UNIT-II

**BASIC COMPUTER ORGANIZATION**

**CONTENTS:**

1. Instruction codes,
2. Computer Registers-Types,
3. Computer Instructions – Instruction set completeness,
4. Timing and Control,
5. Instruction Cycle- Fetch and Decode,
6. Register Reference Instructions,
7. Memory-Reference Instructions,
8. Input- Output and Interrupt – Input-Output Configuration,
9. Input- Output Instructions,
10. Program Interrupt,
11. Interrupt Cycle

**1.INSTRUCTION CODES:**

**Instruction Codes**

Modern processor is a very complex device. It contains:

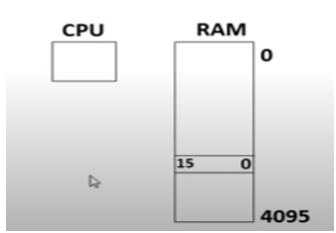
- Many registers

-Multiple arithmetic units, for both integer and floating point calculations

- The ability to pipeline several consecutive instructions to speed execution Etc.

Every different processor type has its own design

M. Morris Mano introduced a simple processor model ,called**Basic Computer.**

* The Basic computer has two components, a processor and memory
* The memory has 4096 words in it
* 4096=212  so it takes 12 bits to select a word in memory
* Each word is 16 bits long
* 

Program: A set of instructions that specifies operation, operands, and sequence of processing

has to occur.

The instructions of a program, along with any needed data are stored in memory.

The CPU reads the next instruction from memory and places it in an Instruction

Register(IR).

Control circuitry in control unit then translates the instruction into the sequence of micro-

operations necessary to implement it .

Computer Instruction: A binary code that specifies a sequence of micro-operations for the computer.

Every computer has its own unique instruction set.

**Instruction code**: A group of bits instructing the computer to perform a specific operation.

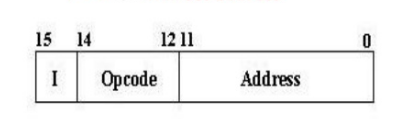
It is divided into parts, each with particular meaning.

**Instruction Format**

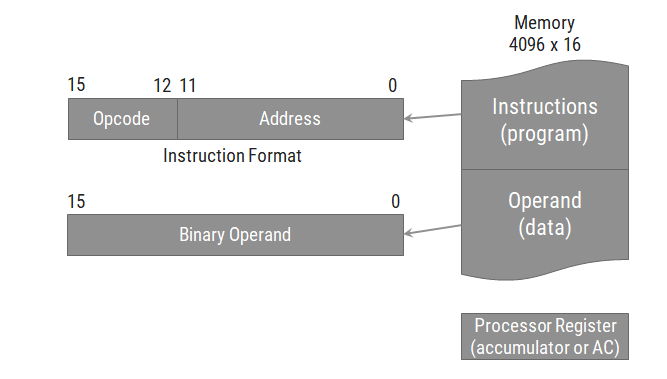
**Operation code**: This is the most important part of instruction code .

It defines the type operation like add, subtract, multiply, shift, and complement.

* In the basic computer, since the memory contains 4096=(212) words, we needs 12 bits to specify which memory address this instruction will use
* In the basic computer bit 15 of the instruction specifies the addressing mode(0:direct addressing,1:Indirect addressing)
* Since the memory words and hence the instructions, are 16 bits long ,that leaves 3 bits for instructions opcode



**Stored Program Organization**

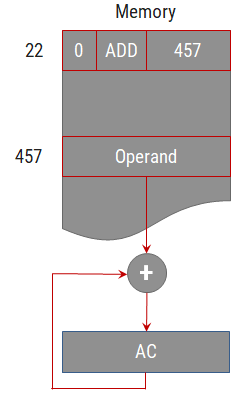


* The simplest way to organize a computer is to have one processor register(AC) and an instruction code format with two parts.
  + The first part specifies the operation (opcode) to be performed and the second specifies an address (operand).
* The memory address tells the control where to find an operand 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.
* Instructions are stored in one section of memory and data in another.
* For a memory unit with 4096 words, we need 12 bits to specify an address since 212 = 4096.
* If we store each instruction code in one 16-bit memory word, we have available four bits for operation code (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 then executes the operation specified by the operation code.

**Direct & Indirect Addressing of Memory**

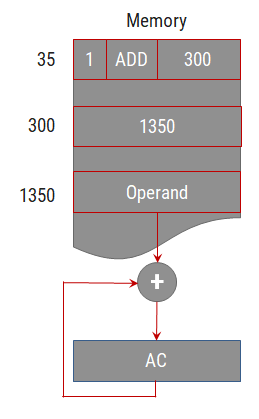
**Direct Address**

* If the second part of an instruction format specifies the address of an operand, the instruction is said to have a **direct address**.

