**Date: 22-04-2021**

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

**Topic:** Timing and Control and Instruction cycle

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

The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit. The clock pulses do not change the state of a register unless the register is enabled by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.

There are two major types of control organization:

1. hardwired control and
2. microprogrammed control.

In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation.

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

A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed.

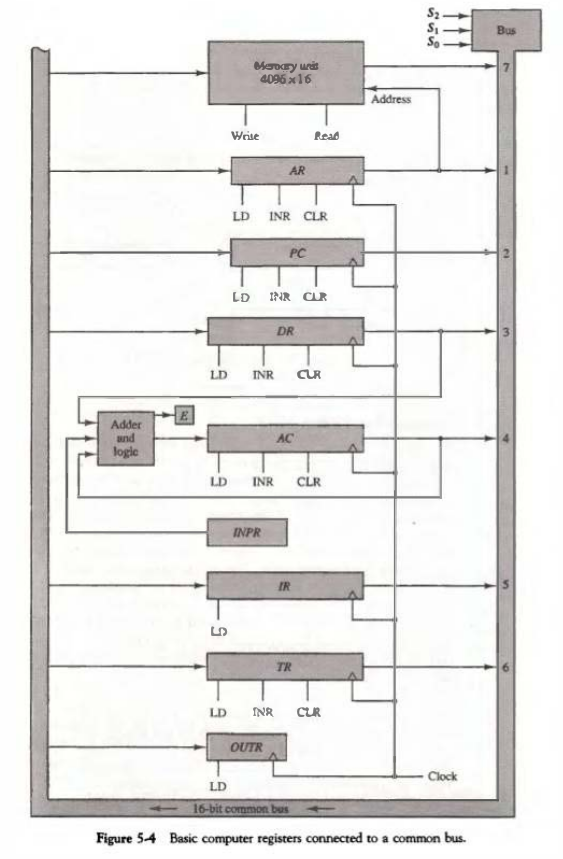
In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

The block diagram of the control unit is shown in Fig. 5.6.

It consists of two decoders,

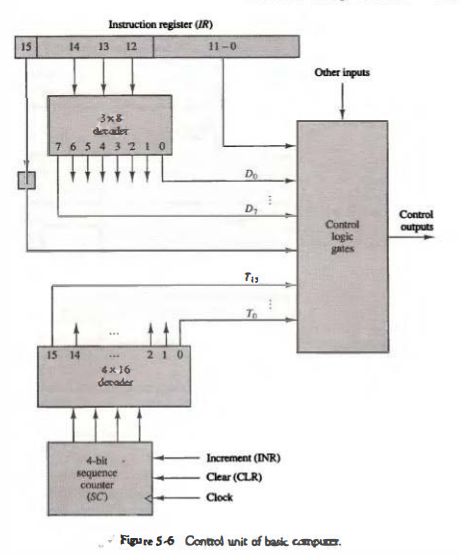
1. a sequence counter, and
2. a number of control logic gates.

An instruction read from memory is placed in the instruction register (IR).position of this register in the common bus system is indicated in Fig 5.4.



The instruction register is shown again in Fig. 5.6, where it is divided into three parts:

1. the 1 bit,
2. the operation code, and
3. bits 0 through 11.



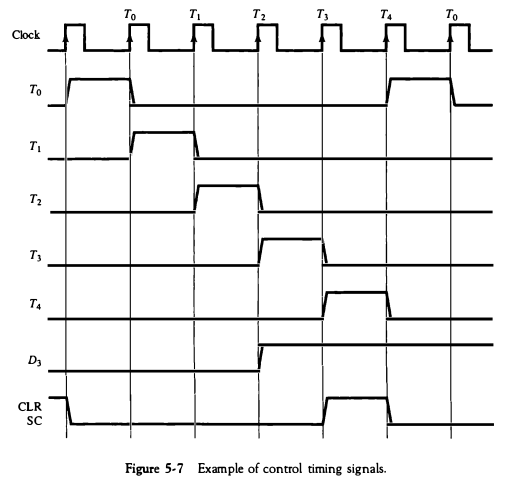
The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15.

The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be T0.

As an example, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3, and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement

D3T4: SC <\_\_ 0

The timing diagram of Fig. 5-7 shows the time relationship of the control signals.



