Module 2

Module II: CPU and Register Transfer Operations: (6 Hours)
Instruction Codes, Computer Registers, Computer Instructions, Register Transfer Language, Timing and Control, Instruction Cycle, Memory, Input-Output and Interrupt Reference Instructions, Signed multiplication, Booth's algorithm. Division of integers: Restoring and non-restoring division Floating point arithmetic: Addition, subtraction.

Instruction Codes -

Instruction codes are binary representations of instructions that the CPU understands and executes. Each instruction in a computer program corresponds to a unique code, often composed of several fields:

Instruction Format

An instruction is typically divided into fields, including:

- **Opcode (Operation Code):** This specifies the operation to be performed (e.g., ADD, SUB, MOV, etc.). The length of the opcode varies depending on the instruction set architecture (ISA).
- **Operand(s):** These are the data or addresses on which the operation is performed. Operands can be registers, memory locations, or immediate values.
- **Addressing Mode (optional):** This specifies how the operand is to be located. Common addressing modes are Immediate, Direct, Indirect, Indexed, and Register.

Instruction Formats

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:

Common Fields in Instruction Format:

The most common fields found in an instruction format include:

1. **Operation Code (Opcode)**: Specifies the operation to be performed.
2. **Address Field**: Designates a memory address or a processor register.
3. **Mode Field**: Specifies the way the operand or the effective address is determined.

Other special fields are sometimes employed under certain circumstances, as for example a field that gives the number of shifts in a shift-type instruction.
The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.

Operations specified by computer instructions are executed on some data stored in memory or processor registers, Operands residing in processor registers are specified with a register address. A register address is a binary number of k bits that defines one of 2k 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.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

1. **Single Accumulator Organization**: Uses an accumulator (AC) register for computations.
    - Example: ADD X → AC ← AC + M[X]
2. **General Register Organization**: Uses multiple general-purpose registers.
    - Example: ADD R1, R2, R3 → R1 ← R2 + R3
3. **Stack Organization**: Uses an implicit stack for operations.
    - Example: PUSH X

All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as ADD.

Where X is the address of the operand. The ADD instruction in this case results in the operation AC ← AC + M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

The instruction format in this type of computer needs three register address fields. Thus the instruction for an arithmetic addition may be written in an assembly language as

ADD R1, R2, R3

To denote the operation R1 ← R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers. Thus the instruction

ADD R1, R2

Would denote the operation R1 ← R1 + R2. Only register addresses for R1 and R2 need be specified in this instruction.

Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction. Thus the instruction

MOV R1, R2

Denotes the transfer R1 ← R2 (or R2 ← R1, depending on the particular computer). Thus transfer-type instructions need two address fields to specify the source and the destination.
General register-type computers employ two or three address fields in their instruction format. Each address field may specify a processor register or a memory word. An instruction symbolized by

ADD R1, X

Would specify the operation R1 ← R + M [X]. It has two address fields, one for register R1 and the other for the memory address X.

Computers with stack organization would have PUSH and POP instructions which require an address field. Thus the instruction

PUSH X

Will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack. The instruction ADD in a stack computer consists of an operation code only with no address field. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. There is no need to specify operands with an address field since all operands are implied to be in the stack.

To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement X = (A + B) * (C + D).

Using zero, one, two, or three address instruction. We will use the symbols ADD, SUB, MUL, and DIV for the four arithmetic operations; MOV for the transfer-type operation; and LOAD and STORE for transfers to and from memory and AC register. We will assume

that the operands are in memory addresses A, B, C, and D, and the result must be stored in memory at address X.

Types of Instruction Formats

1. Three-Address Instruction Format

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) * (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

- Uses three addresses for operands and results.
- Each address field can specify a processor register or memory operand.

Example:

```
ADD R1, A, B    R1 ← M[A] + M[B]
ADD R2, C, D    R2 ← M[C] + M[D]
MUL X, R1, R2   M[X] ← R1 * R2
```

It is assumed that the computer has two processor registers, R1 and R2. The symbol M [A] denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses. An example of a commercial computer that uses three-address instructions is the Cyber 170. The instruction formats in the Cyber computer are restricted to either three register address fields or two register address fields and one memory address field.

Advantages:

- Results in shorter programs for arithmetic expressions.

