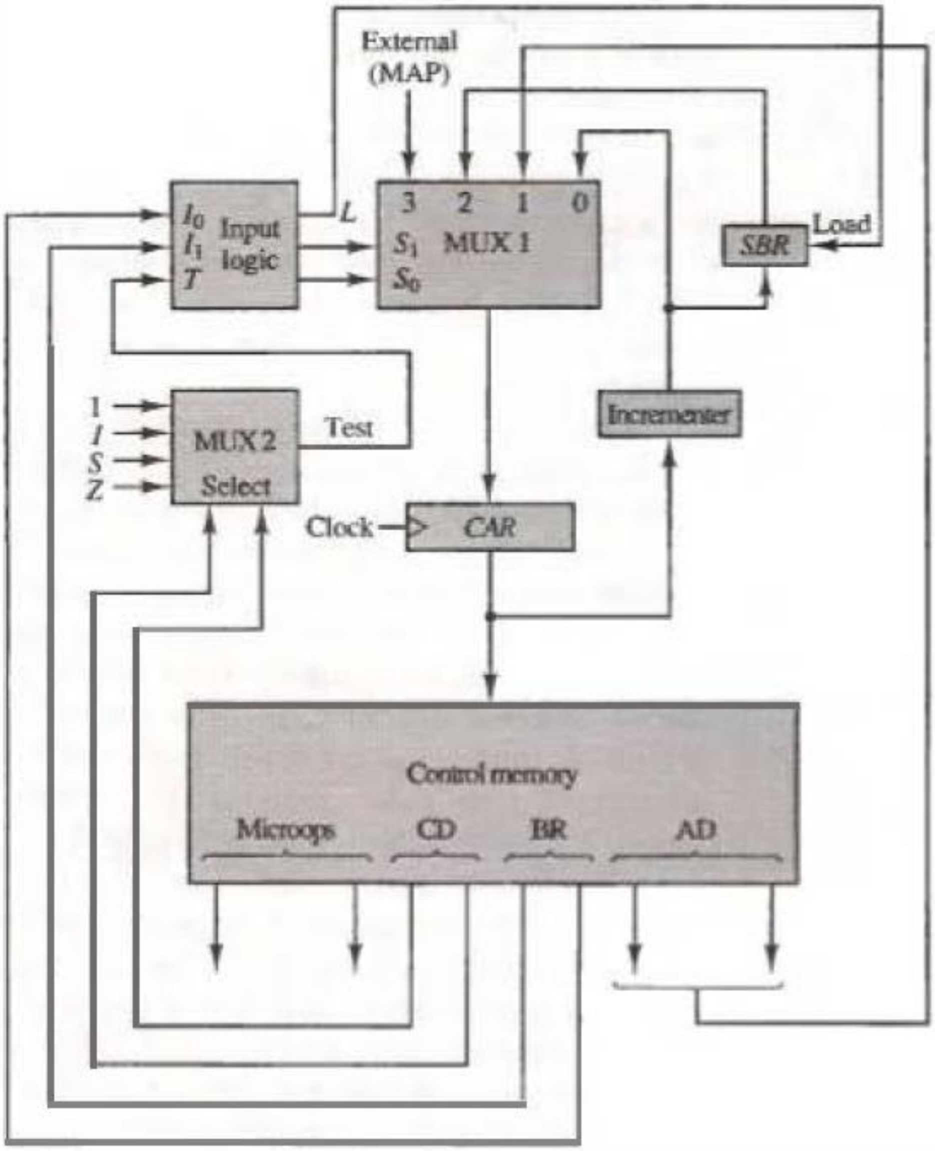
The basic components of a micro programmed control unit are the control memory and the circuits that select the next address. The address selection part is called a micro program sequencer.

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

The purpose of a micro program sequencer is to present an address to the control memory so that a microinstruction may be read and executed.

The next-address logic of the sequencer determines the specific address source to be loaded into the control address register. The choice of the address source is guided by the next-address information bits that the sequencer receives from the present microinstruction.

Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during micro program looping and subroutine calls. Some sequencers provide an output register which can function as the address register for the control memory.



