Organization of Computer

be specified with register transfer statements. The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses. The design of the computer is then

Digital Modules

Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operations

Microoperation

operations that are performed on the data stored in them. The operations executed on data stored in registers are called microoperations. A microoperation is an elementary operation performed on the information stored in one or more registers. The result of the operation may replace the previous binary information of a register or may be transferred to another register. Examples

Digital System

A digital system is an interconnection of digital hardware modules that accomplish a specific information-processing task. Digital systems vary in size and

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

Internal Hardware Organization

The internal hardware organization of a digital computer is best defined by specifying:

1. The set of registers it contains and their function.
2. The sequence of microoperations performed on the binary information stored in the registers.
3. The control that initiates the sequence of microoperations.

Program

perform. The user of a computer can control the process by means of a program. A program is a set of instructions that specify the operations, operands, and the sequence by which processing has to occur. The data-processing task may be altered by specifying a new program with different instructions or specifying the same instructions with different data.

Instruction Code

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

Operation Code

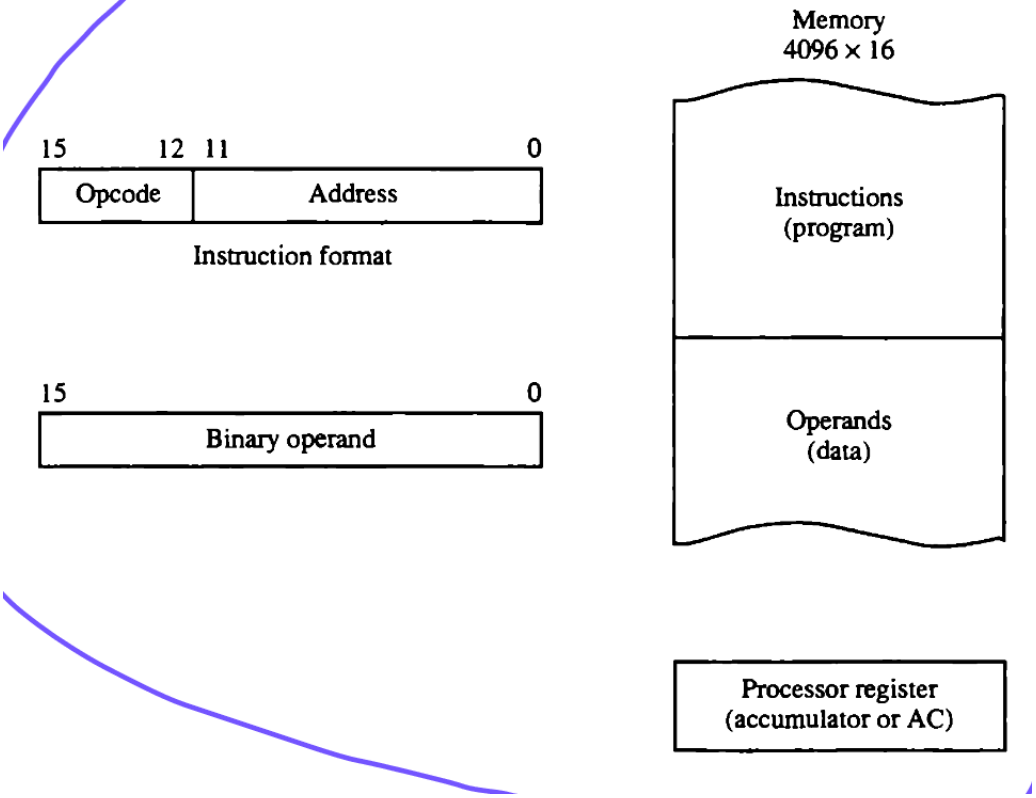
operation part. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer. The operation code

Stored Program Organization

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.



Accumulator

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 addressing,

the instruction contains a memory address that, in turn, holds the actual address of the data.The CPU follows this indirect address to locate the data. The indirect address serves as a pointer to the actual memory location.An example might be "load the value from the memory address stored in memory location 2000 into register B.

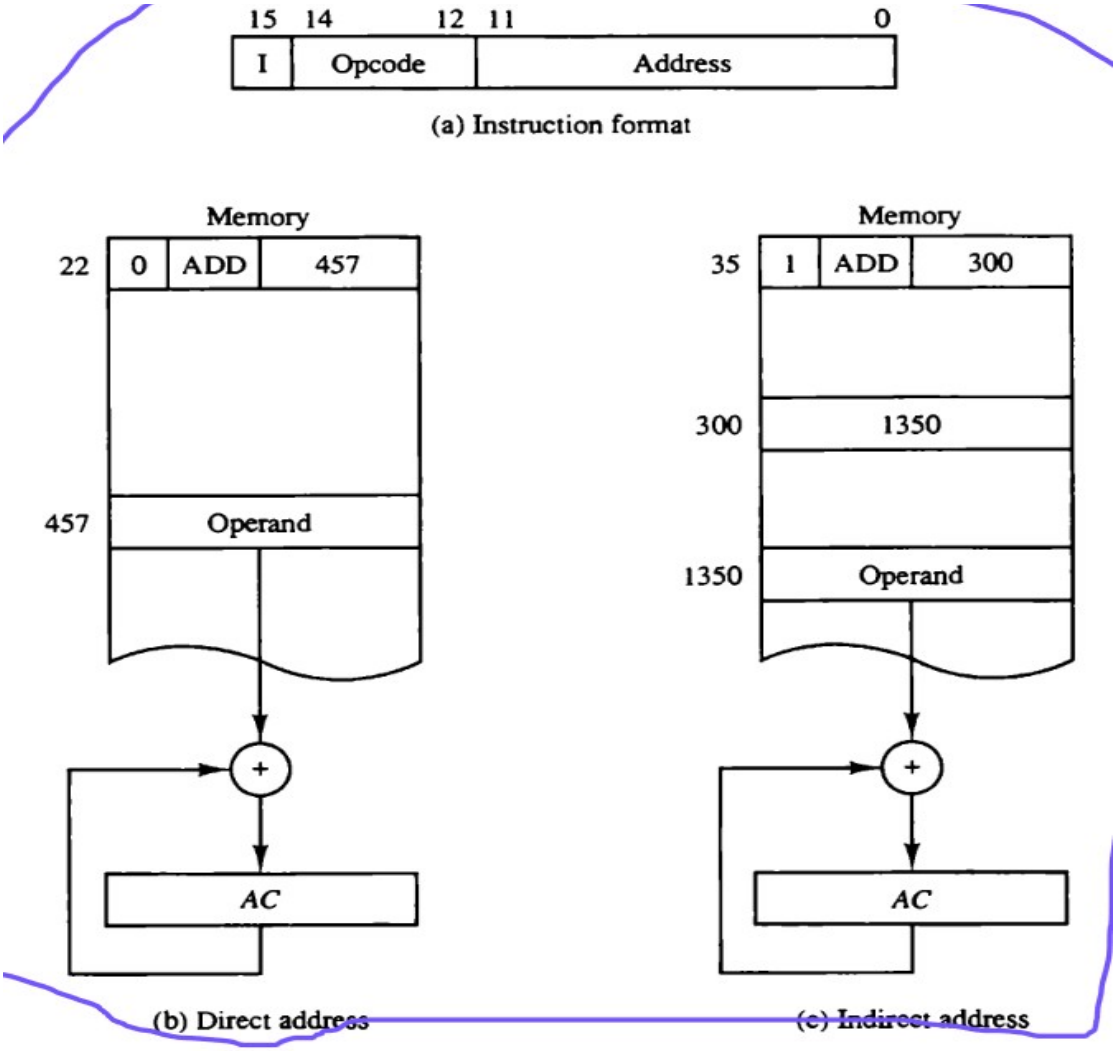


Figure 5-2 Demonstration of direct and indirect address.

Computer Register

Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time. The control reads an instruction

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

Common Bus System

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

The 16 lines of the common bus receive information from six registers and the memory unit. The bus lines are connected to the inputs of six registers and the memory. Five registers have three control inputs: LD (load), INR (increment), and CLR (clear). This type of register is equivalent to a binary counter with parallel load and synchronous clear similar to the one shown in Fig. 2-11. The increment operation is achieved by enabling the count input of the counter. Two registers have only a LD input. This type of register is shown in Fig. 2-7.

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 always be used to specify a memory address. By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise. The content of any register can be specified for the memory

LD (Load) is used to receive data.