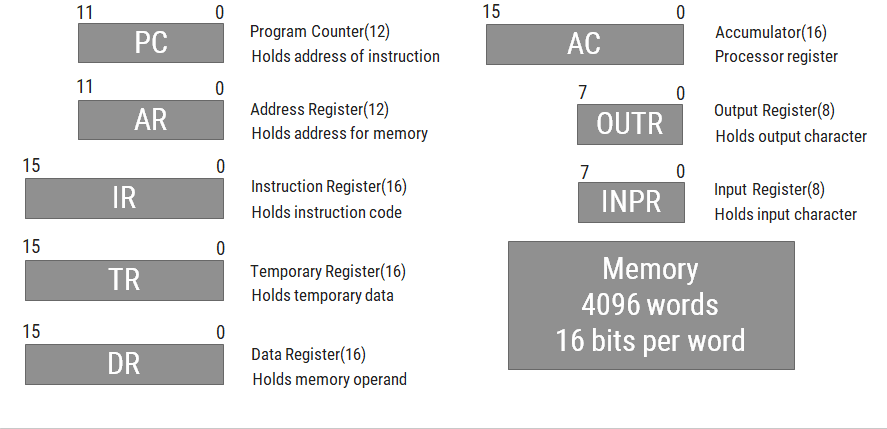


**Indirect Address**

* In Indirect address, 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**.**

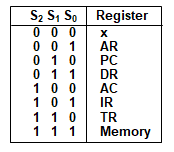
****

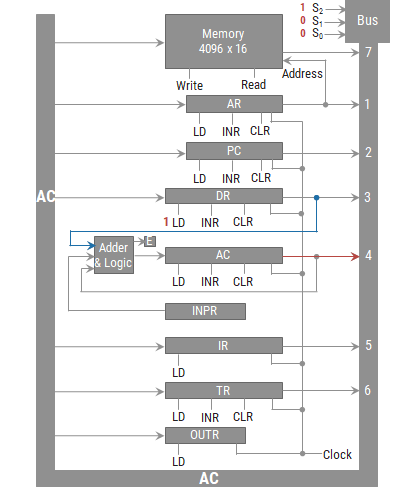
**2.Computer Registers**

****

**Common Bus System of Basic Computer**

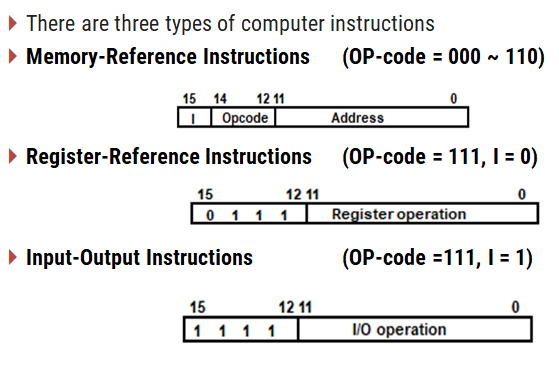
Three control lines, S2, S1, and S0 control which register the bus selects as its input

****

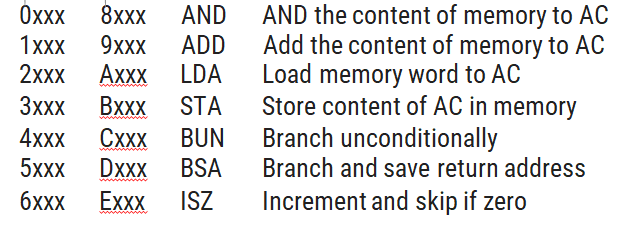
****

* Either one of the registers will have its load signal activated, or the memory will have its read signal activated
  + Will determine where the data from the bus gets loaded
* The 12-bit registers, AR and PC, have 0’s loaded onto the bus in the high order 4 bit positions
* When the 8-bit register OUTR is loaded from the bus, the data comes from the low order 8 bits on the bus

