**Basic Computer Organization and Architecture**

Introduction

We know that a computer is a machine that performs some tasks on a given set of data. The computer needs a processor to perform the mathematical and logical operations of the job. Every processor type has a unique design. We will discuss a simplified processor model called Basic Computer to understand how processors work and the design of a basic computer further in this article.

## Hardware components of basic computer

The basic computer comprises the following hardware components:

1. Memory Unit with 4096 words of 16 bits each
2. Eight registers
3. Seven flip-flops
4. Two decoders
5. A 16-bit common bus
6. Control logic gates
7. Adder and logic circuit connected to the input of AC

## Memory / Ram

## Memory or Ram is the array of registers that store instructions and data which are used at the time of program execution. In our computer architecture, a simple 4096 X 16 Ram chip is used, which has 4096 addresses to store 16-bit data. For better execution of the program, we store instructions in a consecutive manner. Memory is an important part of our computer system, as all the instructions are stored in memory and fetched one by one to execute the program

## 

Computer Registers

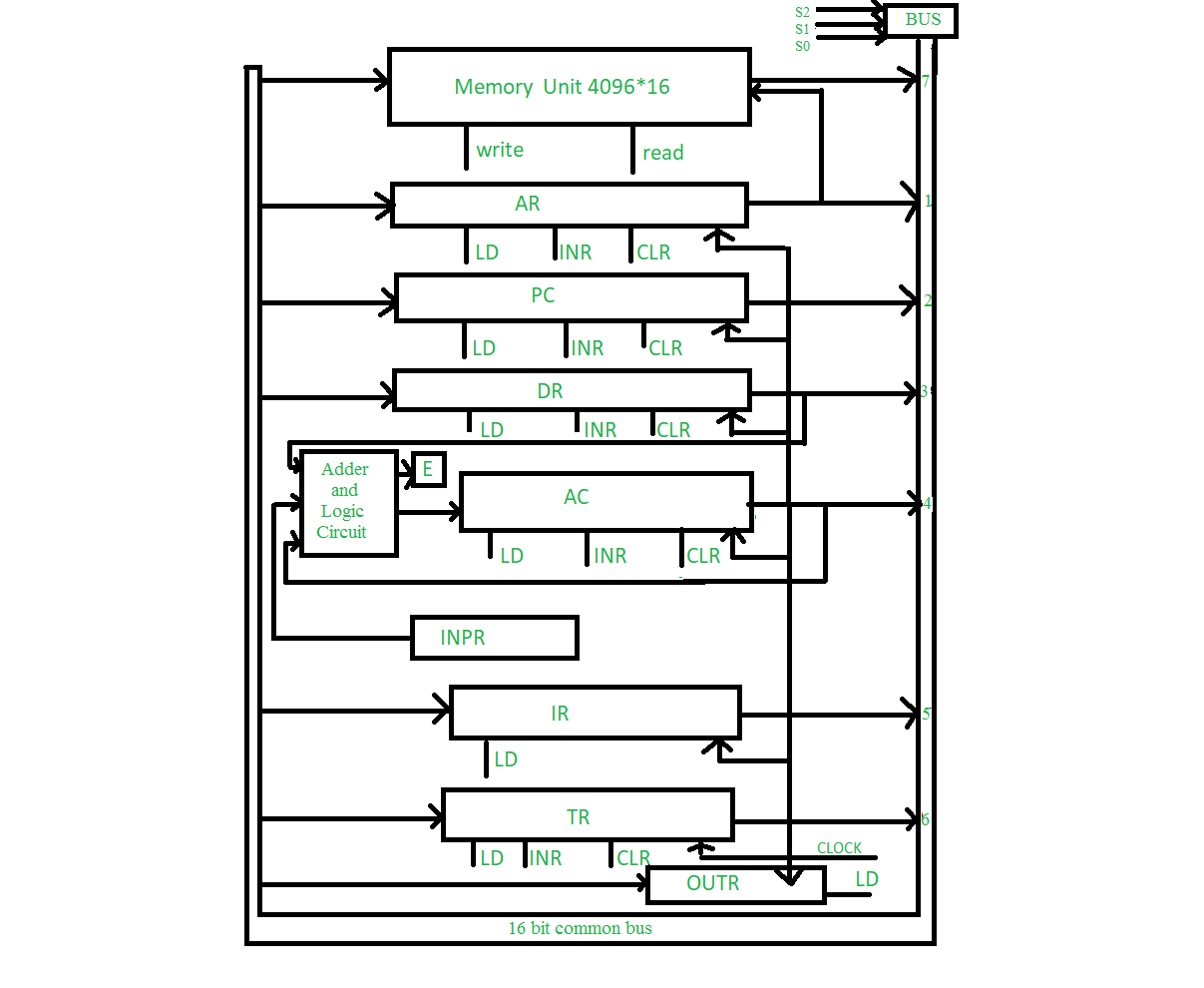
Registers are temporary storage areas for instructions or data. They are not a part of memory; rather they are special additional storage locations that offer the advantage of speed. Registers work under the direction of the control unit to accept, hold, and transfer instructions or data and perform arithmetic or logical comparisons at high speed. Our computer system uses 8 total registers that work particularly according to their specific work.