Disadvantages:

- Requires a large number of bits to encode three addresses.

2. Two-Address Instruction Format

Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) * (C + D) is as follows:

- Most common in commercial computers.
- Each address field can specify a processor register or a memory word.

Example:

MOV R1, A → R1 ← M[A]
ADD R1, B → R1 ← R1 + M[B]
MOV R2, C → R2 ← M[C]
ADD R2, D → R2 ← R2 + M[D]
MUL R1, R2 → R1 ← R1 * R2
MOV X, R1 → M[X] ← R1

The MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

3. One-Address Instruction Format

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second and assume that the AC contains the result of tall operations. The program to evaluate X = (A + B) * (C + D) is

LOAD A → AC ← M[A]
ADD B → AC ← AC + M[B]
STORE T → M[T] ← AC
LOAD C → AC ← M[C]
ADD D → AC ← AC + M[D]
MUL T → AC ← AC * M[T]
STORE X → M[X] ← AC

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

4. Zero-Address Instruction Format

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) * (C + D) will be written for a stack organized computer. (TOS stands for top of stack)

```
PUSH A   → TOS ← A
PUSH B   → TOS ← B
ADD      → TOS ← (A + B)
PUSH C   → TOS ← C
PUSH D   → TOS ← D
ADD      → TOS ← (C + D)
MUL      → TOS ← (C + D) * (A + B)
POP X    → M[X] ← TOS
```

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

Summary of Instruction Formats

Format	**Address Fields**	**Example**
Three-Address	3	ADD R1, A, B
Two-Address	2	ADD R1, B
One-Address	1	ADD B (AC implicit)
Zero-Address	0	ADD (stack-based)

This classification of instruction formats helps in understanding how different CPU architectures handle instruction execution efficiently.

Instruction Codes

A set of instructions that specify the operations, operands, and the sequence by which processing has to occur. An instruction code is a group of bits that tells the computer to perform a specific operation part.

Format of Instruction

The format of an instruction is depicted in a rectangular box symbolizing the bits of an instruction. Basic fields of an instruction format are given below:

1. An operation code field that specifies the operation to be performed.

2. An address field that designates the memory address or register.

3. A mode field that specifies the way the operand of effective address is determined.

Computers may have instructions of different lengths containing varying number of addresses. The number of address field in the instruction format depends upon the internal organization of its registers.

Addressing Modes

To understand the various addressing modes to be presented in this section, it is imperative that we understand the basic operation cycle of the 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.

There is one register in the computer called the program counter of PC that keeps track of the instructions in the program stored in memory. PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory. The decoding done in step 2 determines the operation to be performed, the addressing mode of the instruction and the location of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.

In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified. Other computers use a single binary code that designates both the operation and the mode of the instruction. Instructions may be defined with a variety of addressing modes, and sometimes, two or more addressing modes are combined in one instruction.

1. The operation code specified the operation to be performed. The mode field is sued 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. Moreover, as discussed in the preceding section, the instruction may have more than one address field, and each address field may be associated with its own particular addressing mode.
Although most addressing modes modify the address field of the instruction, there are two modes that need no address field at all. These are the implied and immediate modes.

1 Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that sue an accumulator are implied-mode instructions.

Op code Mode Address

Instructions since the operands are implied to be on top of the stack.

2 Immediate Mode: In this mode the operand is specified in the instruction itself. In other words, an immediate- mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate-mode instructions are useful for initializing registers to a constant value.

It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register. When the address field specifies a processor register, the instruction is said to be in the register mode.

3 Register Mode: In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2k registers.

4 Register Indirect Mode: In this mode the instruction specifies a register in the CPUwhose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address fo the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction sues fewer bits to select a register than would have been required to specify a memory address directly.

5 Auto increment or Auto decrement Mode: This is similar to the register indirect modeexcept that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction. However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access.
The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory. Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated. To differentiate among the various addressing modes it is necessary to distinguish between the address part of the instruction and the effective address used by the control when executing the instruction. The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational-type instruction. It is the address where control branches in response to a branch-type instruction. We have already defined two addressing modes in previous chapter.