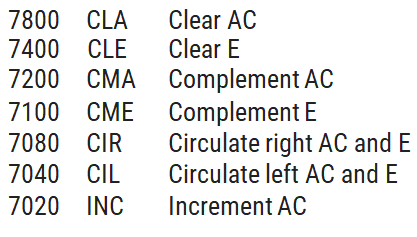
**3.Computer Instructions**

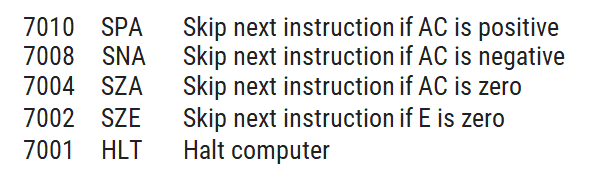
****

**Set of Memory Reference Instructions**

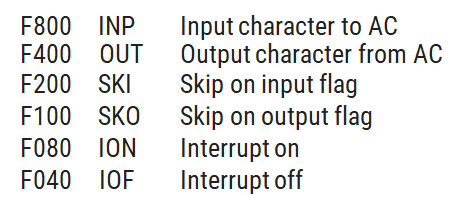
****

**Set of Register Reference Instructions**

****

****

**Set of Input-Output Instruction**

****

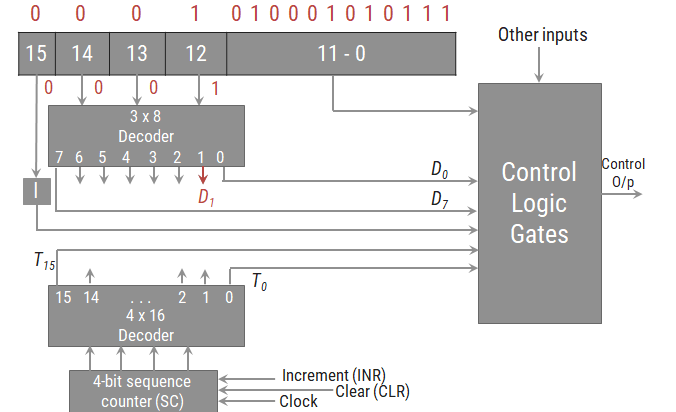
**Instruction Set Completeness**

* Instruction set is said to be complete if it includes enough instructions in each of the following categories:
  1. Arithmetic, logical and shift instructions
  2. Instructions for moving information to and from memory and processor registers
  3. Program control instructions together with instructions that check status conditions
  4. Input and output instructions

4.**Timing and Control**

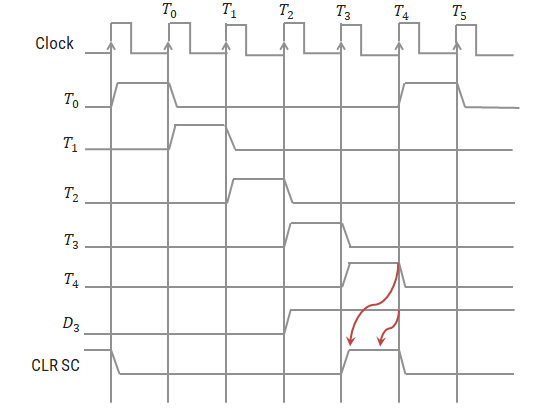
* Control unit (CU) of a processor translates from machine instructions to the control signals (for the microoperations) that implement them
* Control units are implemented in one of two ways
* *Hardwired* Control
  + CU is made up of sequential and combinational circuits to generate the control signals
* *Microprogrammed* Control
  + A control memory on the processor contains microprograms that activate the necessary control signals
  + We will consider a hardwired implementation of the control unit for the Basic Computer

Control unit of basic computer



* Components of Control unit are
  1. Two decoders
  2. A sequence counter
  3. Control logic gates
* An instruction read from memory is placed in the instruction register (IR).
* In control unit the IR is divided into three parts: I bit, the operation code (12-14)bit, and bits 0 through 11.
* The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder.
* Bit-15 of the instruction is transferred to a flip-flop designated by the symbol I.
* The eight outputs of the decoder are designated by the symbols D0 through D7.
* 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 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 4 X 16 decoder.
* Once in awhile, the counter is cleared to 0, causing the next 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
* Initially, the CLR input of SC is active.
* The first positive transition of the clock clears SC to 0, which in turn activates the timing 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 connected to timing signal T0.

**Timing Cycle for D3T4: SC ← 0**



* SC is incremented with every positive clock transition, unless its CLR input is active.
* This procedures the sequence of timing signals T0, T1, T2, T3 and T4, and so on. If SC is not cleared, the timing signals will continue with T5, T6, up to T15 and back to T0.
* The last three waveforms shows 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 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.

5.**Instruction Cycle**

* A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of the following phases:
  1. Fetch an instruction from memory.
  2. Decode the instruction.
  3. Read the effective address from memory if the instruction has an indirect address.
  4. Execute the instruction.
* After step 4, the control goes back to step 1 to fetch, decode and execute the next instruction.
* This process continues unless a HALT instruction is encountered.

