Subject: Computer Organization and Architecture

Register Transfer and Operations

Table of Content:

Register Transfer and Operations

Computer Organization

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

Register Transfer and Operations

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 own particular interpretation.

1) Operation Part
2) Address Part

Register Transfer and Operations

OPERATION PART

- The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.
- The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.
- The operation code must consist of at least n bits for a given 2^n (or less) distinct operations.

Register Transfer and Operations

EXAMPLE

- Consider a computer with 64 distinct operations, one of them being an ADD operation.
- The operation code consists of six bits, with a bit configuration 110010 assigned to the ADD operation .
- When this operation code is decoded in the control unit, the computer issues control signals to read an operand from memory and add the operand to a processor register.

Register Transfer and Operations

ADDRESS PART

- An instruction code must therefore specify the registers or the memory words where the operands are to be found, as well as the register or memory word where the result is to be stored.

- Memory words can be specified in instruction codes by their address.
- Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2^k registers.

Register Transfer and Operations

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

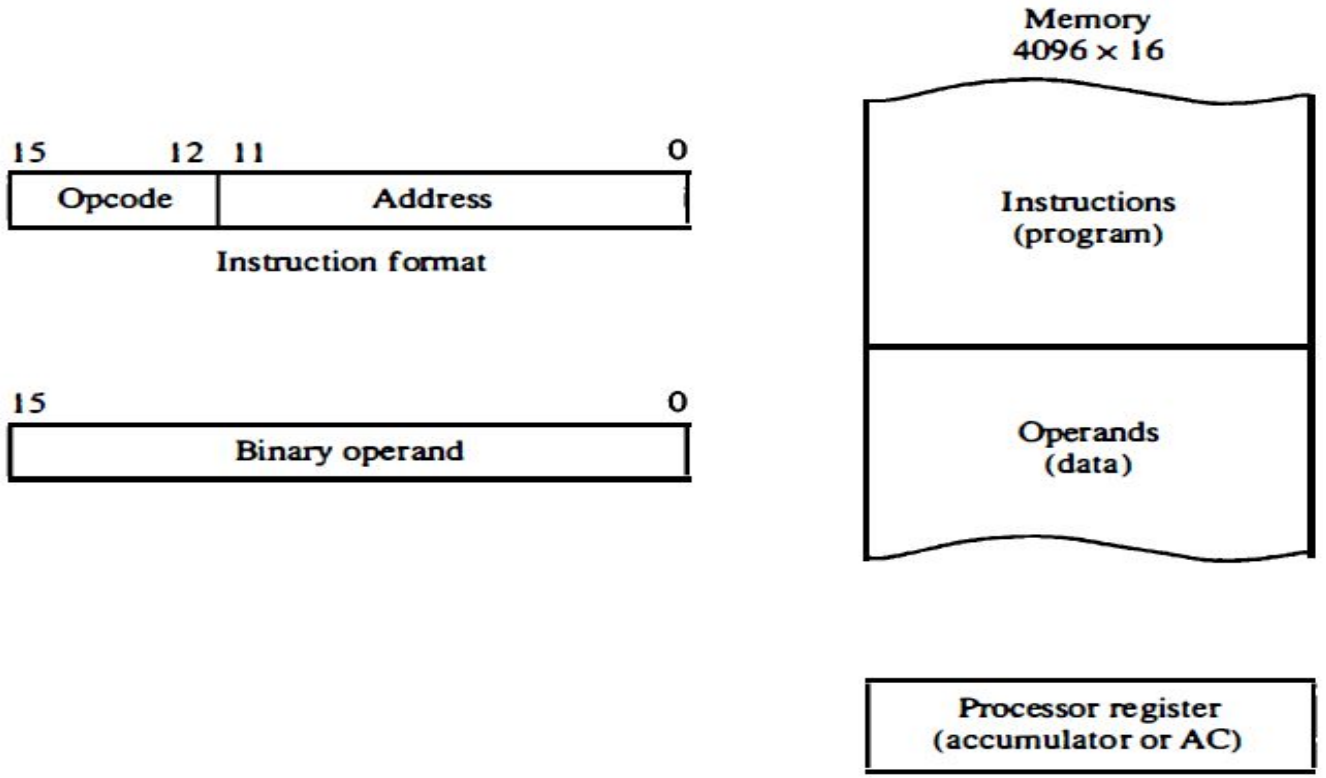
  specifies an address.

- 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

Register Transfer and Operations

Stored Program Organization

Figure 5-1 Stored program organization.



Register Transfer and Operations

Stored Program Organization

- Instructions are stored in

  one section of memory and

  data in another section of the memory.

- For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$.

- If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.

- The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read a 16 bit operand from the data portion.

Register Transfer and Operations

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 .

Register Transfer and Operations

- If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes.
- For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register. They do not need an operand from memory.
- For these types of operations, the second part of the instruction code (bits 0 through 11) is not needed for specifying a memory address and can be used to specify other operations for the computer.

Register Transfer and Operations

Indirect Address

- We can 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.
- 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.

Register Transfer and Operations

Demonstration of Direct and Indirect Address

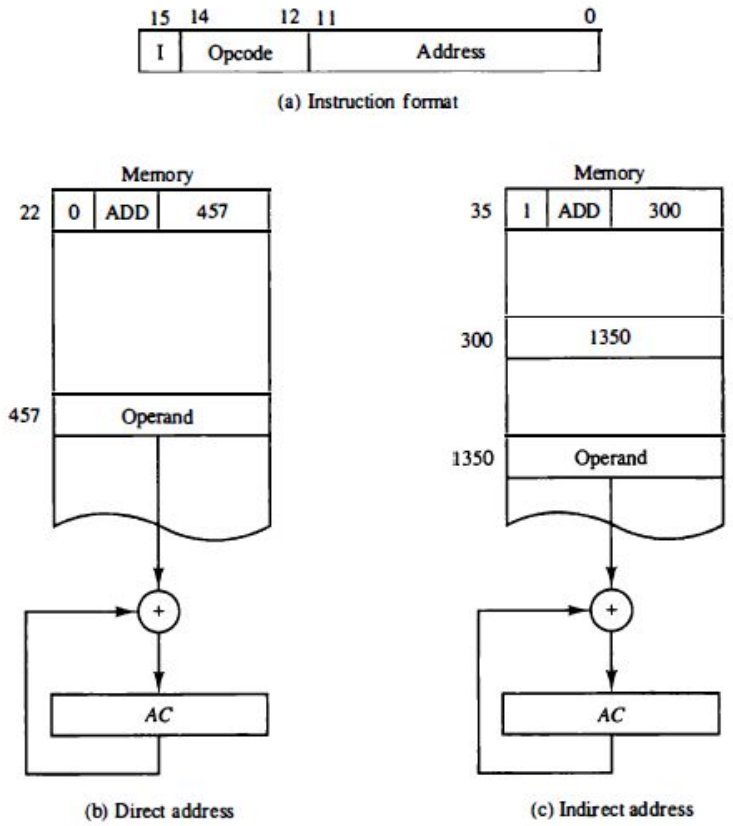


Figure 5-2 Demonstration of direct and indirect address.