6 Direct Address Mode: In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address.

7 Indirect Address Mode: In this mode the address field of the instruction gives theaddress where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

8 Relative Address Mode: In this mode the content of the program counter is added to theaddress part of the instruction in order to obtain the effective address. The address part of the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction. To clarify with an example, assume that the program counter contains the number 825 and the address part of the instruction contains the number 24. The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826 + 24 = 850. This is 24 memory locations forward from the address of the next instruction. Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself. It

results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory address.

9 Indexed Addressing Mode: In this mode the content of an index register is added to theaddress part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stores in the index register. Any operand in the array can be accessed with the same instruction provided that the
index register contains the correct index value. The index register can be incremented to facilitate access to consecutive operands. Note that if an index-type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation. Some computers dedicate one CPU register to function solely as an index register. This register is involved implicitly when the index-mode instruction is used. In computers with many processor registers, any one of the CPU registers can contain the index number. In such a case the register must be specified explicitly in a register field within the instruction format.

10 Base Register Addressing Mode: In this mode the content of a base register is added tothe address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register. The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction. A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of programs in memory. When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of the base register requires updating to reflect the beginning of a new memory segment.

Numerical Example-

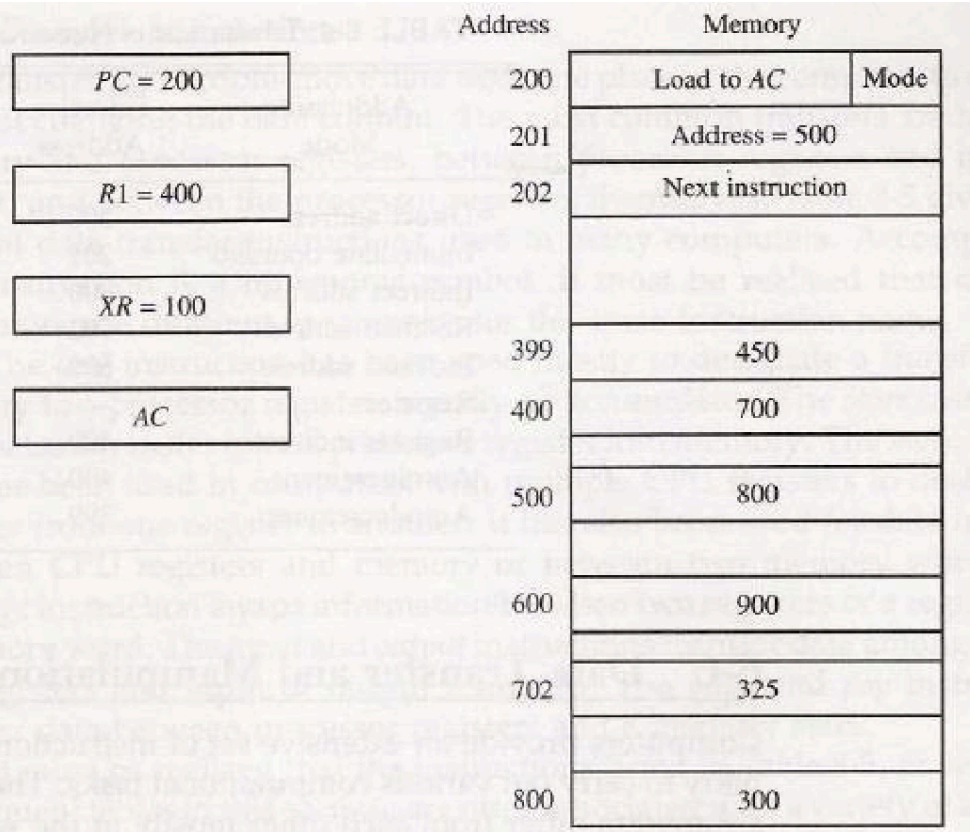


Figure 8-7 Numerical example for addressing modes.

TABLE 8-4 Tabular List of Numerical Example

Addressing Mode	Effective Address	Content of *AC*
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Computer Registers

- Data Register(**DR**) : hold the operand(Data) read from memory
- Accumulator Register(**AC**) : general purpose processing register
- Instruction Register(**IR**) : hold the instruction read from memory
- Temporary Register(**TR**) : hold a temporary data during processing
- Address Register(**AR**) : hold a memory address, 12 bit width
- Program Counter(**PC**) :

»hold the address of the next instruction to be read from memory after the current instruction is executed
»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
Input Register(INPR) : receive an 8-bit character from an input device

- Output Register(**OUTR**) : hold an 8-bit character for an output device

The following registers are used in Mano's example computer.