Those 8 registers are.

* Address Register / AR : (12 bits) It is the address storage register that holds the address for memory.
* Program Counter / PC : (12 bits) It is the address storage register that holds the address of the instruction.
* Data Register / DR : (16 bits) It is the data storage register that holds the data coming from memory reading.
* Accumulator / AC : (16 bits) It is the data storage register that holds the data coming from the ALU operation.
* Instruction Register / IR : (16 bits) It is the data storage register that holds the instruction code of a program.
* Temporary Register / TR : (16 bits) It is the data storage register that holds the temporary data.
* INPR : (8 bits) It is the input register that holds the input characters.
* OUTR : (8 bits) It is the output register that holds the output characters

Common Bus System

Common bus system is the most effective way to transfer data from one register to another. The concept of a common bus system works as the register or ram transfers its data into the common bus and the register or ram which wants data loads that data from the common bus. Since we transfer both data and address values through a common bus system, the size of the common bus system should be 16 bits so that data values are also stored in the common bus system. To control the common bus system from where the data of the common bus system load should be selected by selection lines S0, S1, and S2.

****

**Fig:-** Diagram of Basic Computer Structure

Arithmetic and Logic Unit (ALU)

An arithmetic unit, or ALU, enables computers to perform mathematical operations on binary numbers. Our computer ALU has connected to the Input register, a Data register, and Accumulator and gives output in Accumulator. ALU in our computer can perform AND, ADD, Compliment, Shift LEFT, Shift RIGHT, and many more operations. The specialty of our ALU is that it works bit by bit, meaning we have to construct 16 ALU blocks in order to create the whole ALU; full construction of ALU will be discussed in this chapter after some more topics.

Common Clock

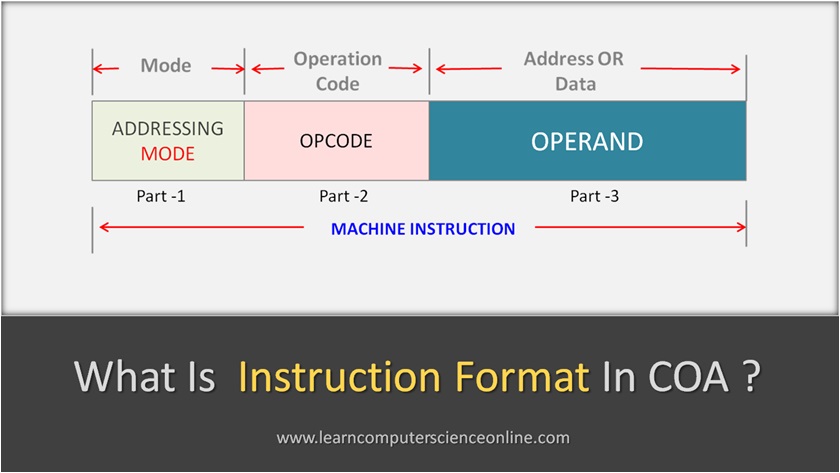
Computers use a common clock to synchronize all of their calculations. The common clock ensures that the various circuits inside a computer work together at the same time. Clock speed is measured by how many ticks per second the clock makes. The unit of measurement is called a hertz (Hz). The common clock is connected to all registers, ram and flip-flops, and other memory-related components.

Instruction Code

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

An instruction comprises groups called fields. These fields include: -

* The Mode field which specifies how the operand will be located i.e, Direct / Indirect.
* The Opcode field specifies the operation to be performed.
* The Address field or Operation field which contains the location of the operand in case of memory reference or in other cases these bits are used to decode further operations.



Modes in instruction:

There are two types of modes or addressing in instruction code.

* Direct Addressing is a scheme in which the address specifies which memory word or register contains the operand.
* Indirect Addressing is a scheme in which the address specifies which memory word or register contains not the operand but the address of the operand.