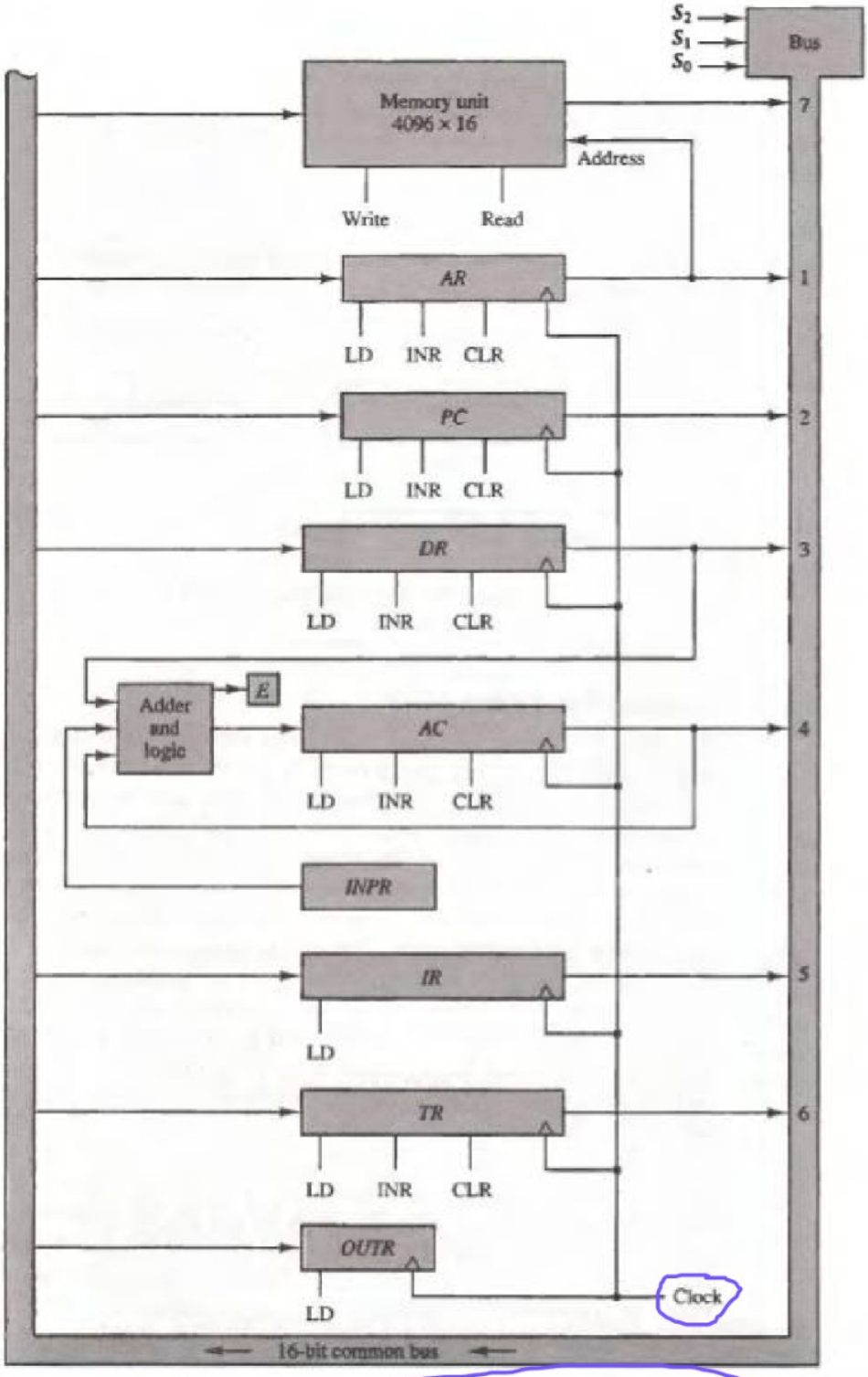
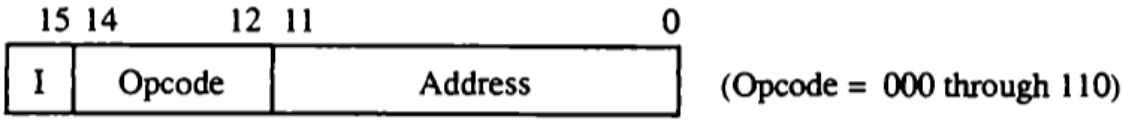


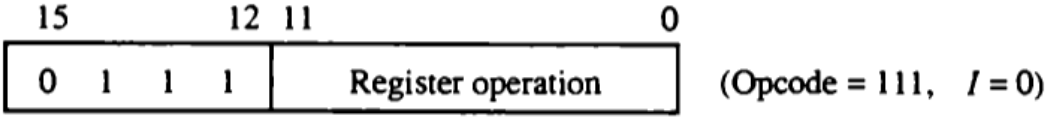
Figure 5-4 Basic computer registers connected to a common bus.

Instruction Format

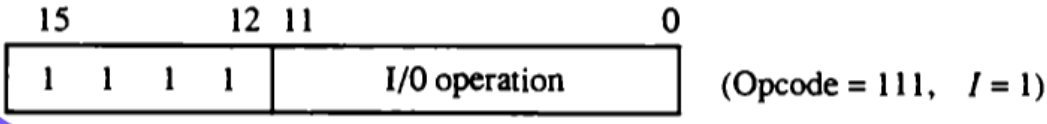
Figure 5-5 Basic computer instruction formats.



(a) Memory – reference instruction



(b) Register – reference instruction



(c) Input – output instruction

"I" refers to the type of address, 0 for direct and 1 for Indirect address

Instruction Set Completeness

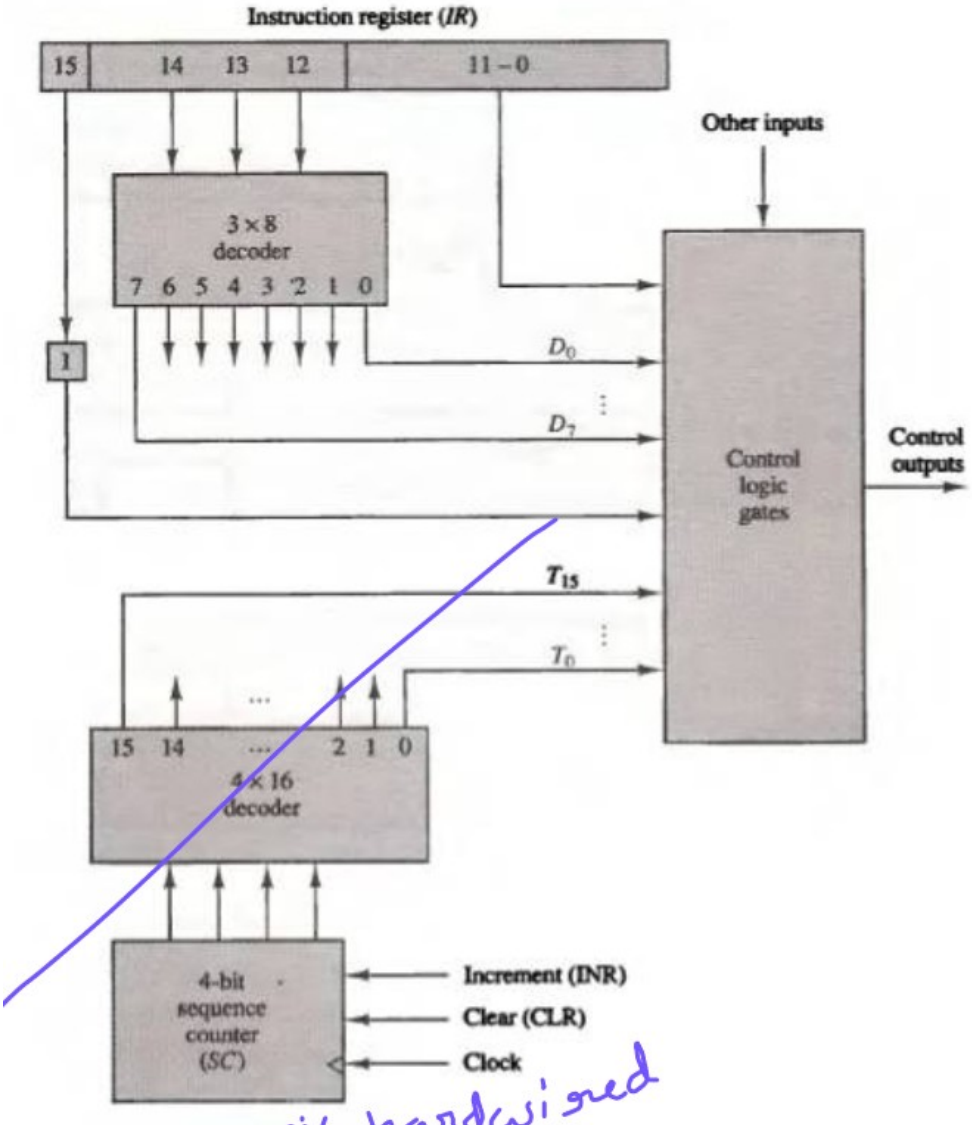
Instruction Set Completeness

Before investigating the operations performed by the instructions, let us discuss the type of instructions that must be included in a computer. A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable. 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

Hardwired and Microprogrammed Control

There are two major types of control organization: hardwired control and microprogrammed 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 microprogrammed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations. 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 microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory. A hardwired control for the basic computer is presented in this section. A microprogrammed control unit for a similar computer is presented in Chap. 7.



basic hardwired

Figure 5-6 Control unit of basic computer.