Register symbol	Number of bits	Register name	Register Function
DR	16	Data register	Holds memory operands
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

Computer Instructions

A basic computer uses a **16-bit Instruction Register (IR)** to store instructions. These instructions can belong to one of the following three categories:

1. **Memory Reference Instructions**
   - These instructions access memory to fetch an operand.
   - The second operand is always the **Accumulator (AC)**.
   - The instruction format consists of:
     - **12-bit memory address**
     - **3-bit opcode** (other than 111)
     - **1-bit addressing mode** (direct or indirect)

   Example: Suppose the **IR register** contains:

   0001XXXXXXXXXXX

   This means the instruction is **ADD**. After fetching and decoding, the system performs:

   DR ← M[AR] (Fetch data from memory to Data Register)
   AC ← AC + DR (Add data to Accumulator)
   SC ← 0 (Reset the sequence counter)

2. **Register Reference Instructions**

- These instructions operate on **registers** instead of memory.
- The instruction is identified when:
  - **IR(14-12) = 111** (indicates it is not a memory reference instruction)
  - **IR(15) = 0** (indicates it is not an I/O instruction)
- The remaining **12 bits** define the operation on registers.

Example: If **IR register** contains:

0111001000000000

This represents the **CMA** (Complement Accumulator) instruction. After execution:

AC ← ~AC (Invert all bits of the accumulator)

3. **Input/Output (I/O) Instructions**
   - These instructions enable communication between the computer and external devices.
   - The instruction is identified when:
     - **IR(14-12) = 111** (not a memory reference instruction)
     - **IR(15) = 1** (not a register reference instruction)
   - The remaining **12 bits** specify the I/O operation.

Example: If **IR register** contains:

1111100000000000

This represents the **INP** (Input) instruction. After execution:

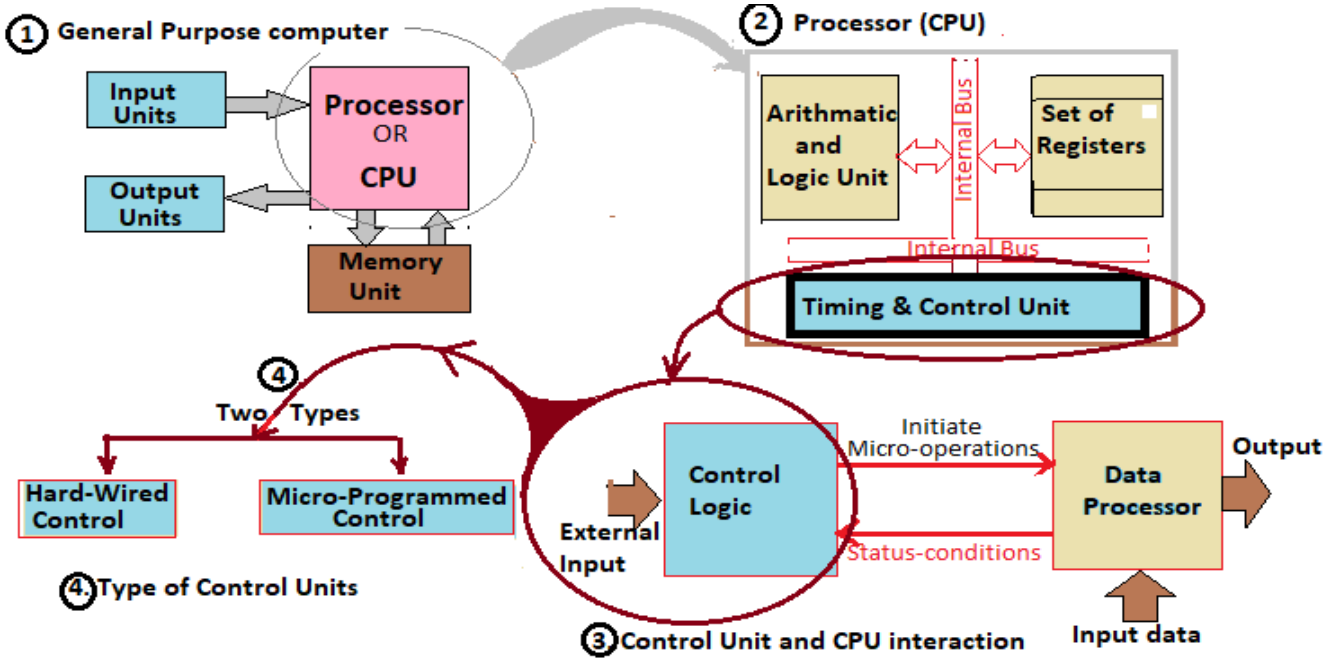
INPUT character from an external device and store it.