**Fetch & Decode**

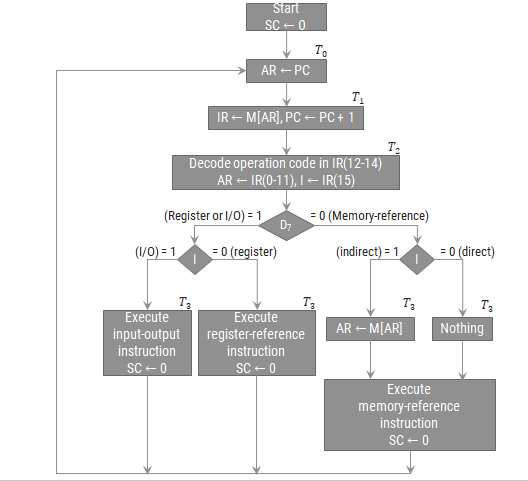
PC is loaded with the address of the first instruction in the program.

The micro-operations for fetch and decode phases are as follows:

Determine the type of instruction

During time , the control unit determines the type of instruction i.e. Memory reference, Register reference or Input-Output instruction.

If then instruction must be register reference or input-output else memory reference instruction.

****

**6.Memory-Reference Instructions**

1. **AND: AND to AC**

This is an instruction that performs 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.

**D0T4: DR ←M[AR]**

**D0T5: AC ←AC∧ DR, SC ← 0**

1. **ADD: 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.

**D1T4: DR ←M[AR]**

**D1T5: AC ←AC + DR, E ←Cout, SC ← 0**

1. **LDA: Load to AC**

This instruction transfers the memory word specified by the effective address to AC.

**D2T4: DR ←M[AR]**

**D2T5: AC ← DR, SC ← 0**

1. **STA: Store AC**

This instruction stores the content of AC into the memory word specified by the effective address.

**D3T4: M[AR] ← AC, SC ← 0**

1. **BUN: Branch Unconditionally**

This instruction transfers the program to instruction specified by the effective address. The BUN instruction allows the programmer to specify an instruction out of sequence and the program branches (or jumps) unconditionally.

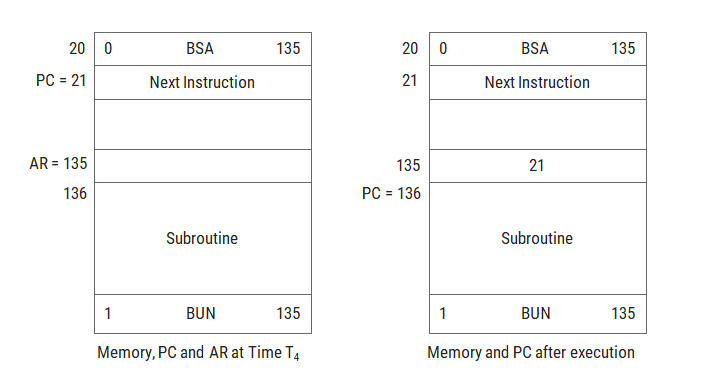
**D4T4: PC ← AR, SC ← 0**

1. **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

**D5T4: M[AR] ← PC, AR ← AR + 1**

**D5T5:PC← AR, SC ← 0**

****

1. **ISZ: Increment and Skip if Zero**

These instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. 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