Register Transfer and Operations

EFFECTIVE ADDRESS

- The address of the operand is called as an effective address.

Register Transfer and Operations

COMPUTER REGISTERS

- Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time.
- The control reads an instruction from a specific address in memory and executes it.
- It then continues by reading the next instruction in sequence and executes it, and so on.
- This type of instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed.
- It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory.

Register Transfer and Operations

- The computer needs processor registers for manipulating data and a register for holding a memory address.
- These requirements dictate the register configuration shown in Figure.

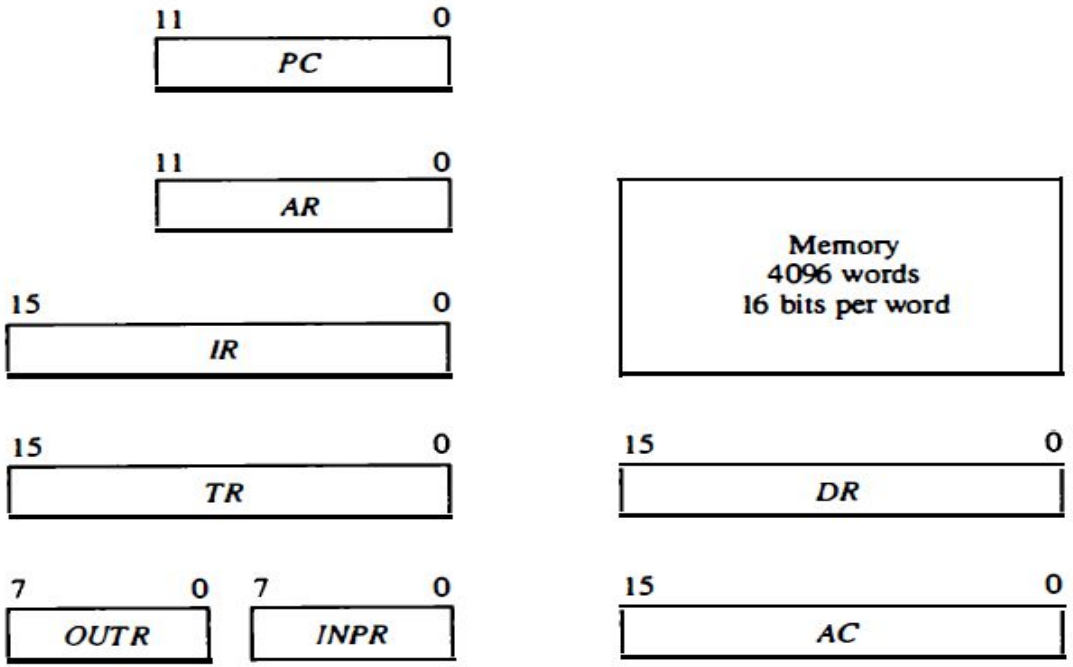


Figure 5-3 Basic computer registers and memory.

Register Transfer and Operations

The registers are also listed in Table together with a brief description of their function and the number of bits that they contain.

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

Register Transfer and Operations

- The memory unit has a capacity of 4096 words and each word contains 16 bits.
- Twelve bits of an instruction word are needed to specify the address of an operand. This leaves three bits for the operation part of the instruction and a bit to specify a direct or indirect address.
- The data register (DR) holds the operand read from memory.
- The accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the instruction register (IR).
- The temporary register (TR) is used for holding temporary data during the processing.

Register Transfer and Operations

- 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 non-consecutive instruction in the program. The address part of a branch instruction is transferred to PC to become the address of the next instruction.

Register Transfer and Operations

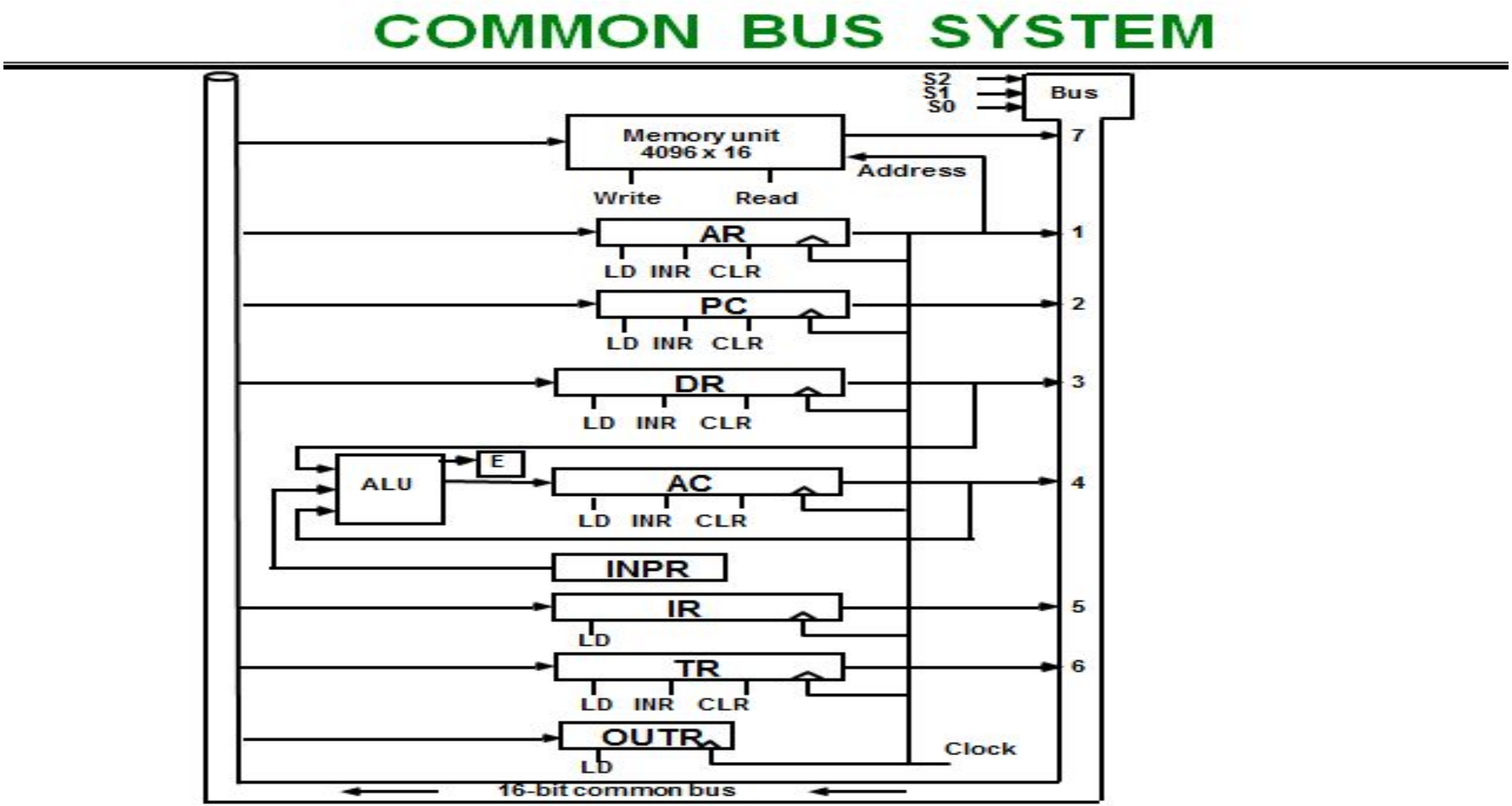
- To read an instruction, the content of PC is taken as the address for memory and a memory read cycle is initiated.
- PC is then incremented by one, so it holds the address of the next instruction in sequence.
- Two registers are used for input and output.
- The input register (INPR) receives an 8-bit character from an input device.
- The output register (OUTR) holds an 8-bit character for an output device.