This classification helps in efficiently decoding and executing instructions in a basic computer system.

Timing and Control -

A general purpose computer consists of input-output units, processing unit and a memory unit. Input-output permits interaction with the physical world whereas the memory unit stores user program and the binary information.

The functional part of general purpose digital computer, its functional units, the interaction between the control and the types of the control logic.



All sequential circuits in the Basic Computer CPU are driven by a master clock, with the exception of the INPR register. At each clock pulse, the control unit sends control signals to control inputs of the bus, the registers, and the ALU.

Basic function of the control logic-

- To generate timing sequences.
- To take input from the external sources and initiates the sequence of micro-operations for the processor to perform.
- General interaction between the processor and the control logic.

Types of control logic-

Control unit design and implementation can be done by two general methods:

1. Hardwired control logic
2. Micro-programmed control logic

Instruction from memory are read into IR, decoded in the control unit; now the control unit will generate a binary control variable which is a string of 1`s and 0`s which is called a control word. Control words can be generated by logic hardware or can be programmed to perform various operations on the components of a system.

1. **Hard-wired control:**

A hardwired control unit is designed from scratch using traditional digital logic design techniques to produce a minimal, optimized circuit. In other words, the control unit is like an ASIC (application-specific integrated circuit).

Hardwired control is a control mechanism to generate control signals by using appropriate finite state machine (FSM). The pair of "microinstruction-register" and "control storage address register" can be regarded as a "state register" for the hardwired control. Note that the control storage can be regarded as a kind of combinational logic circuit. We can assign any 0, 1 value to each output corresponding to each address, which can be regarded as the input for a combinational logic circuit. This is a truth table.

Figure below shows a simplified block diagram of the hardwired control unit. The control unit is built using the logic gates and is designed as a sequential state machine.