5-5 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

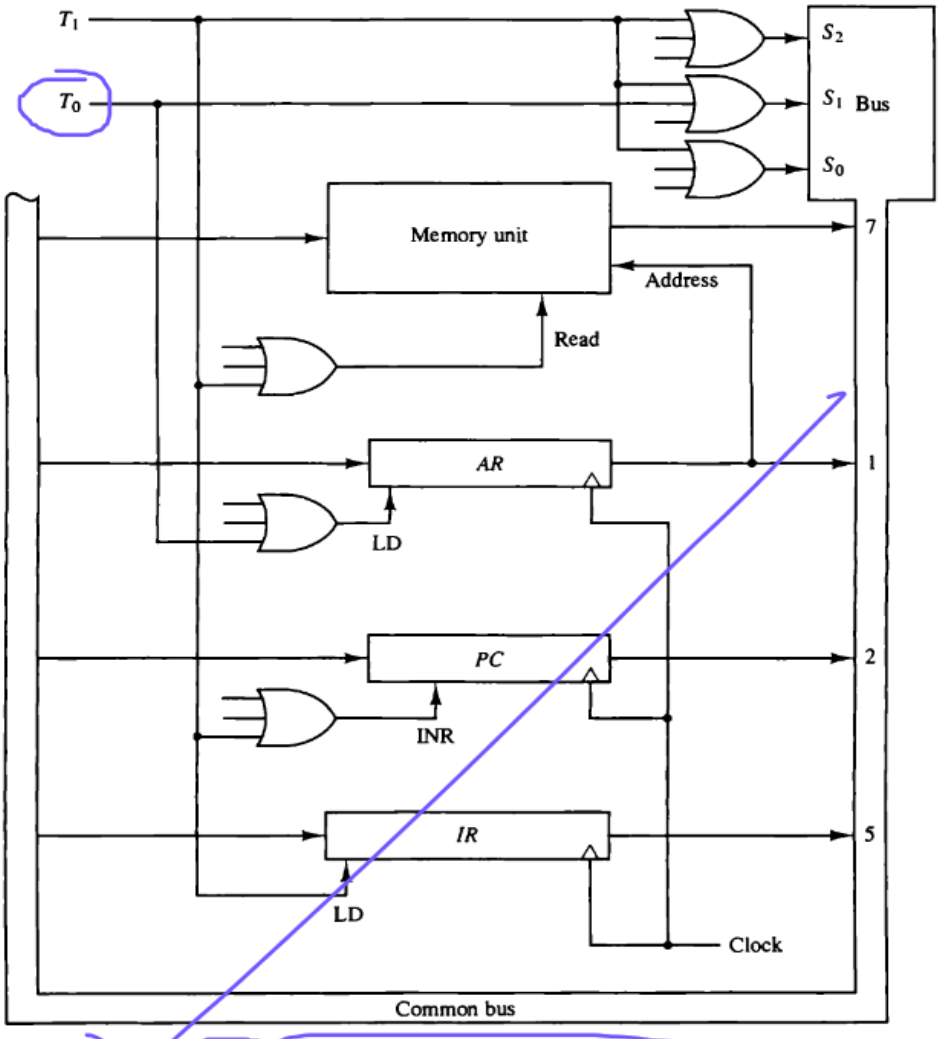


Figure 5-8 Register transfers for the fetch phase.

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 T_0 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 $T_0 = 1$. In order to implement the second statement

$$T_1: \quad IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$

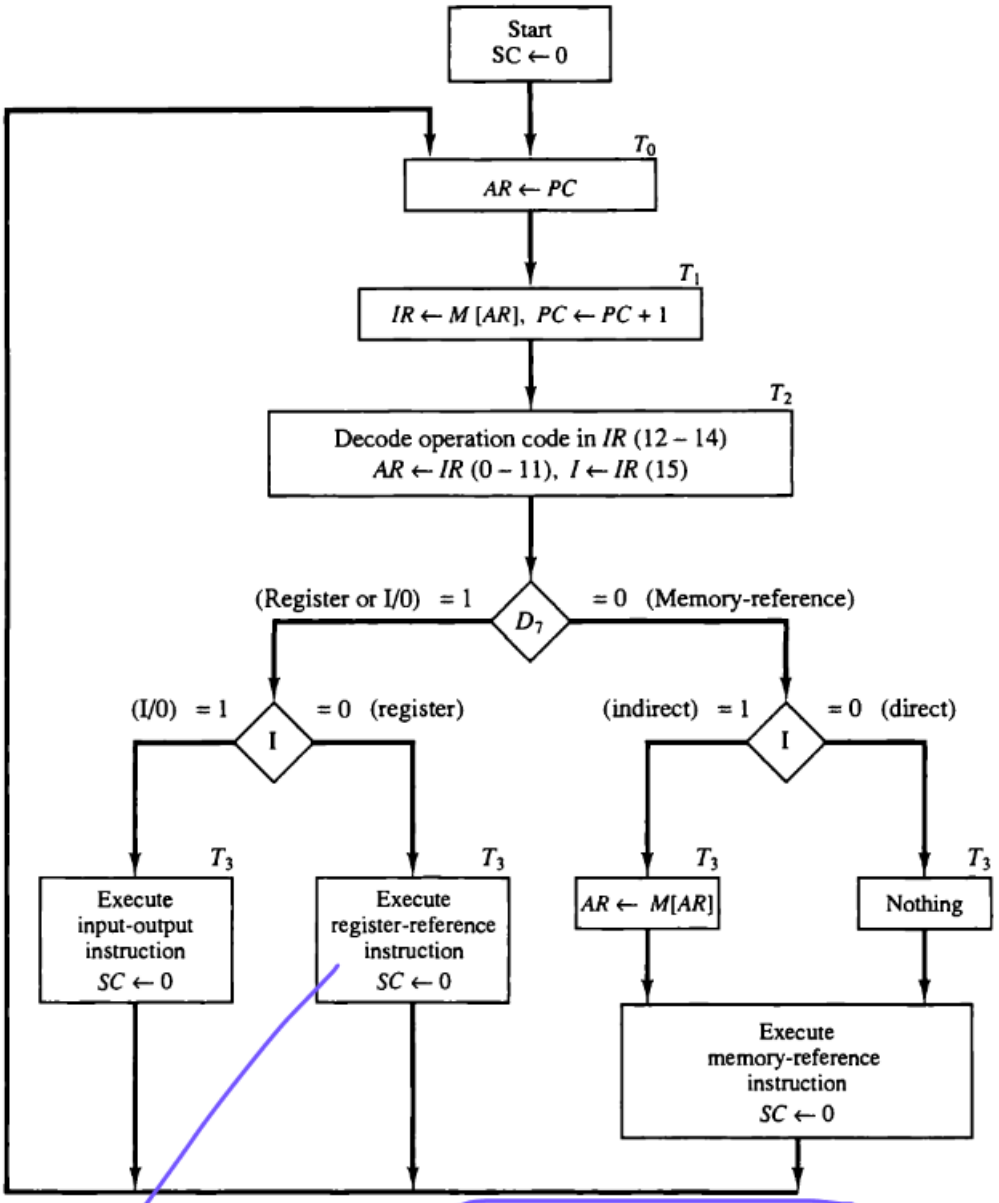


Figure 5-9 Flowchart for instruction cycle (initial configuration).

Register-Reference Instructions

Register-reference instructions are recognized by the control when $D_7 = 1$ 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 T_2.

Memory Refernce Instruction

listed, we find that some instructions have an ambiguous description. This is because the explanation of an instruction in words is usually lengthy, and not enough space is available in the table for such a lengthy explanation. We will now show that the function of the memory-reference instructions can be defined precisely by means of register transfer notation.

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 to each instruction is included in the table. The effective address of the instruc-

Register Transfer Language (RTL)

The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language. The term "register transfer" implies the availability of hardware logic circuits that can perform a stated microoperation and transfer the result of the operation to the same or another register. The word "language" is borrowed from programmers, who apply this term to programming languages. A programming language is a

Register Transfer

Information transfer from one register to another is designated in symbolic form by means of a replacement operator. The statement

$$R2 \leftarrow R1$$

Control Function

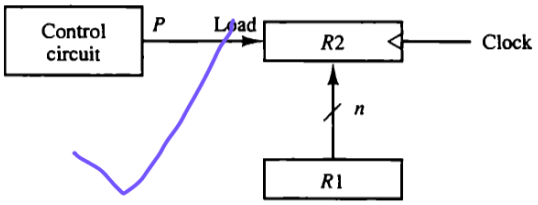
by specifying a *control function*. A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

$$P: \quad R2 \leftarrow R1$$

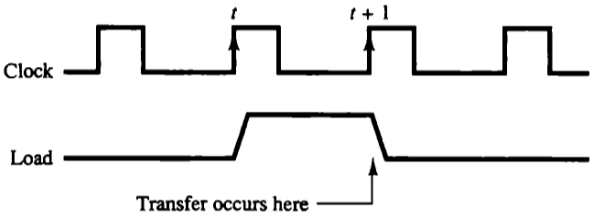
The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if $P = 1$.

Every statement written in a register transfer notation implies a hardware construction for implementing the transfer. Figure 4-2 shows the block dia-

Figure 4-2 Transfer from R1 to R2 when P = 1.