Register Transfer and Operations

Common Bus System

- The basic computer has
  - eight registers,
  - a memory unit,
  - and a control unit.
- Paths must be provided to transfer information from one register to another and between memory and registers.
- The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers.
- A more efficient scheme for transferring information in a system with many registers is to use a common bus.
- We have seen how to construct a bus system using multiplexers or three-state buffer gates. The connection of the registers and memory of the basic computer to a common bus system is shown in Figure.

Register Transfer and Operations



Register Transfer and Operations

- The outputs of seven registers and memory are connected to the common bus.
- The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S2, S1, and S0.
- The number along each output shows the decimal equivalent of the required binary selection.
- For example, the number along the output of DR is 3. The 16-bit outputs of DR are placed on the bus lines when S2S1S0 = 011.
- since this is the binary value of decimal 3. The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.
- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition.
- The memory receives the contents of the bus when its write input is activated.
- The memory places its 16-bit output onto the bus when the read input is activated and S2S1S0 = 1 1 1 .

Register Transfer and Operations

- Four registers, D R , AC, IR, and TR, have 16 bits each. Two registers, AR and PC, have 1 2 bits each since they hold a memory address.
- When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to 0's.
- When AR or PC receive information from the bus, only the 12 least significant bits are transferred into the register.
- The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus.
- INPR is connected to provide information to the bus but OUTR can only receive information from the bus.

Register Transfer and Operations

- This is because INPR receives a character from an input device which is then transferred to AC .
- OUTR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers.
- The 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.

Register Transfer and Operations

- Five registers have three control inputs:
  - LD (load),
  - INR (increment), and
  - CLR (clear).

  The increment operation is achieved by enabling the count input of the counter.
- Two registers have only a LD input.
- The input data and output data of the memory are connected to the common bus, but the memory address is connected to AR .

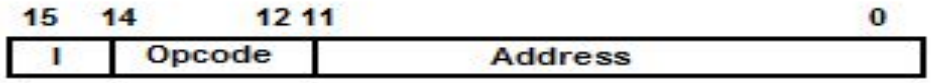
Register Transfer and Operations

- The content of any register can be specified for the memory data input during a write operation.
- Similarly, any register can receive the data from memory after a read operation except AC.
- The 16 inputs of AC come from an adder and logic circuit.
- ALU has three sets of inputs.
- First set- outputs of AC -implement register Micro operations - complement AC and shift AC .
- Another set - data register DR - arithmetic and logic Micro operations -add DR to AC or AND DR to AC .
- A third set - of 8-bit inputs- input register INPR .

BASIC COMPUTER INSTRUCTIONS

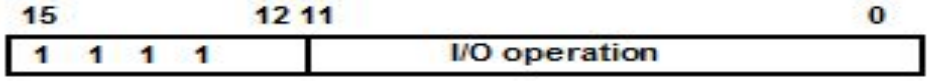
- **Basic Computer Instruction Format**

Memory-Reference Instructions (OP-code = 000 ~ 110)



Register-Reference Instructions (OP-code = 111, I = 0)

Input-Output Instructions (OP-code =111, I = 1)



Register Transfer and Operations

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

INSTRUCTION SET COMPLETENESS

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.

- Instruction Types
  - Functional Instructions
    - Arithmetic, logic, and shift instructions
    - ADD, CMA, INC, CIR, CIL, AND, CLA
  - Transfer Instructions
    - Data transfers between the main memory and the processor registers
    - LDA, STA
  - Control Instructions
    - Program sequencing and control
    - BUN, BSA, ISZ
  - Input/Output Instructions
    - Input and output
    - INP, OUT

Register Transfer and Operations

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 Basic Computer, a machine instruction is executed in the following cycle:**
  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**
- **After an instruction is executed, the cycle starts again at step 1, for the next instruction**
- ***Note*: Every different processor has its own (different) instruction cycle**

Register Transfer and Operations

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.

Register Transfer and Operations

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$
T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14), \quad AR \leftarrow IR(0\text{–}11), \quad I \leftarrow IR(15)$

- Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T0.
- The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1.
- At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program.
- At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR .
- Note that SC is incremented after each clock pulse to produce the sequence To, T1, and T2.