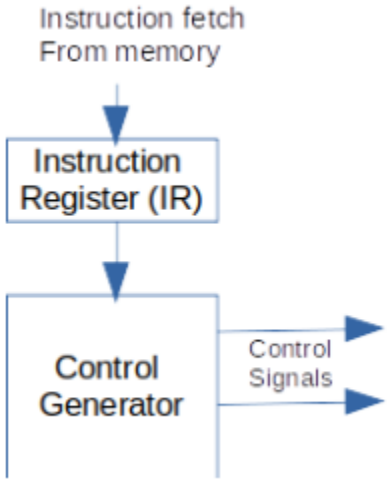


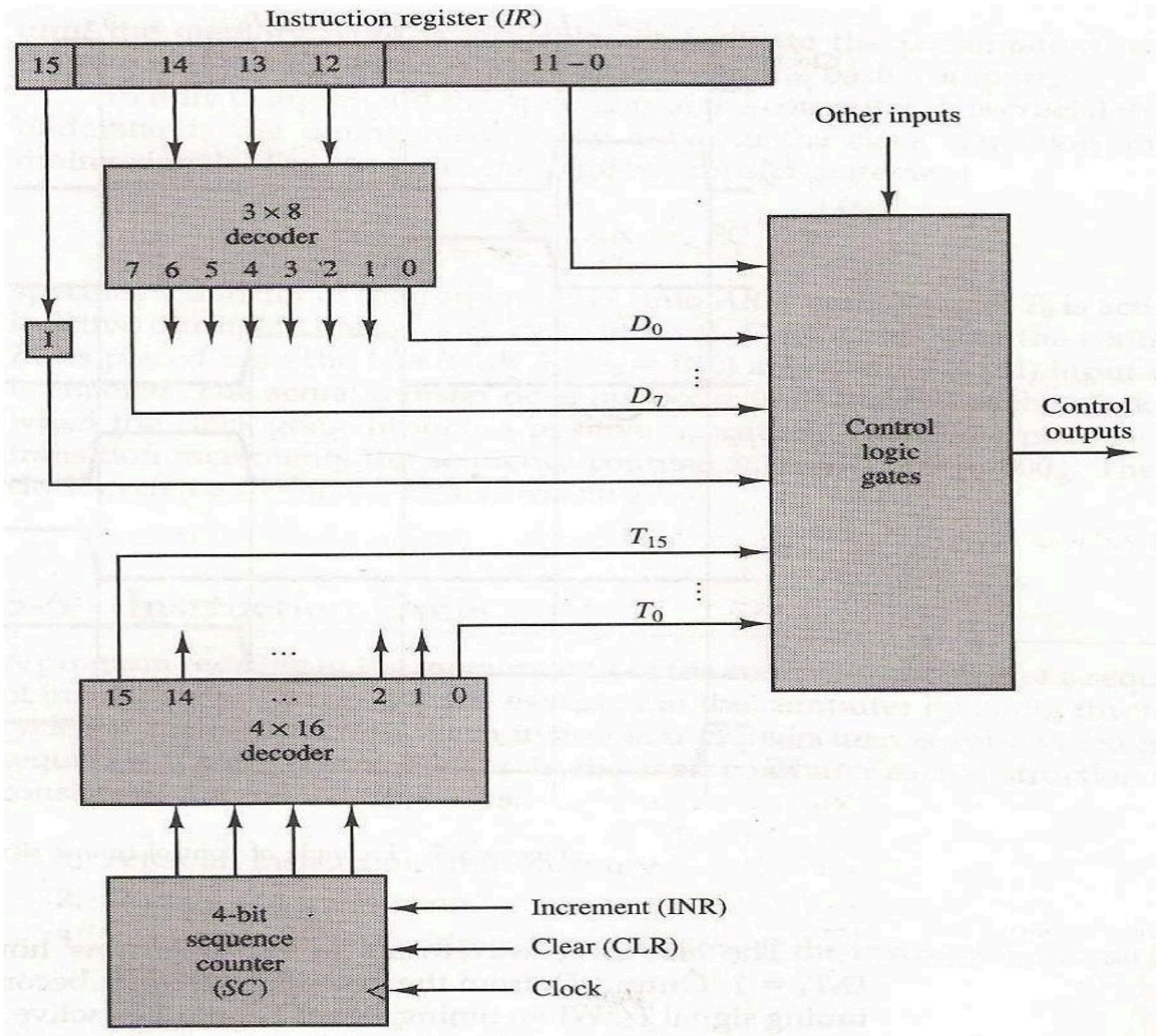
Figure : simplified diagram of hardwired control unit

The instructions are passed to the control unit for decoding, and the control unit generates a set of micro-operations for each instruction. These micro-operations control the internal operation of the CPU.

Example of hardwired control unit-

The block diagram of a hardwired control unit is shown below. It consists of;

- Instruction registers,
- 3*8 instruction decoder,
- flag I to store the addressing mode bit,
- Control logic,
- 4*16 Sequence decoder,
- 4-bit sequence counter

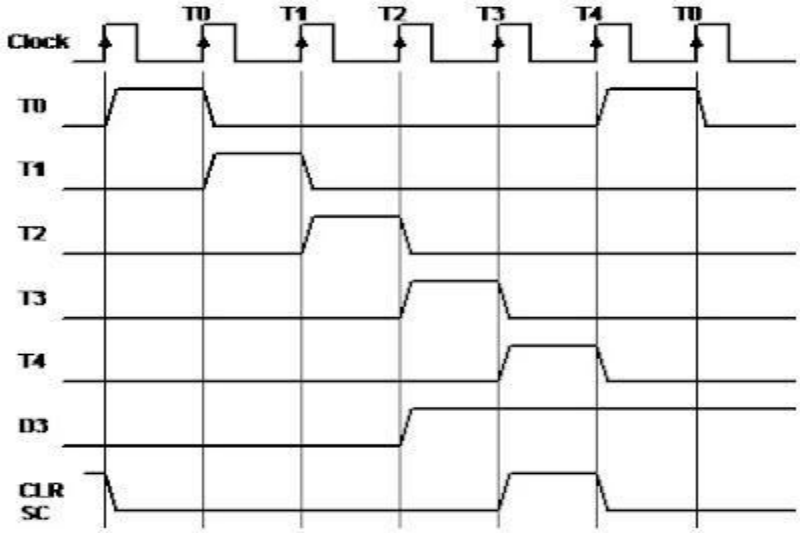


Mechanism:

- An instruction fetched from memory, it is transferred into the instruction resister (IR) where it is decoded into three parts: I bit, operation code and bits 0 through 11.
- The operation code bit is decoded with 3 x 8 decoder producing 8 outputs D0 through D7.
- bit 15 of the instruction is transferred to a flip-flop I.
- And operand bits are applied to control logic gates.
- The 16 outputs of 4-bit sequence counter (SC) are decoded into 16 timing signals T0 through T15.

This means instruction cycle of basic computer cannot take more than 16 timing pulse.

Timing and control Signals generated:



- Generated by 4-bit sequence counter and 4x16 decoder.
- The SC can be incremented or cleared.
- Example: T0, T1, T2, T3, T4, T0, T1 . . .

The sequence counter generates the precisely controlled T0 to T15 a total of 16 timing pulses. As an operation say memory operation completes in 4T states, it is required to clear the sequence counter so that another operation can be initiated.

The logic for clearing the SC:

D3T4: SC □ 0

The timing diagram shows this micro-operation.

D3 becomes active at the end of T2.

D3.T4 is ended to produce a low CLRSC at the end of T4.

Clear SC□ 0 at the end of T4.

- Assume: At time T4, SC is cleared to 0 if decoder output D3 is active: D3T4: SC □ 0

Micro programmed control:

A control word can be programmed to perform various operations on components of the system. A control unit whose control word is stored in the control memory which is usually a ROM in called as micro-program control unit. Each control

word in the control memory contains within it a microinstruction. Every microinstruction specifies one or more micro-operations.

A micro-program is written for every instruction supported by the CPU. Each instruction of a program cause the corresponding micro-program to be fetched and its control information extracted in a manner that resembles the fetching and execution of a program from the main memory. Since the control signals are embedded into a kind of low-level software- this is also referred as firmware. The general feature of MCU is:

- One time programmable, programmed during manufacture
- We can change its functionality, but generally not required once designed
- Reusability-code

Micro programmed control is a control mechanism to generate control signals by using a memory called control storage (CS), which contains the control signals. Although micro programmed control seems to be advantageous to CISC machines, since CISC requires systematic development of sophisticated control signals, there is no intrinsic difference between these 2 control mechanisms.

Difference between Hardwired control and Micro-programmed control units

Attributes	Hardwired Control	Micro-programmed Control
No. of Instructions	Uses Fixed Instruction	Uses Variable or large instruction set
Design Logic	Binary control word is generated using fixed logic blocks – Logic gates, MUX, decoders, FF etc.	Binary control word are stored in a control memory.
Speed	High Speed of Operation	Comparatively Slow
Cost	Expensive	Inexpensive
Flexibility	Not Flexible for adding new features	New features can easily be incorporated
Design Complexity	Relatively complex design if more functions are to be controlled	Design Complexity is less when compared with hardwired control
Chip Area	Small	Large
Applications	Used in RISC Processors suc as ARM, PA-RISC, Power Architecture, Alpha, AVR, ARC and the SPARC.	Used in CISC Processors such as : Examples of CISC: VAX, Motorola 68000 family, System/360, AMD and the Intel x86 CPUs.

Instruction Cycle

The CPU performs a sequence of microoperations for each instruction. The sequence for each instruction of the Basic Computer can be refined into 4 abstract phases:

1. Fetch instruction
2. Decode
3. Fetch operand
4. Execute

Program execution can be represented as a top-down design:

1. Program execution
    a. Instruction 1
        i. Fetch instruction
        ii. Decode
        iii. Fetch operand
        iv. Execute
    b. Instruction 2
        i. Fetch instruction
        ii. Decode
        iii. Fetch operand
        iv. Execute
    c. Instruction 3 ...

