**Date: 23-04-2021**

**Branch: CSE (III Year) and EE (III Year)**

**Topic:** Memory-Reference Instructions, Input-output and interrupt

and Complete computer description

**Time: 08:00 AM -09:00 AM**

**Memory-Reference Instructions**

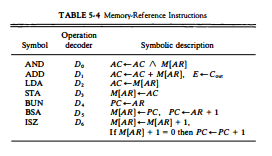
In order to specify the microoperations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely. Some instructions have an ambiguous description. This is because the explanation of an instruction in words is usually lengthy, 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 D; 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 symbolic de­scription of each instruction is specified in the table in terms of register transfer notation.

The actual execution of the instruction in the bus system will require a sequence of microoperations. 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. We now explain the operation of each instruction and list the control functions and microoperations needed for their execution.  
A flowchart that summarizes all the microoperations is pre­sented at the end of this section



AND to AC

This is an instruction that perform the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC . The micro operations that execute this instruction are:  
D0T4: DR <- M [AR]  
D0T5: AC <- AC /\ DR, SC <--- 0  
  
The control function for this instruction uses the operation decoder D0 since this output of the decoder is active when the instruction has an AND operation whose binary code value 000. Two timing signals are needed to execute the instaruction. The clock transition associared with  timing signal T4 transfers the operand from memory into DR . The clock transition associated with the next timing signal T5 transfers to AC the result of the AND logic operation between the contents of DR and AC. The same clock transition clears SC to 0, transfer­ring control to timing signal T0 to start a new instruction cycle.

ADD to AC

 This instruction adds the content of the memory word specified by the effective address to the value of AC . The sum is transferred into AC and the output carry Cout is transferred to the E (extended accumulator) flip-flop. The rnicroopera­tions needed to execute this instruction are  
D1T4: DR← M[AR]  
D1T5: AC← AC + DR, E← Cout , SC ← 0

Same Two timing signals, T, and T5, are used again but with operation decoder D1 instead of D0, which was used for the AND instruction. After the instruction is fetched from memory and decoded, only one output of the operation decoder will be active, and that output determines the sequence of microoperations that the control follows during the execution of a memory-ref­erence instruction.

LDA: Load to AC

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

 D2T4: DR← M [AR]  
D2T5: AC← DR ,  ← 0  
  
Looking back at the bus system shown in Fig. 5-4 we note that there is no direct path from the bus into AC . The adder and logic circuit receive information from DR which can be transferred into AC . Therefore, it is necessary to read the memory word into DR first and then transfer the content of DR into AC . The reason for not connecting the bus to the inputs of AC is the delay encountered in the adder and logic circuit. It is assumed that the time it takes to read from memory and transfer the word through the bus as well as the adder and logic circuit is more than the time of one clock cycle. By not connecting the bus to the inputs of AC we can maintain one clock cycle per microoperation.

**STA: Store AC**

This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation:  
D3T4: M [AR] ← AC, SC ← 0

BUN: Branch Unconditionally

This instruction transfers the program to the instruction specified by the effective address. Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle. PC is incremented at time T1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:  
D4T4: PC ← AR, SC ← 0

The effective address frpom AR is transferred through the common bus to PC .Resetting SC to 0 transfers control to T0• The next instruction is then fetched and executed from the memory address given by the new value in PC .

**BSA: Branch and Save Return Address**

This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine. This operation was specified in Table 5-4 with the following register transfer:

M[AR] <-- PC, PC <-- AR + I

A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig. 5-10. The BSA instruction is assumed to be in memory at address 20. The I bit is 0 and the address part of the instruction has the binary equivalent of 135. After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address). AR holds the effective address 135. This is shown in part (a) of the figure. The BSA instruction performs the following numerical operation:

M[135] <-- 21, PC <-- 135 + 1 = 136

The result of this operation is shown in part (b) of the figure. The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136. The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine. When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21. When the BUN instruction is executed, the effective address 21 is transferred to PC . The next instruction cycle finds PC with the value 21, so control continues to execute the instruction at the return address.

ISZ: Increment and Skip if Zero

 This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by one in order to skip the next instruction in the program.  
Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory. This is done with the following sequence of microoperations:  
D6T4: DR <-- M [AR]  
D6T5: DR <-- DR + 1  
D,T,: M [AR] <-- DR,  
if (DR = 0) then (PC ← PC + 1), SC ← 0

**Input-output and interrupt**

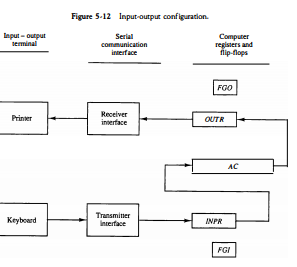
A computer can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device. Commercial computers include many types of input and output devices. To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer. Input-output organization is dicsussed further in Chap. 11.

Input-Output Configuration

The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code. The serial information from the keyboard is shifted into the input register INPR. The serial information for the printer is stored in the output register OUTR. These two registers communicate with a communication interface serially and with the AC in parallel. The input-output configuration is shown in Fig. 5-12. The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially. The operation of the serial communication interface is explained in Sec. 11-3.

input register:

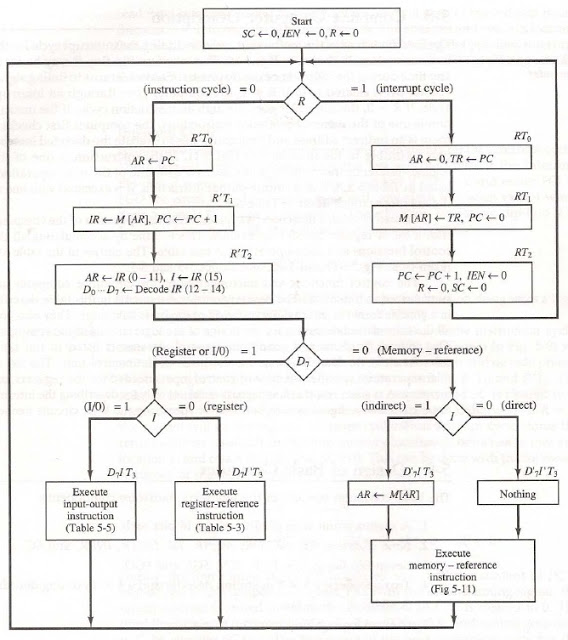
The input register INPR consists of eight bits and holds alphanumeric input information. The 1-bit input flag FGI is a control flip-flop. The flag bit is



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

**Complete Computer Description**

**Flowchart**

This is the final flowchart of the instruction cycle including interrupt cycle for the basic computer.  
[](https://4.bp.blogspot.com/-1AYFcCTkStc/VnGVHDlUAZI/AAAAAAAAA5Y/AqrxBhsqd8Y/s1600/flowchart%2Bfor%2Bcomplete%2Bcomputer%2Bdescription.jpg)