(a) Block diagram

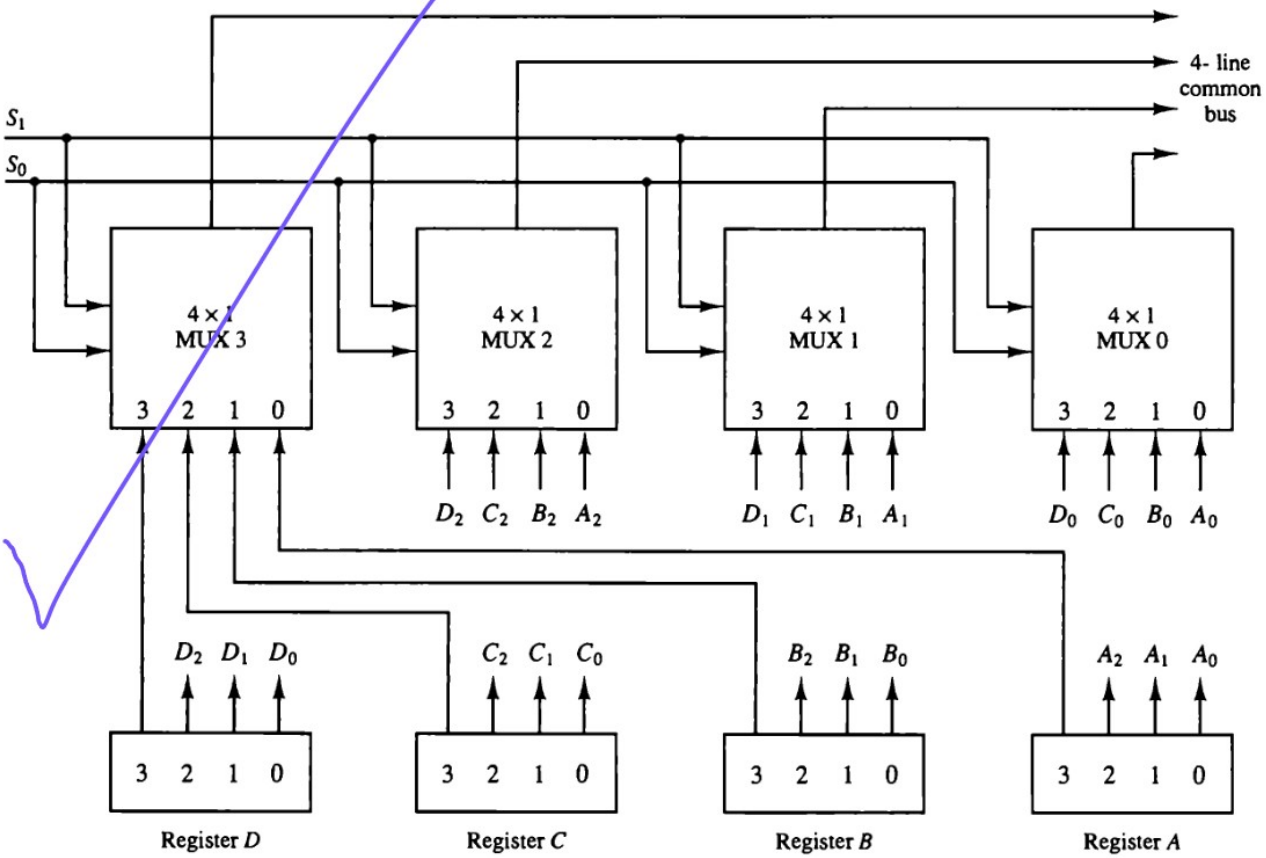


(b) Timing diagram

Common Bus

in the system. A more efficient scheme for transferring information between registers in a multiple-register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals

Figure 4-3 Bus system for four registers.



Arithmetic Microoperations

TABLE 4-3 Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

Arithmetic Circuit

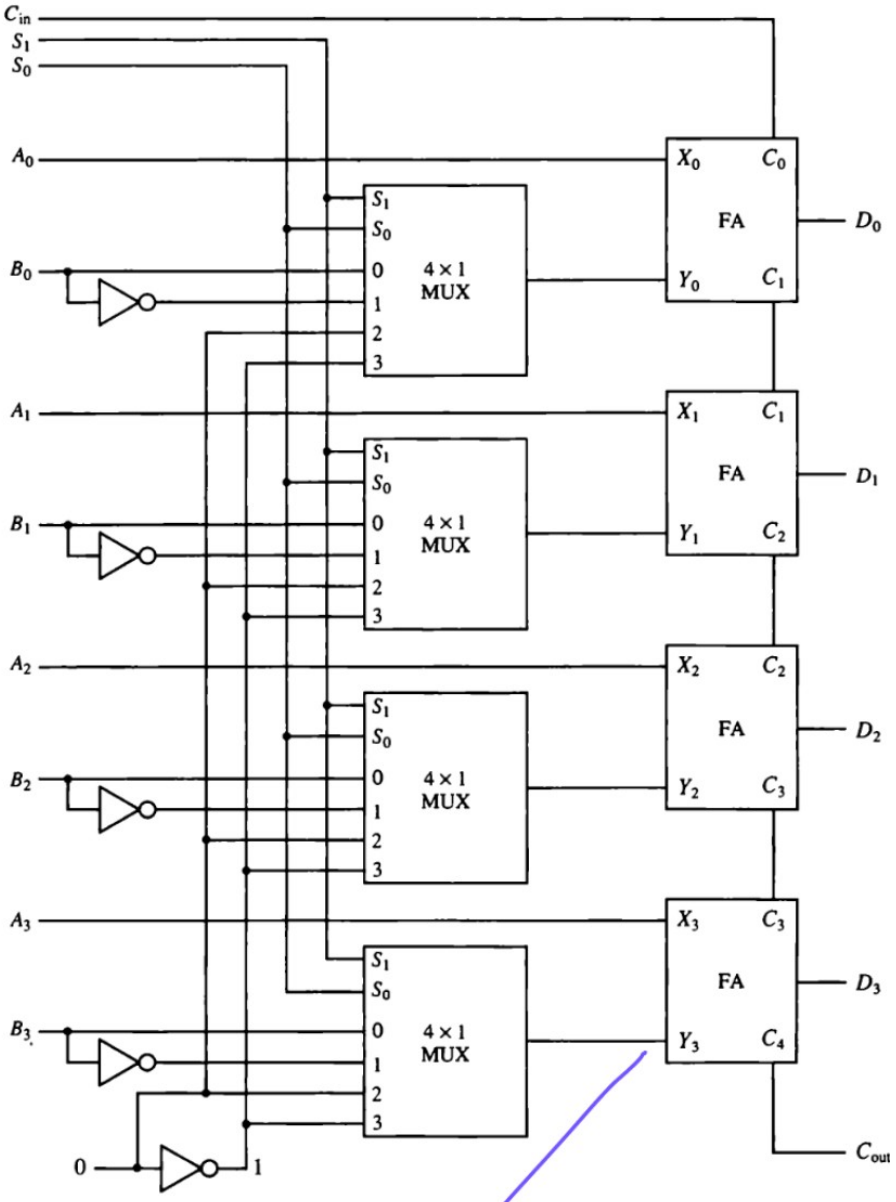


Figure 4-9 4-bit arithmetic circuit.

TABLE 4-4 Arithmetic Circuit Function Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

The output of the binary adder is calculated from the following arithmetic sum:

$$D = A + Y + C_{in}$$

4-5 Logic Microoperations

Logic microoperations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as binary variables. For example, the exclusive-OR microoperation with the contents of two registers $R1$ and $R2$ is symbolized by the statement

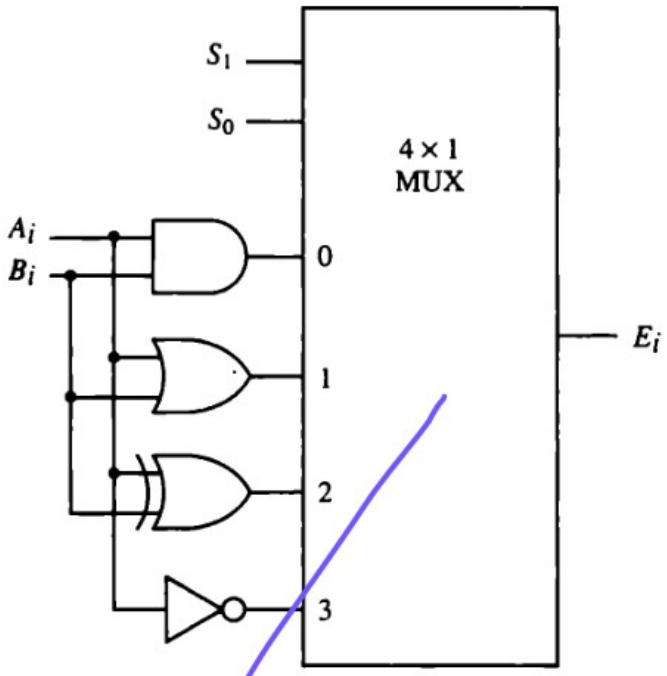
$$P: \quad R1 \leftarrow R1 \oplus R2$$

1010	Content of $R1$
1100	Content of $R2$
0110	Content of $R1$ after $P = 1$

Logic Circuit

Logic microoperations are very useful for manipulating individual bits or a portion of a word stored in a register. They can be used to change bit values, delete a group of bits, or insert new bit values into a register. The following examples show how the bits of one register (designated by A) are manipulated

Figure 4-10 One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

4-6 Shift Microoperations

Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operations. The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input. During a shift-left operation the serial input transfers a

Logical Shift

A *logical* shift is one that transfers 0 through the serial input. We will adopt the symbols *shl* and *shr* for logical shift-left and shift-right microoperations. For example:

R1 ← shl R1 1100 → 11000

R2 ← shr R2 → 110

Circular Shift

The *circular* shift (also known as a *rotate* operation) circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input. We will use the symbols *cil* and *cir* for the circular shift left and right, respectively. The symbolic notation for the shift microoperations is shown in Ta-