Micro program sequencer for a control memory

The block diagram of the micro program sequencer is shown in Fig. The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it. There are two multiplexers in the circuit. The first multiplexer selects an address from one of four sources and routes it into a control address register CAR. The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.

The output from CAR provides the address for the control memory. The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register SBR. The other three inputs to multiplexer number 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction. Although the diagram shows a single subroutine register, a typical sequencer will have a register stack about four to eight levels deep. In this way, a number of subroutines can be active at the same time. A push and pop operation, in conjunction with a stack pointer, stores and retrieves the return address during the call and return microinstructions.

The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer. If the bit selected is equal to 1, the T (test) variable is equal to 1; otherwise, it is equal to 0. The T value together with the two bits from the BR (branch) field go to an input logic circuit. The input logic in a particular sequencer will determine the type of operations that are available in the unit. Typical sequencer operations are: increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations. With three inputs, the sequencer can provide up to eight address sequencing operations. Some commercial sequencers have three or four inputs in addition to the T input and thus provide a wider range of operations.

Design of input logic

The input logic circuit in Fig. has three inputs, l0, l1, and T, and three outputs, S0, S1, and L. Variables So and S, select one of the source addresses for CAR. Variable L enables the load input in SBR.

The binary values of the two selection variables determine the path in the multiplexer. For example, with S1 S0 = 10, multiplexer input number 2 is selected and establishes a transfer path from SBR to CAR. Note that each of the four inputs as well as the outputs of MUX 1 contains a 7-bit address.

The truth table for the input logic circuit is shown in Table. Inputs I1and I0 are Identical to the bit values in the BR field. The function listed in each entry was defined in Table.

The bit values for S1 and S0 are determined from the stated function and the path in the multiplexer that establishes the required transfer. The subroutine register is loaded with the incremented value of CAR during a call microinstruction (BR=01) provided that the status bit condition is satisfied (T = 1).

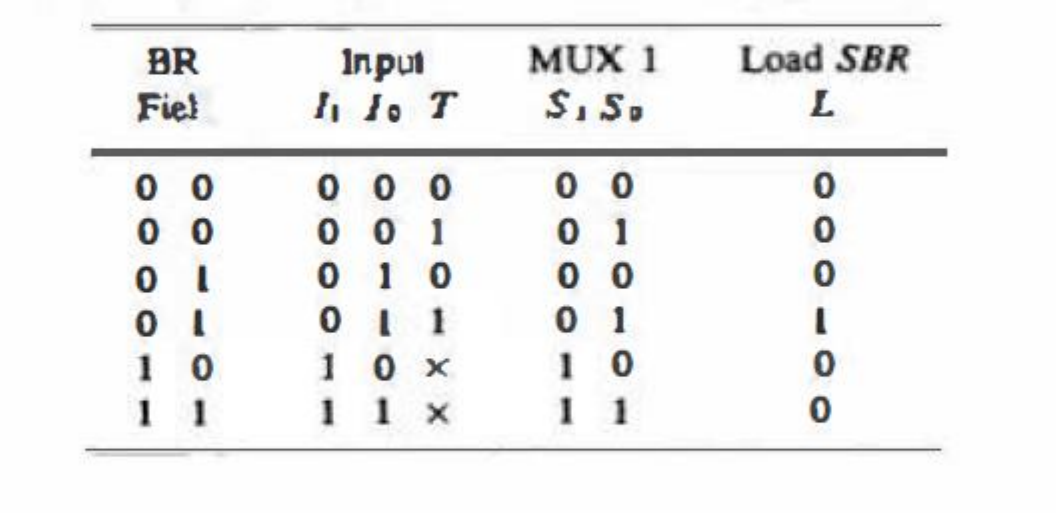
The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

$$S1 = I1$$

$$S0= I1\ I0 + I1'T$$

$$L = Il'\ I0\ T$$

Input logic table for micro program sequencer



BR		Input			MUX 1		Load *SBR*
Fiel		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 an inverter. Note that the incrementer circuit in the sequencer of Fig. is not a counter constructed with flip-flops but rather a combinational circuit constructed with gates. A combinational circuit incrementer can be designed by cascading a series of half-adder circuits. The output carry from one stage must be applied to the input of the next stage. One input in the first least significant stage must be equal to 1 to provide the increment-by-one operation.