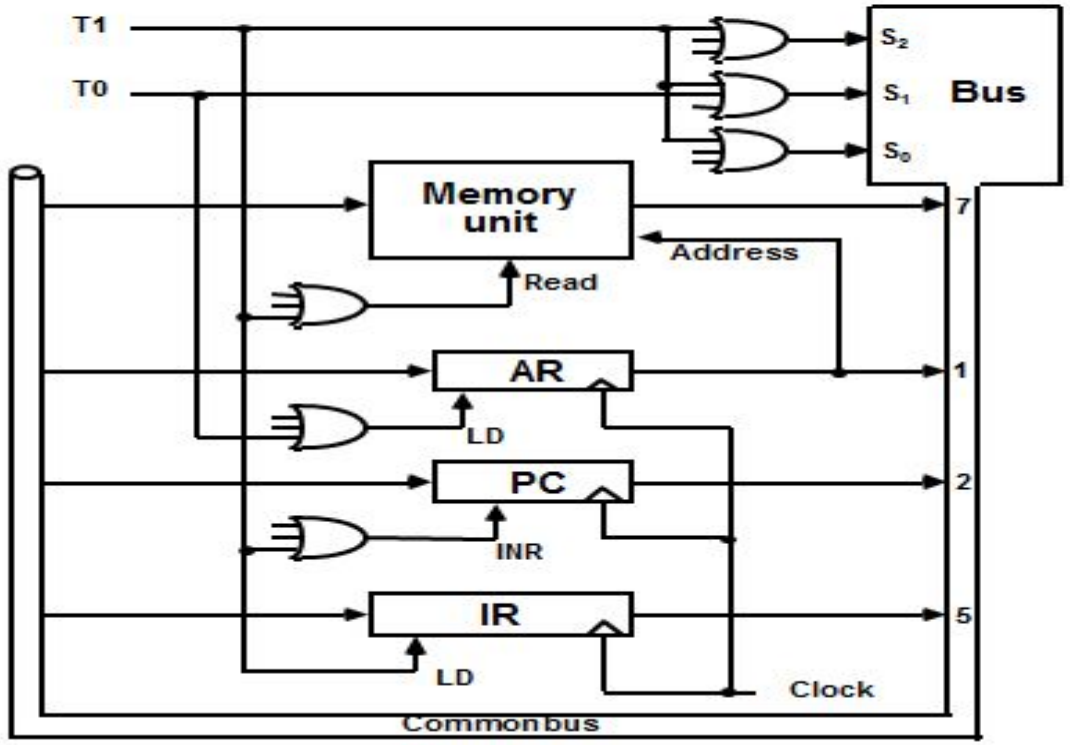
Register Transfer and Operations

How the first two register transfer statements are implemented in the bus system.

FETCH and DECODE

- **Fetch and Decode**

T0: AR ← PC ($S_0S_1S_2$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)



Register Transfer and Operations

To provide the data path for the transfer of PC to AR we must apply timing signal T0 to achieve the following connection:

1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 equal to 010.

2. Transfer the content of the bus to AR by enabling the LD input of AR .

Register Transfer and Operations

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

It is necessary to use timing signal T1 to provide the following connections in the bus system.

1. Enable the read input of memory.

2. Place the content of memory onto the bus by making S2S1S0 = 111.

3. Transfer the content of the bus to IR by enabling the LD input of IR

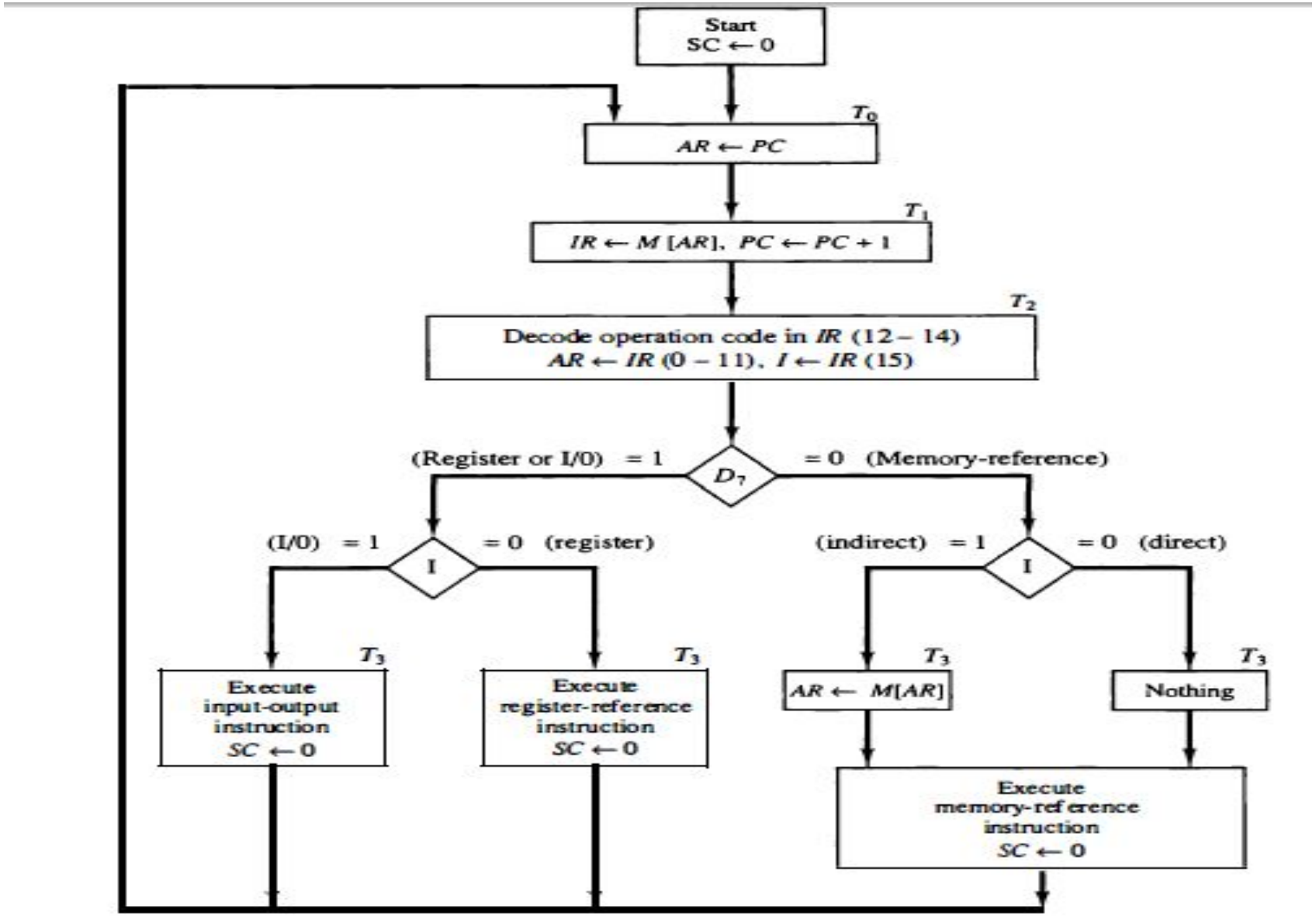
4. Increment PC by enabling the INR input of PC .