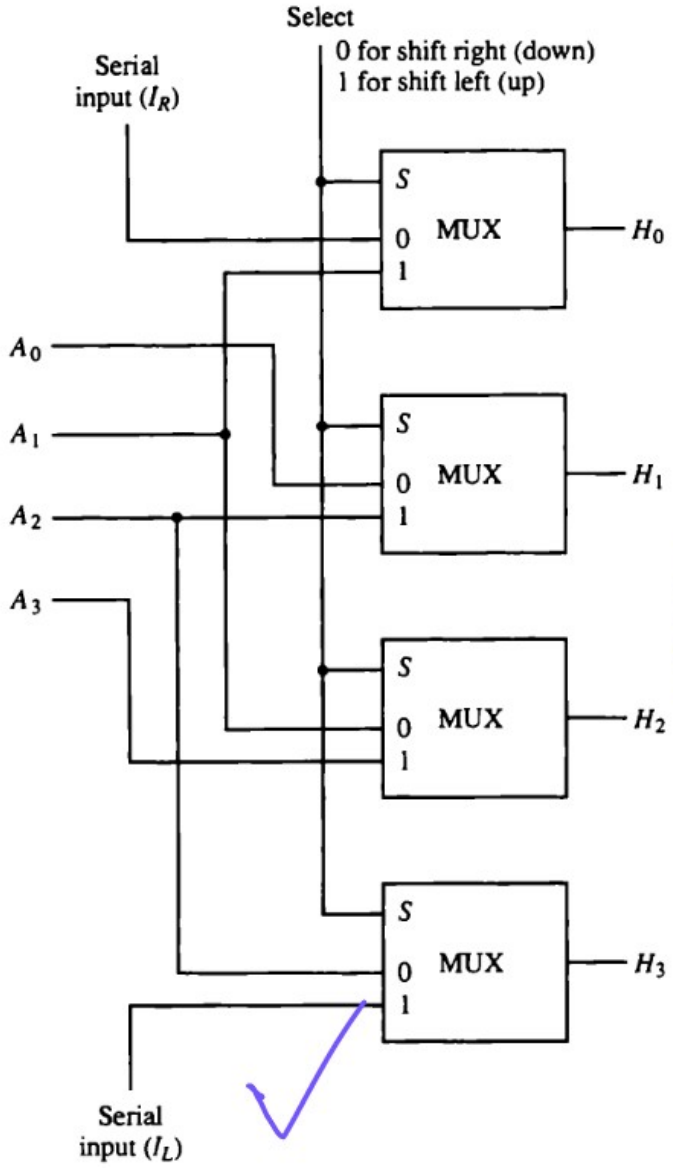
Arithmetic Shift

An *arithmetic* shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same

when it is multiplied or divided by 2. The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive

Shifter

A shifter, in the context of digital design and computer architecture, is a hardware component or circuit responsible for performing bitwise shift operations on binary data. A bitwise shift involves moving the bits of a binary number to the left or right by a certain number of positions. Shifters are commonly used in processors, microcontrollers, and other digital systems for various purposes



Function table

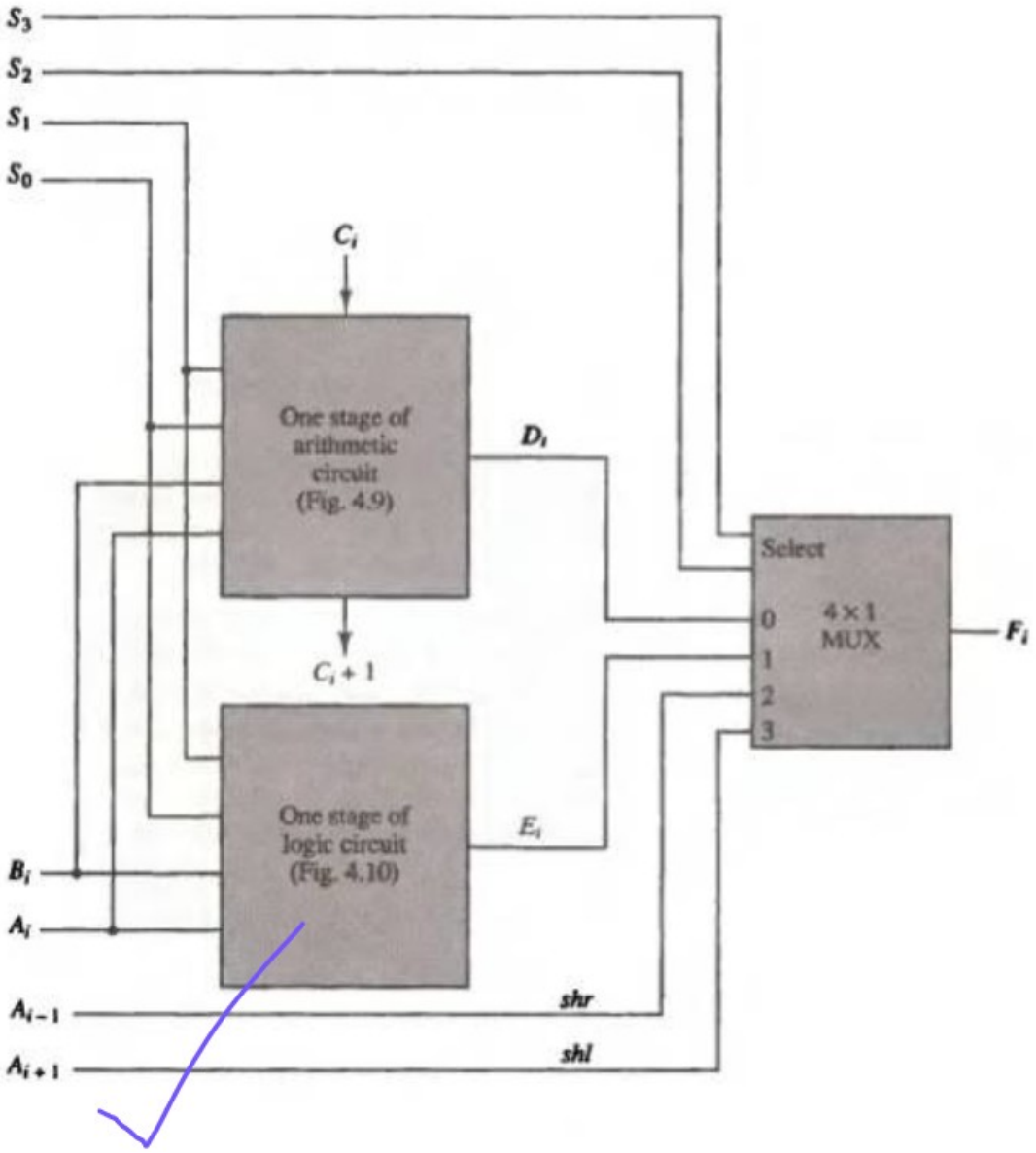
Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Figure 4-12 4-bit combinational circuit shifter.

4-7 Arithmetic Logic Shift Unit

Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU. To

Figure 4-13 One stage of arithmetic logic shift unit.



5-9 Design of Basic Computer

The basic computer consists of the following hardware components:

1. A memory unit with 4096 words of 16 bits each
2. Nine registers: *AR*, *PC*, *DR*, *AC*, *IR*, *TR*, *OUTR*, *INPR*, and *SC*
3. Seven flip-flops: *I*, *S*, *E*, *R*, *IEN*, *FGI*, and *FGO*
4. Two decoders: a 3 × 8 operation decoder and a 4 × 16 timing decoder
5. A 16-bit common bus
6. Control logic gates
7. Adder and logic circuit connected to the input of *AC*

The memory unit is a standard component that can be obtained readily from a commercial source. The registers are of the type shown in Fig. 2-11 and