The sequence counter SC responds to the positive transition of the clock. Initially, the CLR input of SC is active. The first positive transition of the clock clears SC to 0, which in tum activates the timing signal T0 out of the decoder. T0 is active during one clock cycle. The positive clock transition labeled T0 in the diagram will trigger only those registers whose control inputs are transition, to timing signal T0. SC is incremented with every positive clock transition unless its CLR input is active. This produces the sequence of timing signals T0, T1, T2, T3 ,T4 and so on, as shown in the dagram. (Note the the relationshuip between the timing signal and and its corresponding positive clock transition.) If SC is not cleared, the timing signals will continue with T5,  T6 up to T15 and back to T0

The last three waveforms in Fig. 5-7 show how SC is cleared when D3T4 = 1. Output D3 from the operation decoder becomes active at the end of timing signal T2. When timing signal T4 becomes active, the output of the AND gate that implements the control function D3T4 becomes active. This signal is applied to the CLR input of SC. On the next positive clock transition (the one marked T4 in the diagram) the counter is cleared to 0. This causes the timing signal T0 to become active instead of T5 that would have been active if SC were incremented instead of cleared.

A memory read or write cycle will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, a memory read or write cycle ini­ tiated by a timing signal will be completed by the time the next clock goes through its positive transition. The clock transition will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the memory cycle time is usually longer than the processor clock cycle. In such a case it is necessary to provide wait cycles in the processor until the memory word is available. To facilitate the presentation, we will assume that a wait period is not necessary in the basic computer.

To fully comprehend the operation of the computer, it is crucial that one understands the timing relationship between the clock transition and the timing signals. For example, the register transfer statement

T0: AR <\_\_ PC

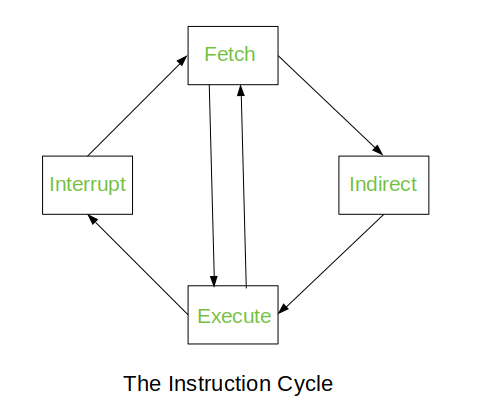
specifies a transfer of the content of PC into AR if timing signal T0 is active. T0 is active during an entire clock cycle intervaL During this time the content of PC is placed onto the bus (with S2S1S0 = 010) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition. This same positive clock transition increments the sequence counter SC from 0000 to 0001 . The next clock cycle has T1 active and T0 inactive.

Computer Organization | Different Instruction Cycles

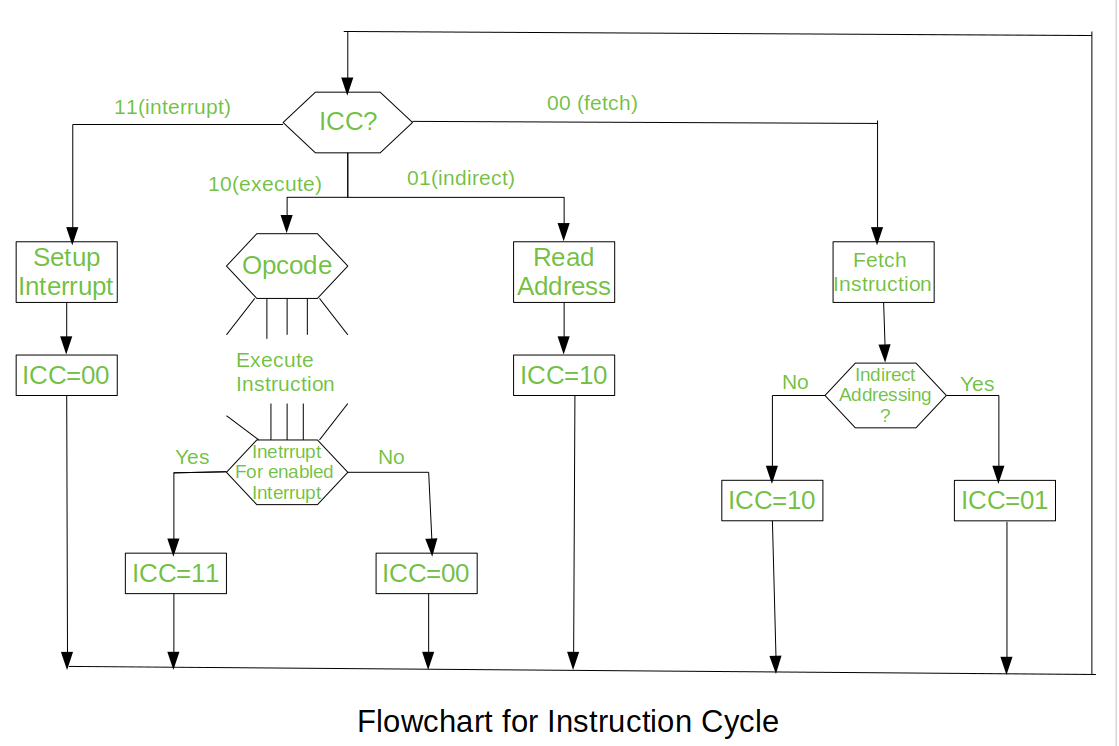
Registers Involved In Each Instruction Cycle:

* **Memory address registers (MAR)** : It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
* **Memory Buffer Register (MBR)** : It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
* **Program Counter(PC)**: Holds the address of the next instruction to be fetched.
* **Instruction Register(IR)** : Holds the last instruction fetched.

**The Instruction Cycle –**

Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations. In the above examples, there is one sequence each for the Fetch, Indirect, Execute and Interrupt Cycles.  


The Indirect Cycle is always followed by the Execute Cycle. The Interrupt Cycle is always followed by the Fetch Cycle. For both fetch and execute cycles, the next cycle depends on the state of the system.



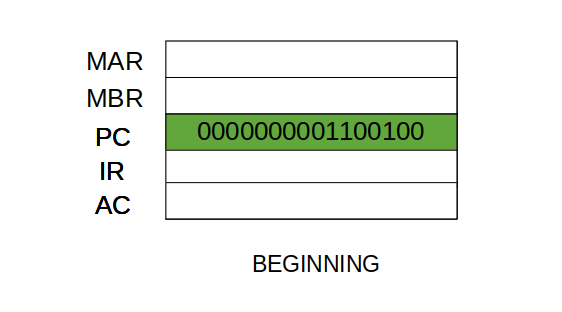
We assumed a new 2-bit register called Instruction Cycle Code (ICC). The ICC designates the state of processor in terms of which portion of the cycle it is in:-

00 : Fetch Cycle  
01 : Indirect Cycle  
10 : Execute Cycle  
11 : Interrupt Cycle

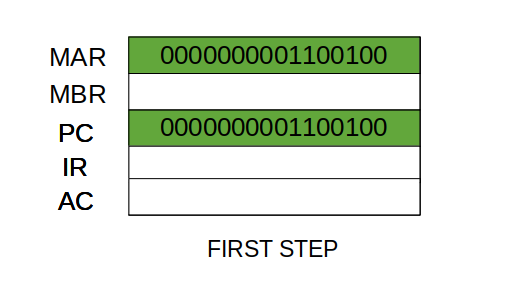
At the end of the each cycles, the ICC is set appropriately. The above flowchart of Instruction Cycle describes the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern(this is a simplified example). The operation of the processor is described as the performance of a sequence of micro-operation.

Different Instruction Cycles:

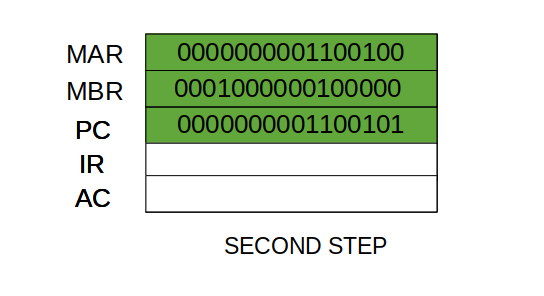
1. **The Fetch Cycle –**  
   At the beginning of the fetch cycle, the address of the next instruction to be executed is in the Program Counter(PC).



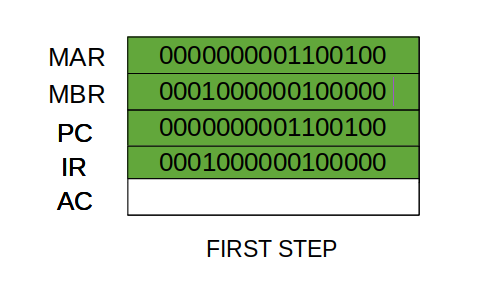
Step 1: The address in the program counter is moved to the memory address register (MAR), as this is the only register which is connected to address lines of the system bus.



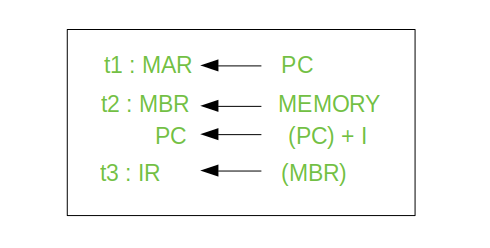
Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction.(These two action can be performed simultaneously to save time)



Step 3: The content of the MBR is moved to the instruction register(IR).