Register Transfer and Operations

Determine the Type of Instruction

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

Register Transfer and Operations



Register Transfer and Operations

- Decoder output D7, is equal to 1 if the operation code is equal to binary 111.
- WKT, if we determine that 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 have a memory reference instruction with an indirect address.
- It is then necessary to read the effective address from memory.

Register Transfer and Operations

- The micro operation for the indirect address condition can be symbolized by the register transfer statement

$$AR \leftarrow M[AR]$$

- Initially, AR holds the address part of the instruction.
- This address is used during the memory read operation.
- The word at the address given by AR is read from memory and placed on the common bus.
- The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word.

Register Transfer and Operations

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

$D'_7 I T_3$: $AR \leftarrow M[AR]$

$D'_7 I' T_3$: Nothing

$D_7 I' T_3$: Execute a register-reference instruction

$D_7 I T_3$: Execute an input–output instruction

Register Transfer and Operations

- When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR.
- However, the sequence counter SC must be incremented when
- so that the execution of the memory-reference instruction can be continued with timing variable T4.
- A register-reference or input-output instruction can be executed with the clock associated with timing signal T3.
- After the instruction is executed,
  - SC is cleared to 0 and control returns to the fetch phase with T0 = 1 .

Register Transfer and Operations

Register-Reference Instructions

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

Register Transfer and Operations

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

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

Register Transfer and Operations

- These instructions are executed with the clock transition associated with timing variable T3.
- Each control function needs the Boolean relation D7I ' T3, which we designate for convenience by the symbol r .
- The control function is distinguished by one o f the bits in IR(0-11). By assigning the symbol Bi, to bit i of IR, all control functions can be simply denoted by rBi.
- For example, the instruction CLA has the hexadecimal code 7800 which gives the binary equivalent 0111 1000 0000 0000.
- The first bit is a zero and is equivalent to I ' .
- The next three bits constitute the operation code and are recognized from decoder output D7.
- Bit 11 in IR is 1 and is recognized from B11 .

Register Transfer and Operations

- The control function that initiates the micro operation for this instruction is D7I ' T3 B11 = rB11.
- The execution of a register-reference instruction is completed at time T3.
- The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signal T0.

Register Transfer and Operations

MEMORY REFERENCE INSTRUCTIONS

- The decoded output Di 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 instruction is in the address register AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1.
- The execution of the memory-reference instructions starts with timing signal T4.
- The actual execution of the instruction in the bus system will require a sequence of micro operations. This is because data stored in memory cannot be processed directly.
- The data must be read from memory to a register where they can be operated on with logic circuits.

Register Transfer and Operations

MEMORY REFERENCE INSTRUCTIONS

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

- The effective address of the instruction is in AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1
- Memory cycle is assumed to be short enough to complete in a CPU cycle
- The execution of MR instruction starts with T_4

AND to AC

D_0T_4:	$DR \leftarrow M[AR]$	Read operand
D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$	AND with AC

ADD to AC

D_1T_4:	$DR \leftarrow M[AR]$	Read operand
D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$	Add to AC and store carry in E

Register Transfer and Operations

MEMORY REFERENCE INSTRUCTIONS

LDA: Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

STA: Store AC

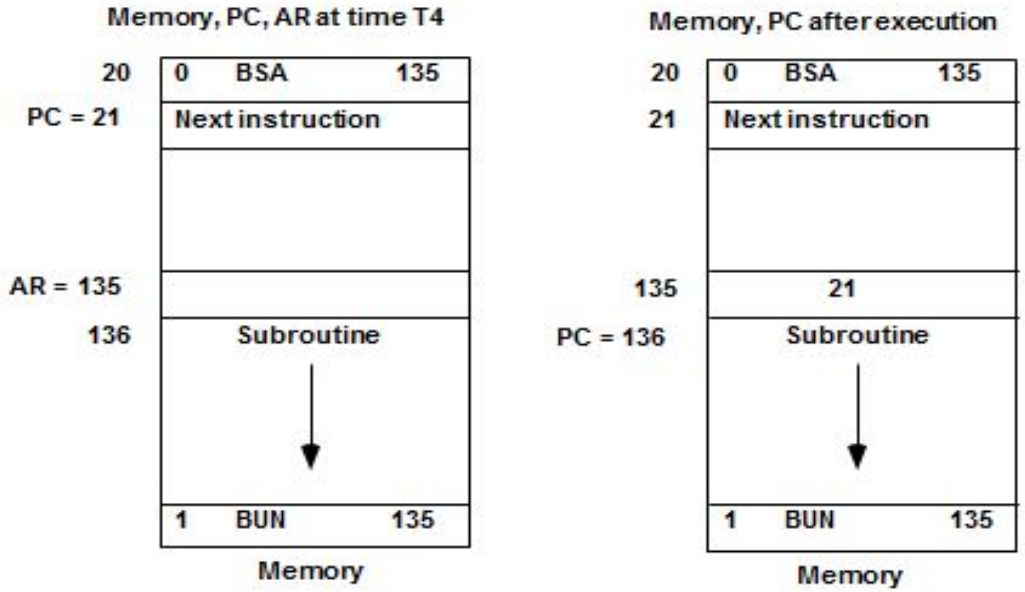
D_3T_4: M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address