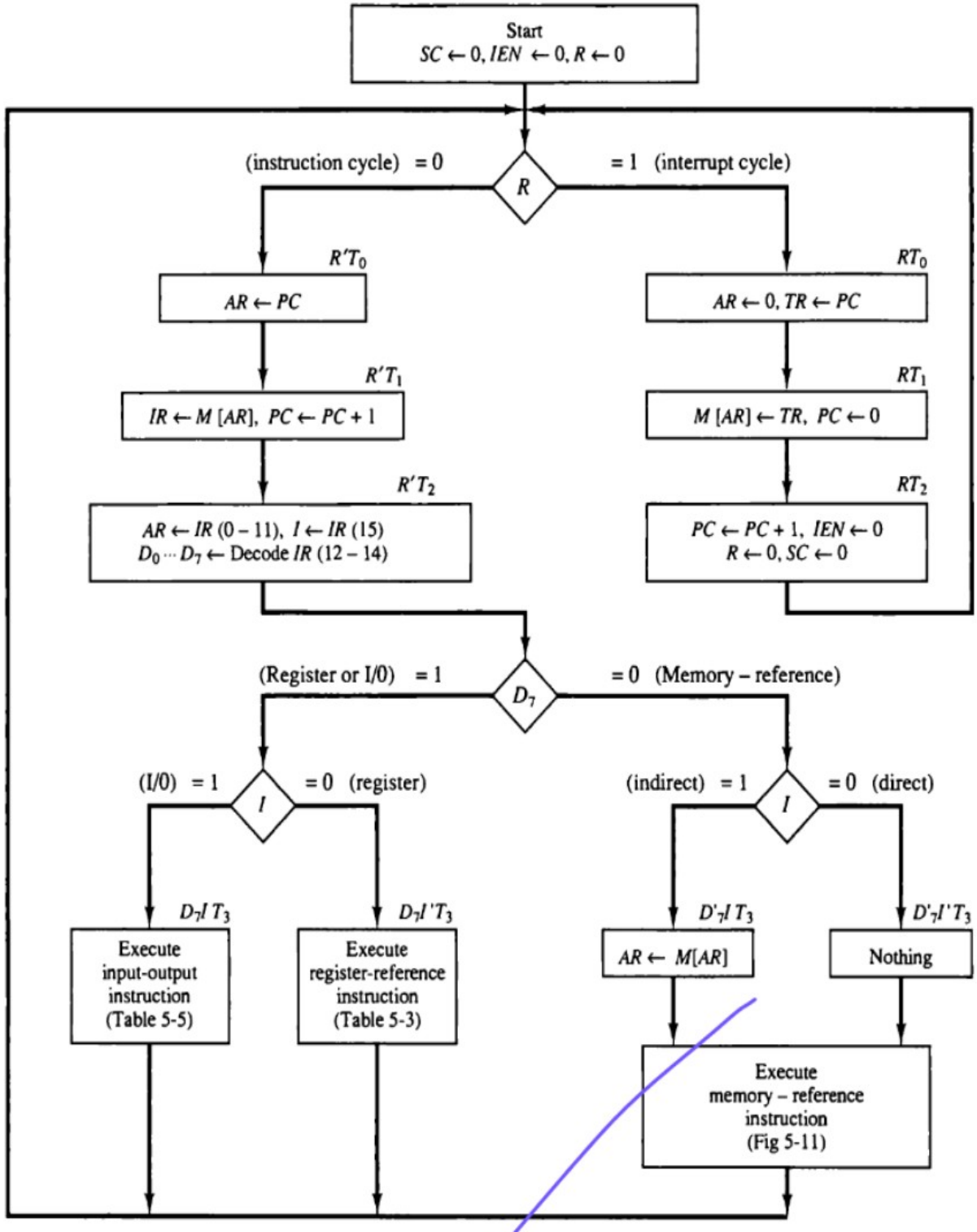


Figure 5-15 Flowchart for computer operation.

Instruction Set

An instruction set is a collection of instructions that a particular central processing unit (CPU) or microprocessor understands and can execute. Each instruction in the set represents a specific operation that the CPU can perform, such as arithmetic operations, data movement, logical operations, and control flow instructions.

Symbolic Code

Symbolic code. The user employs symbols (letters, numerals, or special characters) for the operation part, the address part, and other parts of the instruction code. Each symbolic instruction can be translated into one binary coded instruction. This translation is done by a special program called an *assembler*. Because an assembler translates the symbols, this type of symbolic program is referred to as an *assembly language* program.

Machine Language

Strictly speaking, a *machine language* program is a binary program of category 1. Because of the simple equivalency between binary and octal or hexadecimal representation, it is customary to refer to category 2 as machine language. Because of the one-to-one relationship between a symbolic instruction and its binary equivalent, an assembly language is considered to be a machine-level language.

Assembly Language

Rules of the Language

Each line of an assembly language program is arranged in three columns called fields. The fields specify the following information.

1. The *label* field may be empty or it may specify a symbolic address.
2. The *instruction* field specifies a machine instruction or a pseudoinstruction.
3. The *comment* field may be empty or it may include a comment.

A symbolic address consists of one, two, or three, but not more than three alphanumeric characters. The first character must be a letter; the next two may be letters or numerals. The symbol can be chosen arbitrarily by the programmer. A symbolic address in the label field is terminated by a comma so that it will be recognized as a label by the assembler.

The instruction field in an assembly language program may specify one of the following items:

1. A memory-reference instruction (MRI)
2. A register-reference or input–output instruction (non-MRI)
3. A pseudoinstruction with or without an operand

Pseudo-instruction

A pseudoinstruction is not a machine instruction but rather an instruction to the assembler giving information about some phase of the translation. Four

6-4 The Assembler

An assembler is a program that accepts a symbolic language program and produces its binary machine language equivalent. The input symbolic program

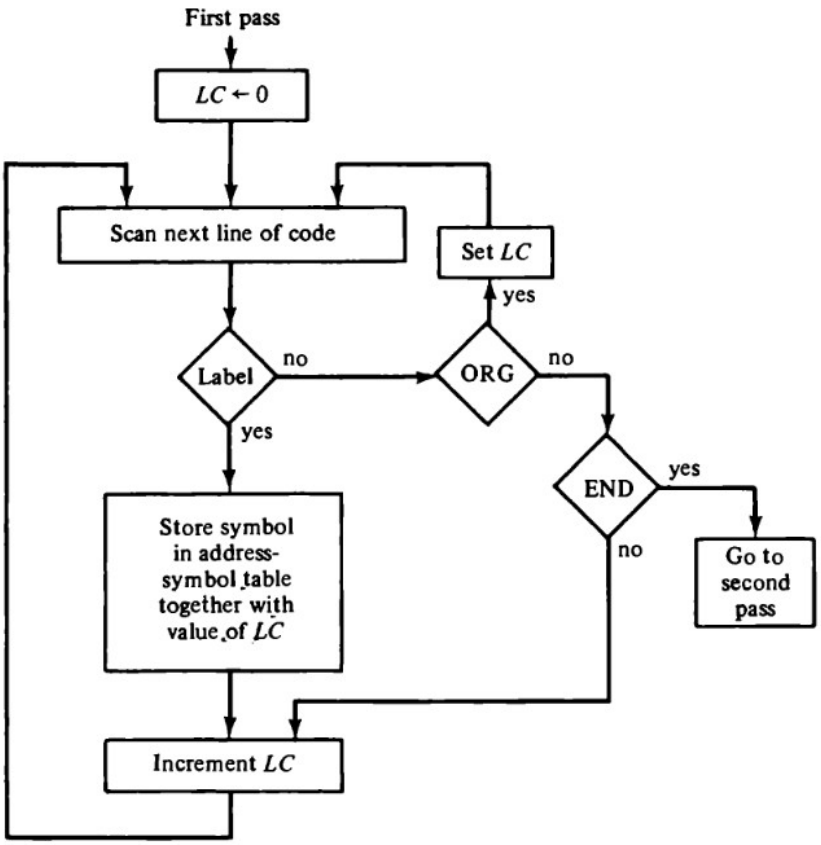


Figure 6-1 Flowchart for first pass of assembler.

their binary equivalent value. The binary translation is done during the second pass. To keep track of the location of instructions, the assembler uses a memory word called a *location counter* (abbreviated LC). The content of LC stores the

Control Unit

The function of the control unit in a digital computer is to initiate sequences of microoperations. The number of different types of microoperations that are

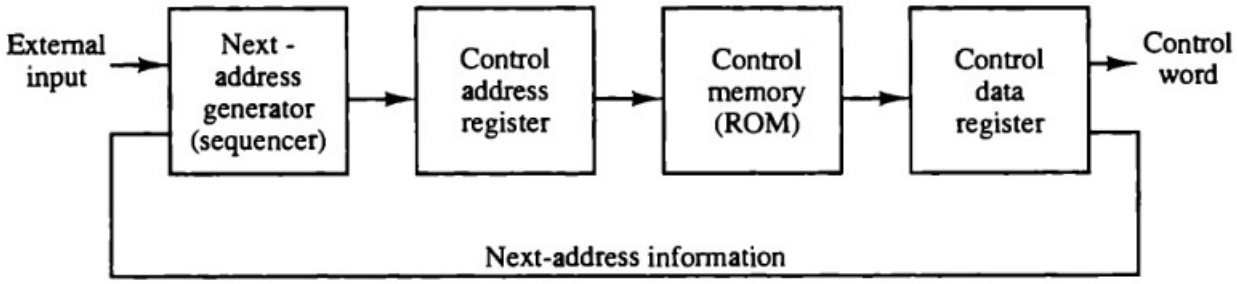
Microprogrammed Instruction

A control unit whose binary control variables are stored in memory is called a *microprogrammed control unit*. Each word in control memory contains within it a *microinstruction*. The microinstruction specifies one or more microoperations for the system. A sequence of microinstructions constitutes a *microprogram*. Since alterations of the microprogram are not needed once the control

Control Memory

change the microprogram) but is used mostly for reading. A memory that is part of a control unit is referred to as a *control memory*.

Figure 7-1 Microprogrammed control organization.



Control unit diagram

Control Address Register

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

Sequencer

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

Pipeline Register

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*. It allows the execution of the microoperations

It should be mentioned that most computers based on the reduced instruction set computer (RISC) architecture concept (see Sec. 8-8) use hardwired control rather than a control memory with a microprogram. An example of a

Routine

Microinstructions are stored in control memory in groups, with each group specifying a *routine*. Each computer instruction has its own microprogram

Mapping

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

Capabilities of Address Sequencing

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.

CAR (Control Address Register)

Control memory address register specifies the address of the micro-instruction, and the control data register holds the micro-instruction read from memory, The micro-instruction contains a control word that specifies one or more micro operations for the data processor.