**D6T4: DR ←M[AR]**

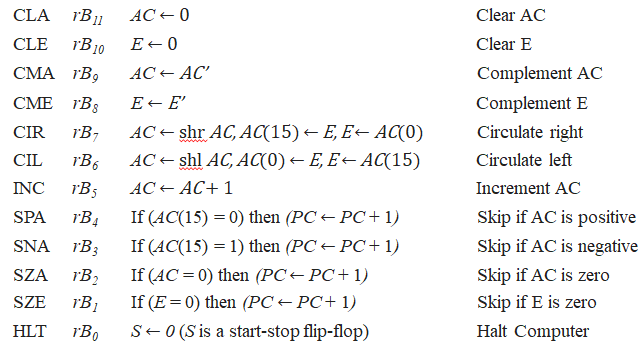
**D6T5: DR ←DR + 1**

**D6T6: M[AR] ← DR, if (DR = 0) then (PC ←PC + 1), SC ← 0**

**Register Reference Instruction**

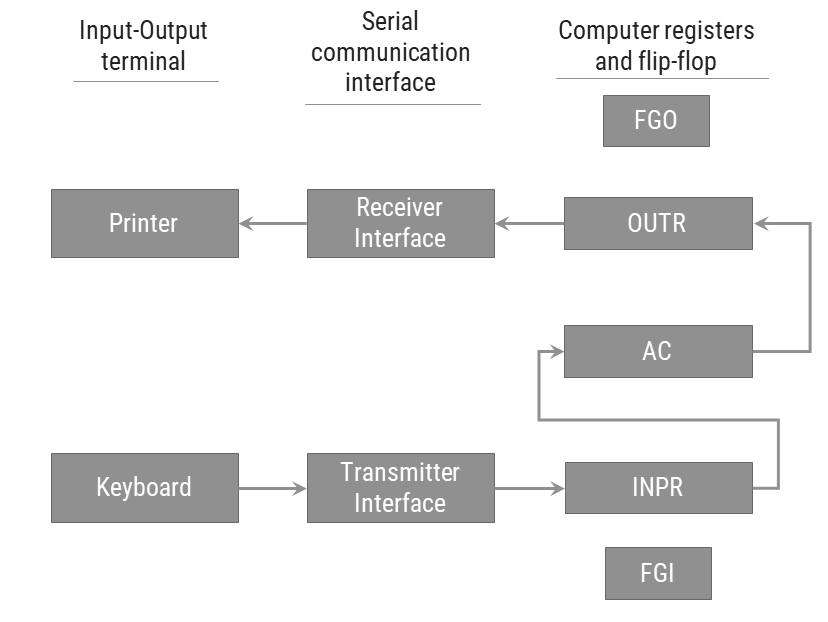
*D7I’T3 = r* (common to all register reference instructions)

*IR(i) = Bi* [bit in *IR*(0-11) that specifies the operation]



**Input-output and Interrupt**

**Input-Output of basic computer**

****

**Process of input & output information transfer**

**Input Transfer**

* 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.

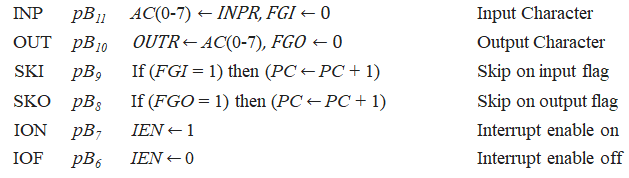
**Output Transfer**

* The output register OUTR works similarly but the direction of information flow is reversed.
* Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0.
* The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.
* The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

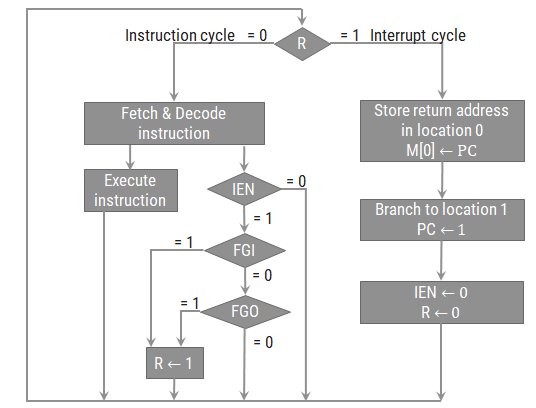
**Input-Output Instruction**

*D7IT3 = p* (common to all input-output instructions)

*IR(i) = Bi* [bit in *IR*(6-11) that specifies the operation]



**Interrupt Cycle**



* The interrupt cycle is a hardware implementation of a branch and save return address operation.
* An interrupt flip-flop R is included in the computer.
* When R = 0, the computer goes through an instruction cycle.
* During the execute phase of the instruction cycle IEN is checked by the control.
* If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
* If IEN is 1, control checks the flag bits.
* If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information.
* In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1.
* At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

**Register transfer statements for Interrupt cycle**

* The flip-flop is set to 1 if IEN = 1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T0, T1 or T2 are active.
* The condition for setting flip-flop R = 1 can be expressed with the following register transfer statement:

T0′ T1′ T2′(IEN) (FGI + FGO): R ← 1

* The symbol + between FGI and FGO in the control function designates a logic OR operation. This is AND with IEN and T0′ T1′ T2′.
* The fetch and decode phases of the instruction cycle must be modified and Replace T0, T1, T2 with R'T0, R'T1, R'T2
* Therefore the interrupt cycle statements are :

RT0 : AR ← 0, TR ← PC

RT1 : M[AR] ← TR, PC ← 0

RT2 : PC ←PC + 1, IEN ← 0, R ← 0, SC ← 0

* During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR.
* With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0.
* The third timing signal increments PC to 1, clears IEN and R, and control goes back to T0 by clearing SC to 0.
* The beginning of the next instruction cycle has the condition RT0 and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

**Demonstration of Interrupt Cycle**