M[AR] ← PC, PC ← AR + 1



Register Transfer and Operations

MEMORY REFERENCE INSTRUCTIONS

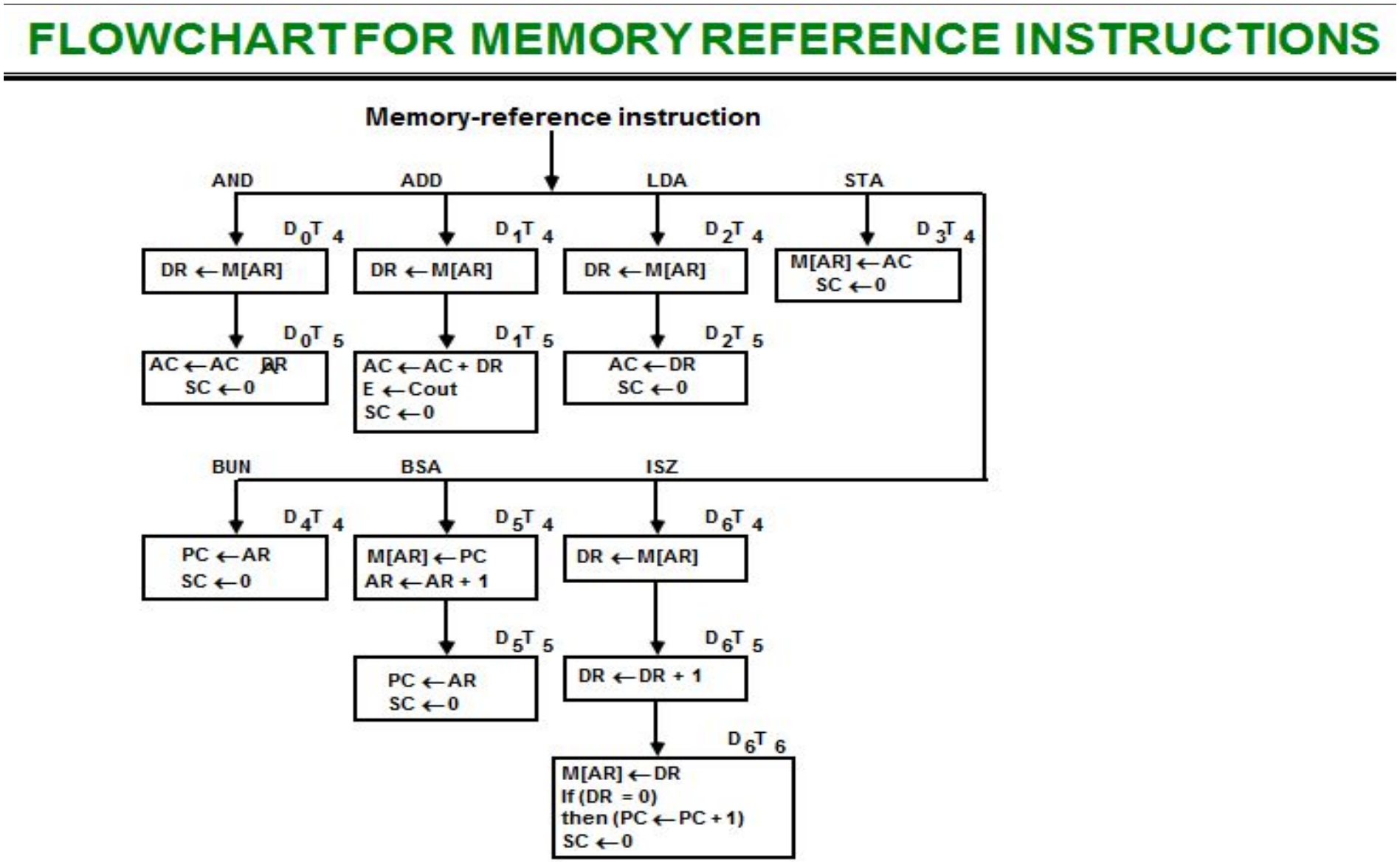
BSA:

D_5T_4: M[AR] ← PC, AR ← AR + 1
D_5T_5: PC ← AR, SC ← 0

ISZ: Increment and Skip-if-Zero

D_6T_4: DR ← M[AR]
D_6T_5: DR ← DR + 1
D_6T_4: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

Register Transfer and Operations



Register Transfer and Operations

ADDRESSING MODES

- The operation field of an instruction specifies the operation to be performed.
- This operation must be executed on some data stored in computer registers or memory words.
- The way the operands are chosen during program execution 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.

Purpose of Addressing modes in computers

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

Register Transfer and Operations

- The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instructions and execution time.
- 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.

Register Transfer and Operations

- There is one register in the computer called the program counter or 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,

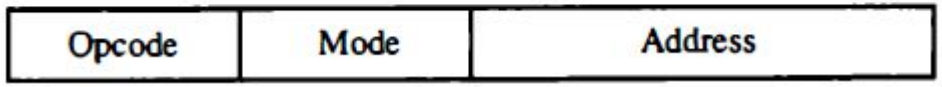
  The addressing mode of the instruction,

  And the location of the operands.
- The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence

Register Transfer and Operations

An example of an instruction format with a distinct addressing mode field is shown in Fig. 8-6.

Figure 8-6 Instruction format with mode field.



- The operation code specifies the operation to be performed.
- 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.
- 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.

Register Transfer and Operations

Implied and Immediate modes

- These two modes need no address field at all.
- 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 use an accumulator are implied-mode instructions.
- Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

Register Transfer and Operations

Immediate Mode

- 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.
- 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.

Register Transfer and Operations

Register Mode

- 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 2^K registers.

Register Transfer and Operations

Register Indirect Mode

- Register Indirect Mode: In this mode the instruction specifies a register in the CPU whose 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.
- The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

Register Transfer and Operations

Autoincrement or Autodecrement Mode

- **Autoincrement or Autodecrement Mode**: This is similar to the register indirect mode except 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.

Register Transfer and Operations

EFFECTIVE ADDRESS

- The effective address is the address of the operand in a computational type instruction.

Register Transfer and Operations

Direct Address Mode & Indirect Address Mode

- 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.
- Indirect Address Mode: In this mode the address field of the instruction gives the address 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

Register Transfer and Operations

- A few addressing modes Require that the address field of the instruction be added to the content of a specific register in the CPU.
- The effective address in these modes is obtained from the following computation:
- Effective address = address part of instruction + content of CPU register
- The CPU register used in the computation may be the

Program counter,

an index register, or

a base register.

- In either case we have a different addressing mode which is used for a different application.

Register Transfer and Operations

Relative Address Mode

- Relative Address Mode: In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- Effective address = address part of instruction + content of Program Counter

Register Transfer and Operations

- 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.
- The effective address computation for the relative address mode is 826 + 24 = 850.
- Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself.

Register Transfer and Operations

Indexed Addressing Mode

- Indexed Addressing Mode: In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- Effective address = address part of instruction + content of Index Register.
- 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 stored 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

Register Transfer and Operations

Base Register Addressing Mode

- Base Register Addressing Mode: In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- Effective address = address part of instruction + content of Base Register.
- 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.