Special Bits

unit. 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

Branch Logic

The branch logic of Fig. 7-2 provides decision-making capabilities in the control unit. The status conditions are special bits in the system that provide parameter

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.

Microprogrammed Control Unit Diagram

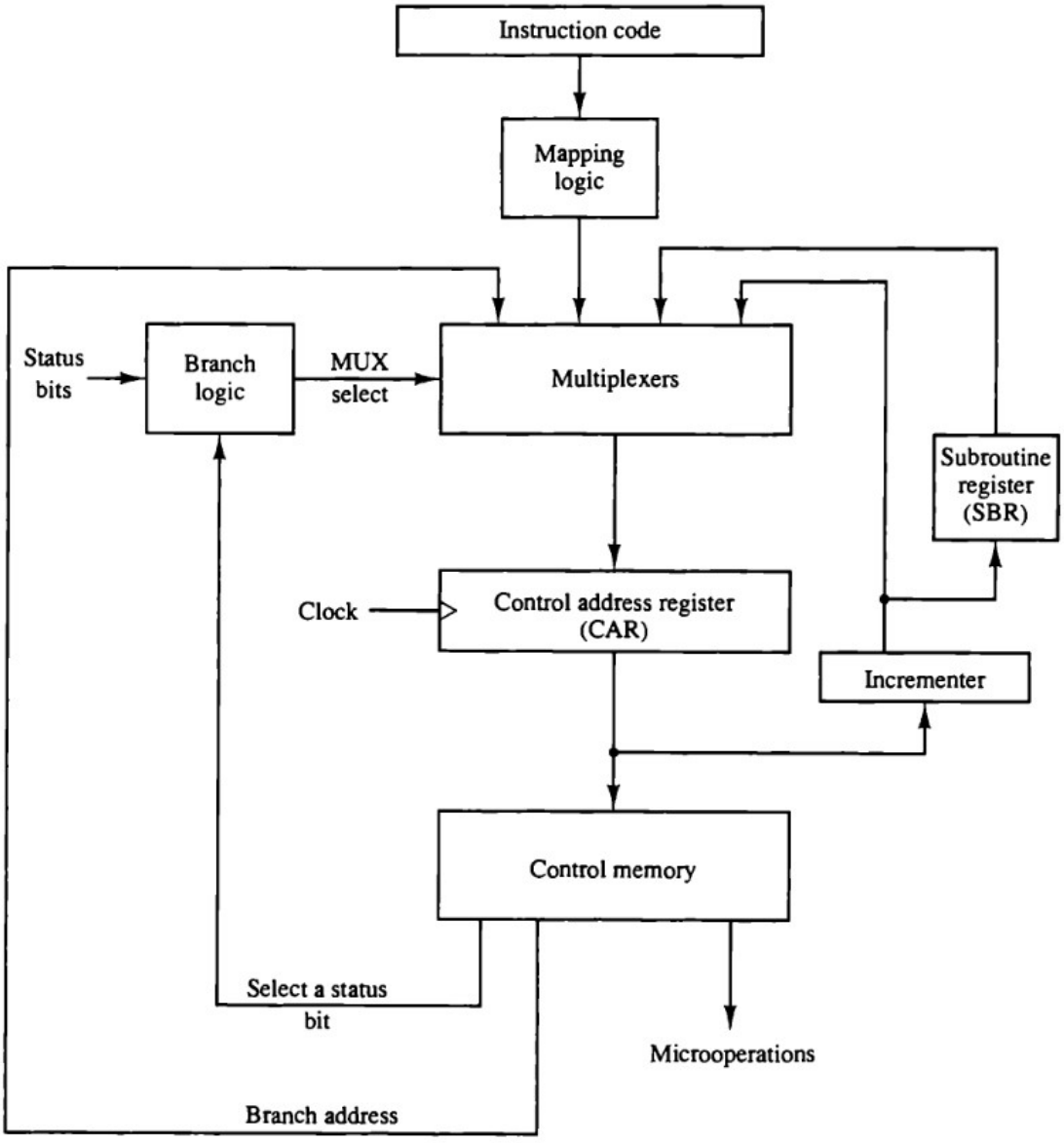
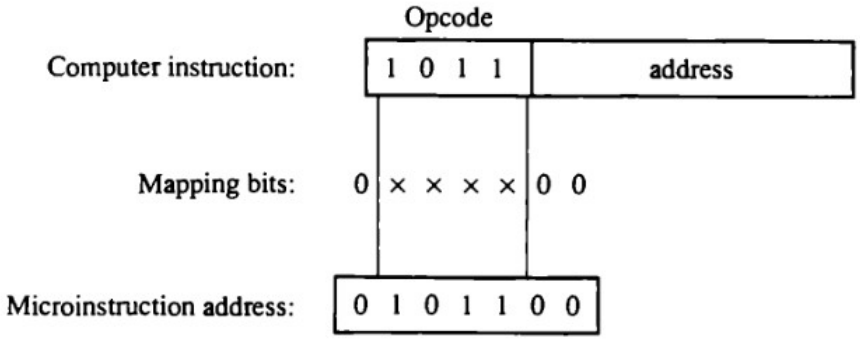


Figure 7-2 Selection of address for control memory.

Mapping of Instruction

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

Figure 7-3 Mapping from instruction code to microinstruction address.



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 microprogram. Frequently, many microprograms contain identical sec-

Subroutine Register

Microprograms 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

Hardware Configuration of Computer

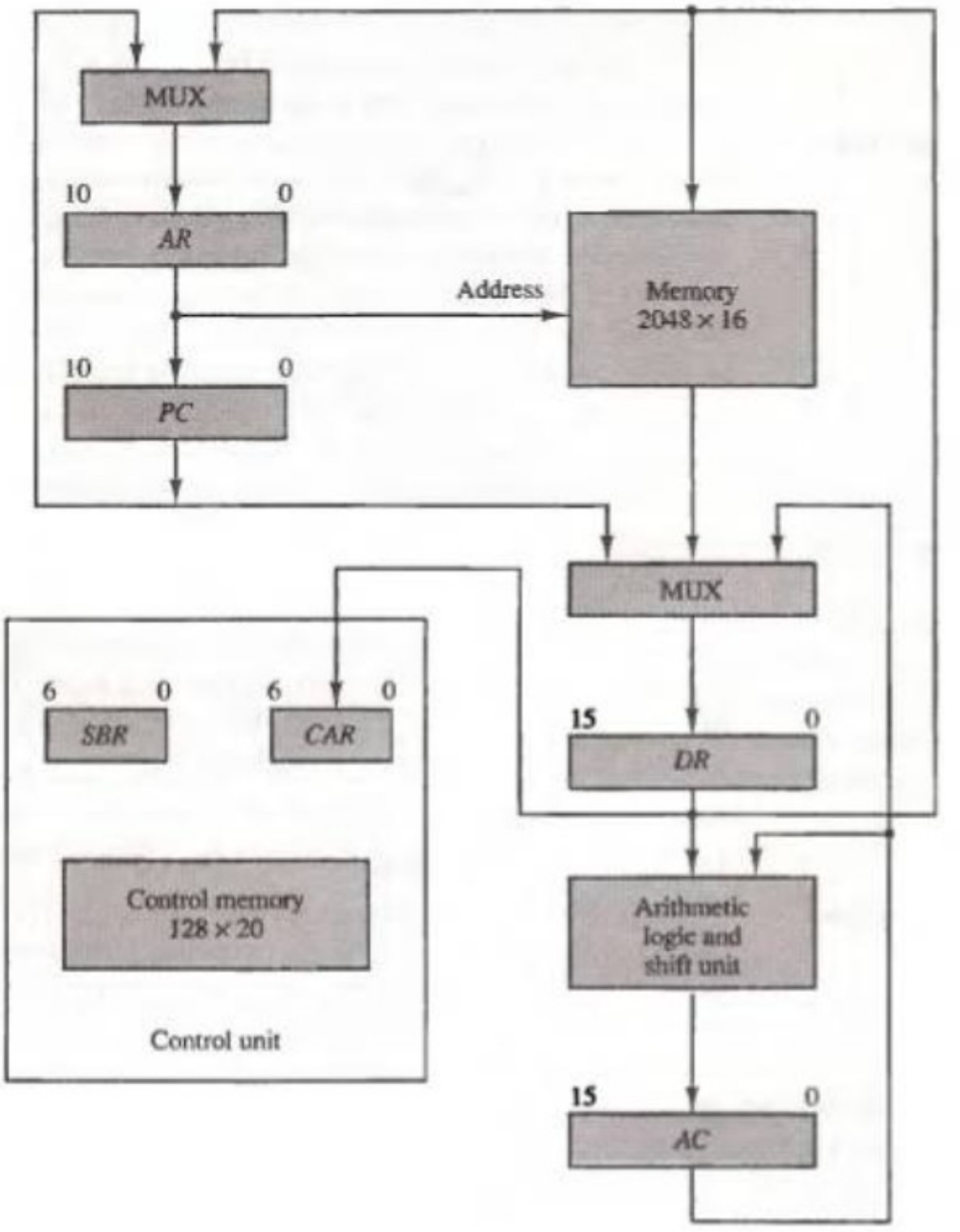
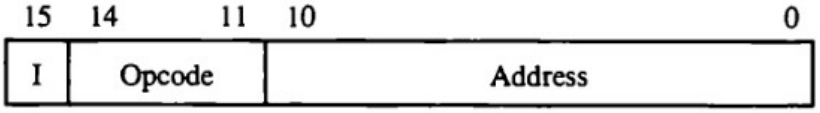


Figure 7-4 Computer hardware configuration.