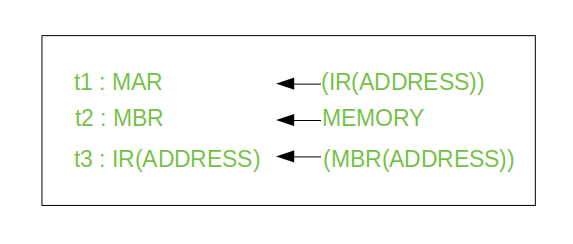
Thus, a simple Fetch Cycle consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:-



Here ‘I’ is the instruction length. The notations (t1, t2, t3) represents successive time units. We assume that a clock is available for timing purposes and it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit.  
First time unit: Move the contents of the PC to MAR.  
Second time unit: Move contents of memory location specified by MAR to MBR. Increment content of PC by I.  
Third time unit: Move contents of MBR to IR.  
**Note:** Second and third micro-operations both take place during the second time unit.

1. **The Indirect Cycles –**

Once an instruction is fetched, the next step is to fetch source operands. Source Operand is being fetched by indirect addressing( it can be fetched by any [addressing mode](https://www.geeksforgeeks.org/addressing-modes/), here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following micro-operations takes place:-

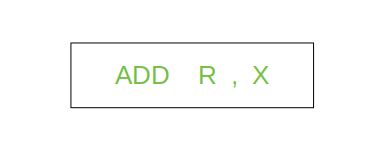


Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.  
Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)  
Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

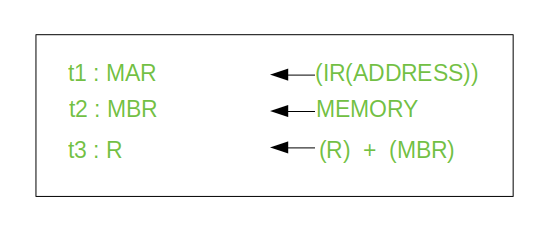
**Note:** Now IR is ready for the execute cycle, but it skips that cycle for a moment to consider the Interrupt Cycle .

1. **The Execute Cycle**

The other three cycles (Fetch, Indirect and Interrupt) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.  
Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.  
Lets take an hypothetical example :-  
consider an add instruction:



Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

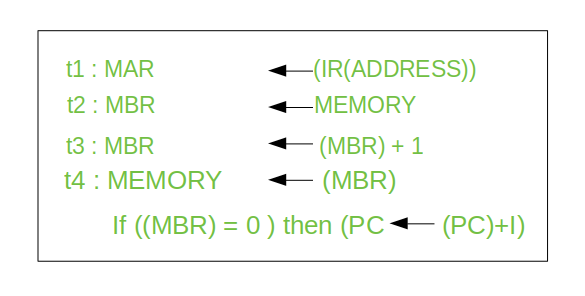


We begin with the IR containing the ADD instruction.  
Step 1: The address portion of IR is loaded into the MAR.  
Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.  
Step 3: Now, the contents of R and MBR are added by the ALU.

Lets take a complex example:-

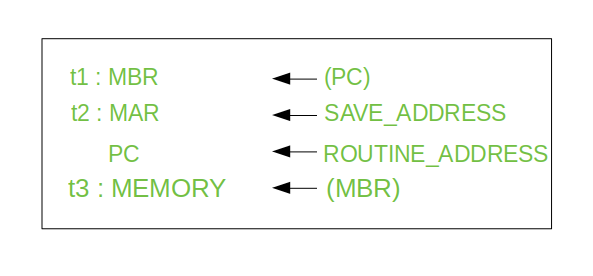


Here, the content of location X is incremented by 1. If the result is 0, the next instruction will be skipped. Corresponding sequence of micro-operation will be :-



Here, the PC is incremented if (MBR) = 0. This test (is MBR equal to zero or not) and action (PC is incremented by 1) can be implemented as one micro-operation.  
**Note** : This test and action micro-operation can be performed during the same time unit during which the updated value MBR is stored back to memory.

1. **The Interrupt Cycle**:  
   At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The natare of this cycle varies greatly from one machine to another.  
   Lets take a sequence of micro-operation:-



Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.  
Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.  
PC is loaded with the address of the start of the interrupt-processing routine.  
Step 3: MBR, containing the old value of PC, is stored in memory.

Note: In step 2, two actions are implemented as one micro-operation. However, most processor provide multiple types of interrupts, it may take one or more micro-operation to obtain the save\_address and the routine\_address before they are transferred to the MAR and PC respectively.