Register Transfer and Operations

Numerical Example

- The two-word instruction at address 200 and 201 is a

  "load to AC" instruction with an address field equal to 500.
- The first word of the instruction specifies the operation code and mode, and the second word specifies the address part.
- PC has the value 200 for fetching this instruction.
- The content of processor register R 1 is 400, and

  the content of an index register XR is 100.
- AC receives the operand after the instruction is executed

Register Transfer and Operations

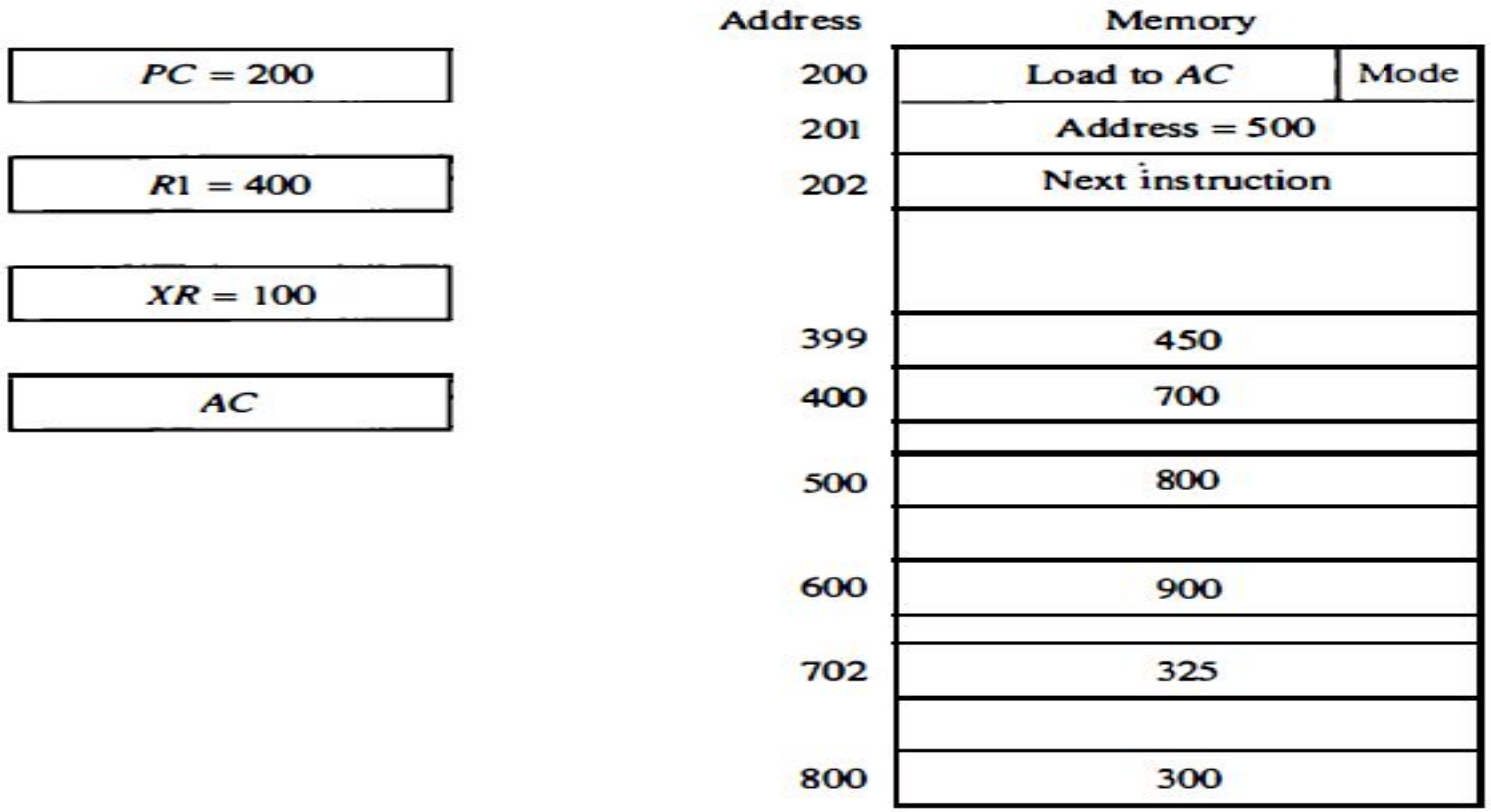


Figure 8-7 Numerical example for addressing modes.

70

Register Transfer and Operations

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

Register Transfer and Operations

DATA TRANSFER AND MANIPULATION

- Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks.
- The instruction set of different computers differ from each other mostly in the way the operands are determined from the address and mode fields.
- The actual operations available in the instruction set are not very different from one computer to another.
- It so happens that the binary code assignments in the operation code field is different in different computers, even for the same operation.
- It may also happen that the symbolic name given to instructions in the assembly language notation is different in different computers, even for the same instruction.
- Nevertheless, there is a set of basic operations that most, if not all, computers include in their instruction repertoire.

Register Transfer and Operations

Most computer instructions can be classified into Three categories

1. Data transfer instructions
2. Data manipulation instructions
3. Program control instructions

Register Transfer and Operations

- Data transfer instructions because transfer of data from one location to another without changing the binary information content.
- Data manipulation instructions are those that perform arithmetic, logic, and shift operations.
- Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer.

Register Transfer and Operations

Data Transfer Instructions

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between

I. Memory and processor registers,

II. Processor registers and input or output,

III. Processor registers themselves.

- Table 8-5 gives a list of eight data transfer instructions used in many computers

Register Transfer and Operations

TABLE 8-5 Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

76

Register Transfer and Operations

- load instruction - transfer from memory to a processor register, usually an accumulator.
- store instruction- designates a transfer from a processor register into memory.
- move instruction - transfer from

  i) one register to another

  ii) between CPU registers and memory or

  iii) between two memory words.
- exchange instruction- swaps information between two registers or a register and a memory word.
- The input and output instructions transfer data among processor registers and input or output terminals.
- The push and pop instructions- transfer data between processor registers and a memory stack.

Register Transfer and Operations

Representation of Same Instruction with different Addressing Modes

- Some assembly language conventions modify the mnemonic symbol to differentiate between the different addressing modes.
- For example, the mnemonic for load immediate becomes LDI. Other assembly language conventions use a special character to designate the addressing mode.