Microinstruction Format

The microinstruction format for the control memory is shown in Fig. 7-6. The 20 bits of the microinstruction are divided into four functional parts. The three fields F1, F2, and F3 specify microoperations for the computer. The CD field

Figure 7-5 Computer instructions.

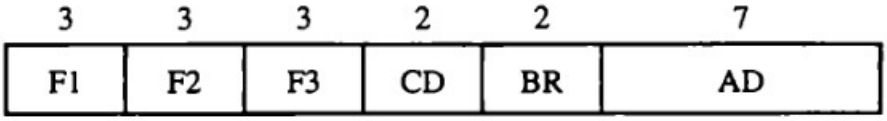


(a) Instruction format

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M\,[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M\,[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

Conditional Field

The *CD* (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table 7-1. The first condition is always a 1, so that a reference to $CD = 00$ (or the symbol U) will always find the condition to be true. When this condition is used in conjunction with the BR (branch) field, it provides an unconditional branch operation. The indirect bit

Branch Field

The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction. As

Symbolic Microinstructions

symbolic form. A symbolic microprogram can be translated into its binary equivalent by means of an assembler. A microprogram assembler is similar in concept to a conventional computer assembler as defined in Sec. 6-3. The simplest and most straightforward way to formulate an assembly language for a microprogram is to define symbols for each field of the microinstruction and to give users the capability for defining their own symbolic addresses.

Fetch Routine

The fetch routine needs three microinstructions, which are placed in control memory at addresses 64, 65, and 66. Using the assembly language conventions defined previously, we can write the symbolic microprogram for the fetch routine as follows:

```
        ORG 64
FETCH:  PCTAR          U    JMP    NEXT
        READ, INCPC    U    JMP    NEXT
        DRTAR          U    MAP
```

ORG64

We will use also the pseudoinstruction ORG to define the origin, or first address, of a microprogram routine. Thus the symbol ORG 64 informs the assembler to place the next microinstruction in control memory at decimal address 64, which is equivalent to the binary address 1000000.

Binary Microprogram

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

Microprogram Sequencer

The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address. The address selection part is called a microprogram sequencer. A microprogram sequencer can be constructed with digital functions to suit a particular application. However, just

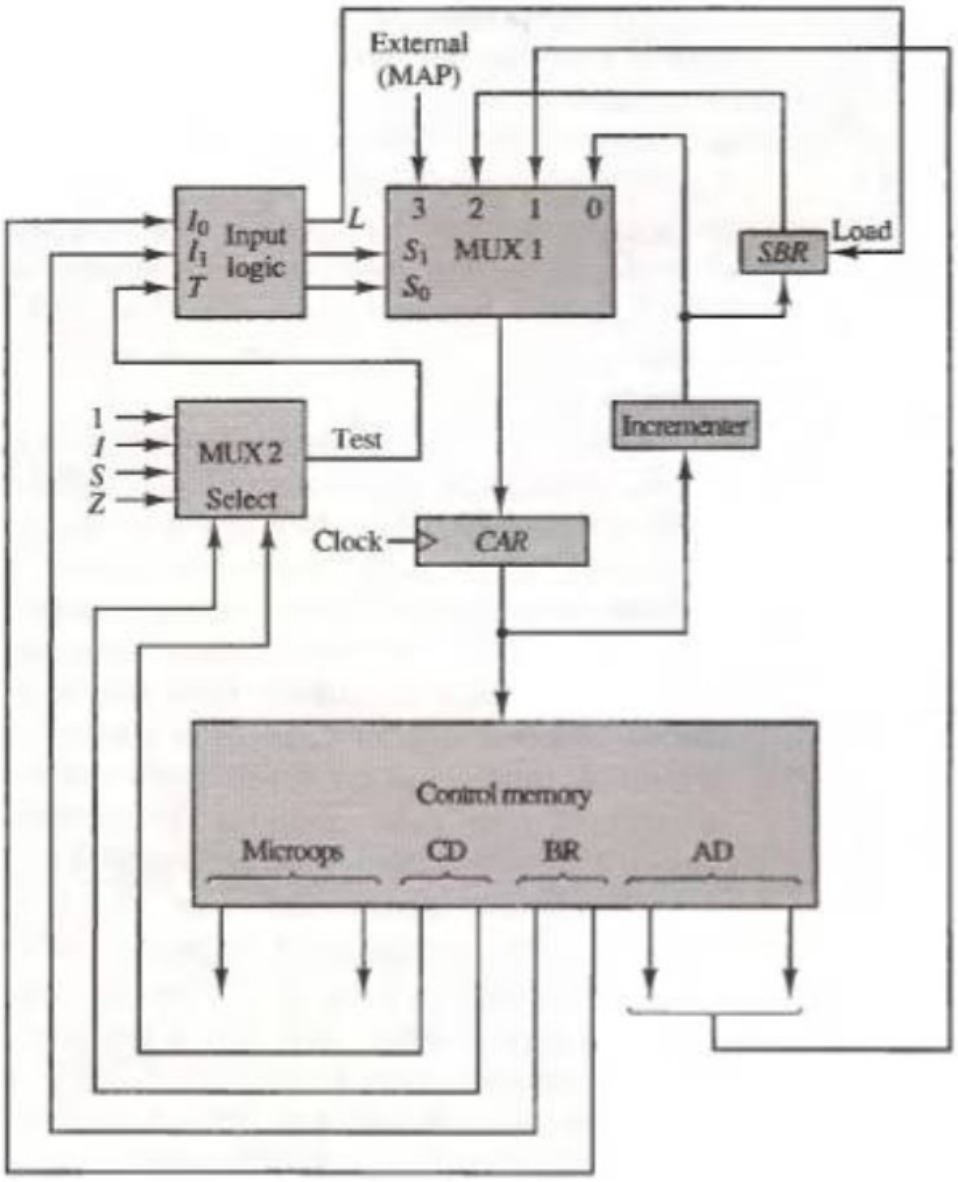


Figure 7-8 Microprogram sequencer for a control memory.

CPU

The part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU. The CPU is made up of three major parts, as shown in Fig. 8-1. The register set stores intermediate data used during the execution of the instructions. The arithmetic logic unit (ALU) performs the required microoperations for executing the instructions. The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

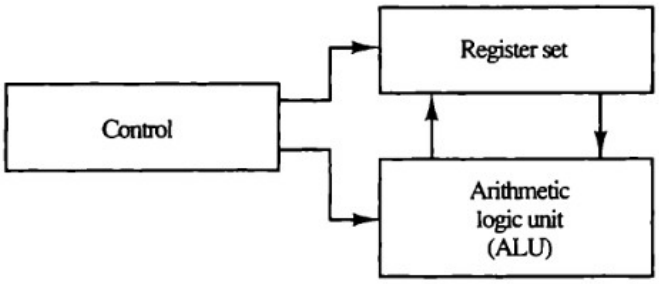


Figure 8-1 Major components of CPU.

Instruction Format

A computer will usually have a variety of instruction code formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.

Register Address

are specified with a register address. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 16 processor registers $R0$ through $R15$ will have a register address field of four bits. The binary number 0101, for example, will designate register $R5$.

Addressing Mode

is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced. Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction.

Control unit during Instruction Cycle

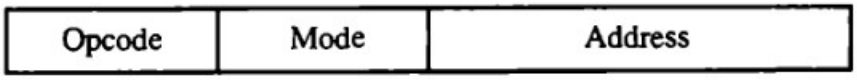
computer. The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Execute the instruction.

Mode Field

formed. The mode field is used to locate the operands needed for the operation. There may or may not be an address field in the instruction. If there is an address field, it may designate a memory address or a processor register.

Figure 8-6 Instruction format with mode field.



CISC Characteristics

1. A large number of instructions—typically from 100 to 250 instructions
2. Some instructions that perform specialized tasks and are used infrequently
3. A large variety of addressing modes—typically from 5 to 20 different modes
4. Variable-length instruction formats
5. Instructions that manipulate operands in memory

RISC Characteristics

1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations done within the registers of the CPU
5. Fixed-length, easily decoded instruction format
6. Single-cycle instruction execution
7. Hardwired rather than microprogrammed control

Machine language instructions

They are the low-level instructions that a computer's central processing unit (CPU) can directly execute. These instructions are represented in binary code, consisting of sequences of 0s and 1s, and they correspond to specific operations that the CPU can perform.

Data Path

This refers to the portion of a processor that is responsible for performing arithmetic and logic operations on data. It is a critical component that facilitates the movement and manipulation of data within a processor. The data path is where actual computation takes place, and it typically includes registers, arithmetic logic units (ALUs), multiplexers, and other components.

Horizontal microprogramming is like using a single recipe card for each step of a cooking process. Each card (microinstruction) contains all the details for a specific action, from chopping vegetables to stirring the soup. Each microinstruction is a standalone set of instructions, like separate recipe cards for each cooking step.

Vertical microprogramming is like having a single recipe card with checkboxes for different tasks. Instead of having separate cards for chopping and stirring, you have one card with checkboxes that you mark as you complete each step. One microinstruction covers multiple tasks, and each task has its dedicated bit or field, akin to a single recipe card with checkboxes.