Program execution begins with:
PC ← address of first instruction, SC ← 0

After this, the SC is incremented at each clock cycle until an instruction is completed, and then it is cleared to begin the next instruction. This process repeats until a HLT instruction is executed, or until the power is shut off.

Instruction Fetch and Decode

The instruction fetch and decode phases are the same for all instructions, so the control functions and microoperations will be independent of the instruction code.

Everything that happens in this phase is driven entirely by timing variables T_0, T_1 and T_2. Hence, all control inputs in the CPU during fetch and decode are functions of these three variables alone.

T_0: AR ← PC
T_1: IR ← M[AR], PC ← PC + 1
T_2: D_{0-7} ← decoded IR(12-14), AR ← IR(0-11), I ← IR(15)

For every timing cycle, we assume SC ← SC + 1 unless it is stated that SC ← 0.

Micro Programmed Control:

Control Memory

- The control unit in a digital computer initiates sequences of microoperations
- The complexity of the digital system is derived form the number of sequences that are performed
- When the control signals are generated by hardware, it is hardwired
- In a bus-oriented system, the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and ALUs.
- The control unit initiates a series of sequential steps of microoperations
- The control variables can be represented by a string of 1's and 0's called a control word
- A microprogrammed control unit is a control unit whose binary control variables are stored in memory
- A sequence of microinstructions constitutes a microprogram
- The control memory can be a read-only memory
- Dynamic microprogramming permits a microprogram to be loaded and uses a writable control memory
- A computer with a microprogrammed control unit will have two separate memories: a main memory and a control memory
- The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations
- These microinstructions generate the microoperations to:
    - fetch the instruction from main memory
    - evaluate the effective address
    - execute the operation
    - return control to the fetch phase for the next instruction

- The control memory address register specifies the address of the microinstruction
- The control data register holds the microinstruction read from memory
- The microinstruction contains a control word that specifies one or more microoperations for the data processor
- The location for the next microinstruction may, or may not be the next in sequence

- Some bits of the present microinstruction control the generation of the address of the next microinstruction
- The next address may also be a function of external input conditions
- While the microoperations are being executed, the next address is computed in the next address generator circuit (sequencer) and then transferred into the CAR to read the next microinstructions
- Typical functions of a sequencer are:
- incrementing the CAR by one
- loading into the CAR and address from control memory
- transferring an external address
- loading an initial address to start the control operations
- A clock is applied to the CAR and the control word and next-address information are taken directly from the control memory
- The address value is the input for the ROM and the control work is the output
- No read signal is required for the ROM as in a RAM

- The main advantage of the microprogrammed control is that once the hardware configuration is established, there should be no need for h/w or wiring changes
- To establish a different control sequence, specify a different set of microinstructions for control memory

Address Sequencing

- Microinstructions are stored in control memory in groups, with each group specifying a routine
- Each computer instruction has its own microprogram routine to generate the microoperations
- 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
- Steps the control must undergo during the execution of a single computer instruction:
    - Load an initial address into the CAR when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction fetch routine – IR holds instruction
    - The control memory then goes through the routine to determine the effective address of the operand – AR holds operand address
    - The next step is to generate the microoperations that execute the instruction by considering the opcode and applying a mapping

    - After execution, control must return to the fetch routine by executing an unconditional branch
- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained
- Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition
- 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 i/o status conditions
- The status bits, together with the field in the microinstruction that specifies a branch address, control the branch logic
- The branch logic tests the condition, if met then branches, otherwise, increments the CAR

- If there are 8 status bit conditions, then 3 bits in the microinstruction are used to specify the condition and provide the selection variables for the multiplexer
- For unconditional branching, fix the value of one status bit to be one load the branch address from control memory into the CAR
- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine is located
- The status bits for this type of branch are the bits in the opcode
- Assume an opcode of four bits and a control memory of 128 locations
- The mapping process converts the 4-bit opcode to a 7-bit address for control memory
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions
- Subroutines are programs that are used by other routines to accomplish a particular task and can be called from any point within the main body of the microprogram
- Frequently many microprograms contain identical section of code
- Microinstructions can be saved by employing subroutines that use common sections of microcode
- Microprograms that use subroutines must have a provisions for storing the return address during a subroutine call and restoring the address during a subroutine return
- A subroutine register is used as the source and destination for the addresses .