  For example, the immediate mode is recognized from a pound sign # placed before the operand.
- In any case, the important thing is to realize that each instruction can occur with a variety of addressing modes.

Register Transfer and Operations

load to accumulator instruction when used with eight different addressing modes.

TABLE 8-6 Eight Addressing Modes for the Load Instruction

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

79

Register Transfer and Operations

- ADR - address,
- NBR - number or operand,
- X is an index register,
- R1 - processor register, and
- AC - Accumulator register.
- @ -indirect address.
- $ character- before an address makes the address relative to the program counter PC .
- # character - precedes the operand in an immediate-mode instruction.
- An indexed mode instruction is recognized by a register that is placed in parentheses after the symbolic address.
- The register mode is symbolized by giving the name of a processor register.
- In the register indirect mode, the name of the register that holds the memory address is enclosed in parentheses.
- The autoincrement mode - plus after the parenthesized register. The autodecrement mode would use a minus instead

80

Register Transfer and Operations

Data Manipulation Instructions

- Data manipulation instructions

  Perform operations on data and

  Provide the computational capabilities for the computer.

  Divided into three basic types:

1. Arithmetic instructions
2. Logical and bit manipulation instructions
3. Shift instructions

Register Transfer and Operations

Arithmetic Instructions.

- The four basic arithmetic operations are

1. Addition,
2. Subtraction
3. Multiplication,
4. Division.

- Most computers provide instructions for all four operations. Some small computers have only addition and possibly subtraction instructions.

Register Transfer and Operations

List of typical arithmetic instructions is given in Table 8-7.

TABLE 8-7 Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Register Transfer and Operations

- An arithmetic instruction may specify fixed-point or floating-point data, binary or decimal data.
- The mnemonics for three add instructions that specify different data types are shown below.

```
ADDI    Add two binary integer numbers
ADDF    Add two floating-point numbers
ADDD    Add two decimal numbers in BCD
```


84

Register Transfer and Operations

Logical and Bit Manipulation Instructions

- Logical instructions perform binary operations on strings of bits stored in registers.
- They are useful for manipulating individual bits or a group of bits that represent binary-coded information.
- The logical instructions consider each bit of the operand separately and treat it as a Boolean variable.
- By proper application of the logical instructions it is possible to change bit values,

i) To clear a group of bits, or

ii) To insert new bit values into operands stored registers or memory words

Register Transfer and Operations

TABLE 8-8 Typical Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Register Transfer and Operations

- There are three bit manipulation operations possible: a selected bit can be

I. Cleared to 0

II. Can be set to 1

III. Can be complemented.

- The three logical instructions are usually applied to do just that

Register Transfer and Operations

Clear Selected Bits

- The AND instruction is also called a mask because it masks or inserts 0' s in a selected portion of an operand.
- The AND instruction is used to clear a bit or a selected group of bits of an operand.
- For any Boolean variable x, the relationships

  x b0 : 0 and

  x b1 : x

- dictate that a binary variable ANDed with a 0 produces a 0; but the variable does not change in value when ANDed with a 1.
- Therefore, the AND instruction can be used to clear bits of an operand selectively by ANDing the operand with another operand that has 0' s in the bit positions that must be cleared.

Register Transfer and Operations

Set Selected Bits

- The OR instruction is used to set a bit or a selected group of bits of an operand.
- For any Boolean variable x, the relationships

  $x + 1 = 1$ and

  $x + 0 = x$

- dictate that a binary variable ORed with a 1 produces a 1; but the variable does not change when ORed with a 0.
- Therefore, the OR instruction can be used to selectively set bits of an operand by ORing it with another operand with 1' s in the bit positions that must be set to 1 .

Register Transfer and Operations

Complement selected Bits

- The XOR instruction is used t o selectively complement bits of an operand.
- This is because of the Boolean relationships

$$x \oplus 1 = x'$$

and

$$x \oplus 0 = x.$$

- Thus a binary variable is complemented when XORed with a 1 but does not change in value when XORed with a 0.

Register Transfer and Operations

Shift Instructions

- Instructions to shift the content of an operand are quite useful and are often provided in several variations.
- Shifts are operations in which the bits of a word are moved to the left or right.
- The bit shifted in at the end of the word determines the type of shift used.
- Shift instructions may specify

I. Logical shifts,

II. Arithmetic shifts

III. Rotate-type operations.

- In either case the shift may be to the right or to the left

Register Transfer and Operations

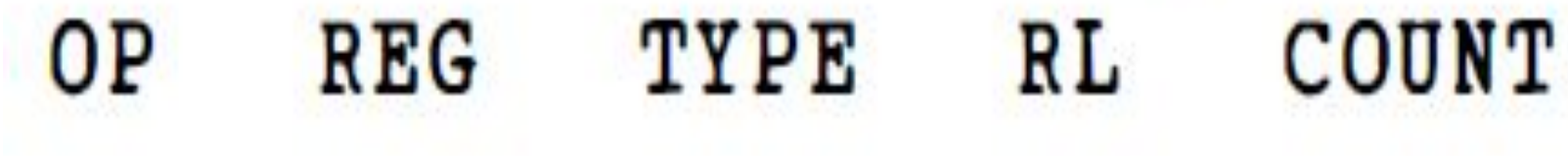
TABLE 8-9 Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Register Transfer and Operations

Multiple-field format for the shift instructions

- One field contains the operation code and the others specify the type of shift and the number of times that an operand is to be shifted.
- A possible instruction code format of a shift instruction may include five fields as follows:



Register Transfer and Operations

- OP - is the operation code field;
- REG - is a register address that specifies the location of the operand;
- TYPE is a 2-bit field specifying the four different types of shifts;
- RL is a 1-bit field specifying a shift right or left;
- COUNT is a k-bit field specifying up to $(2^K - 1)$ shifts. With such a format, it is possible to specify the type of shift, the direction, and the number of shifts, all in one instruction

Register Transfer and Operations

Reduced Instruction Set Computer(RISC)

Characteristics of RISC are as follows:

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction execution
- Hardwired rather than microprogrammed control
- A relatively large number of registers in the processor unit
- Use of overlapped register windows to speed-up procedure call and return
- Efficient instruction pipeline

Complex Instruction Set Computer(CISC)

- Characteristics of CISC are as follows:
- A larger number of instructions-typically from 100 to 250 instructions
- Some instructions that perform specialized tasks and are used infrequently
- A large variety of addressing modes- typically from 5 to 20 different modes
- Variable-length instruction formats
- Instructions that manipulate operands in memory.