Types of Instructions

A basic computer has three instruction code formats which are:

1. Memory - reference instruction

2. Register - reference instruction

3. Input-Output instruction

Memory - reference instruction

In Memory reference the 15th bit is denoted as Direct and Indirect Addressing. 3 bits ( 14 - 12 ) denote the Opcode of the instruction and the rest 12 bits (11 - 0 ) denote the address from where we get data on which operation has to be done. We have a total of 7 operations to do in Memory reference instruction.

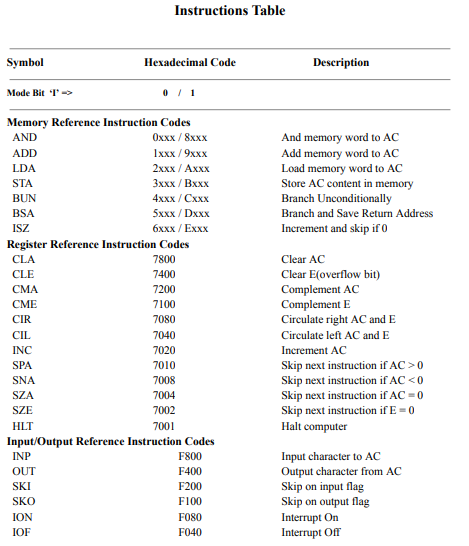
In our computer system, the Opcode range is between 000 to 110 which is divided into 7 different operations. We discuss them further in this chapter.

Register - reference instruction

In register reference, the 15th bit is 0 and Opcode bits are 111 for all cases. The rest 12 bits are the Operation bits that derive which operation has to be done. There are a total of 12 operations that are derived from these Operation bits. Any of the 12 bits becomes 1 else the other should be 0 and the position of this 1 bit derives operation in Register Reference instruction.

Input / Output - reference instruction

In Input / Output reference, the 15th bit is 1, and Opcode bits are 111 for all cases. The rest 12 bits are the Operation bits that derive which operation has to be done similar to Register Reference but the difference is that there are only 6 operations that are derived from these Operation bits. Any of the ( 11 - 6 ) bits becomes 1 else the other should be 0 and the position of this 1 bit derives operation in Input / Output Reference instruction.



Time and Control Unit

Time Unit

The operation performs some micro-operations at every clock pulse. The sequence counter (SC) counts from 0000 to 1111. The output of the sequence counter is fed to a 4X16 decoder which decodes the input binary number and generates a timing signal which will be called T0, T1, T2, T3, and so on in the sections to follow. Thus the timing unit generates a new timing signal every clock cycle. The SC can be incremented or cleared synchronously.

Control Unit

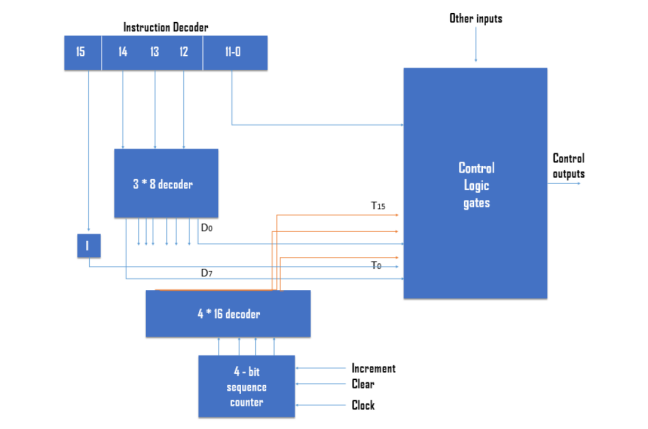
An instruction read from memory is placed in the instruction register (IR), where it is divided into three parts: the I bit, the operation code, and bits 11 through 0. The I bit is the most significant bit (MSB) of the instruction code.

The I bit is stored in the flip-flop. The operation code in bits 12 through 14 is decoded with a 3X8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7.

The output of the decoder specifies the operation to be done. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to the control logic gates.

**3X8 Decoder => D0, D1, D2, D3……… D7**.

These **D0, D1, D2…..** are the Opcode decoded values that provide operation details.



Flags Used in Our System

Total of 7 flags or flip flops are used for specific purposes, those flags are.

* E flag: used for storing overflow value coming from ALU.
* I flag: used for storing Mode decode from IR.
* S flag: used to define whether all instructions are completed or not.If S = 1 then instructions are yet not completed, if S = 0 then the program has been stopped because all instruction has been completed.
* R flag: used to decide whether the instruction is in the Instruction cycle or in the Interrupt cycle.
* IEN flag: used to denote whether there is an interrupt from an external input/output device.
* FGI flag: used to denote whether there is new input from input devices or not.
* FGO flag: used to denote whether there is a new request from output devices or not.

**LOGISIM**

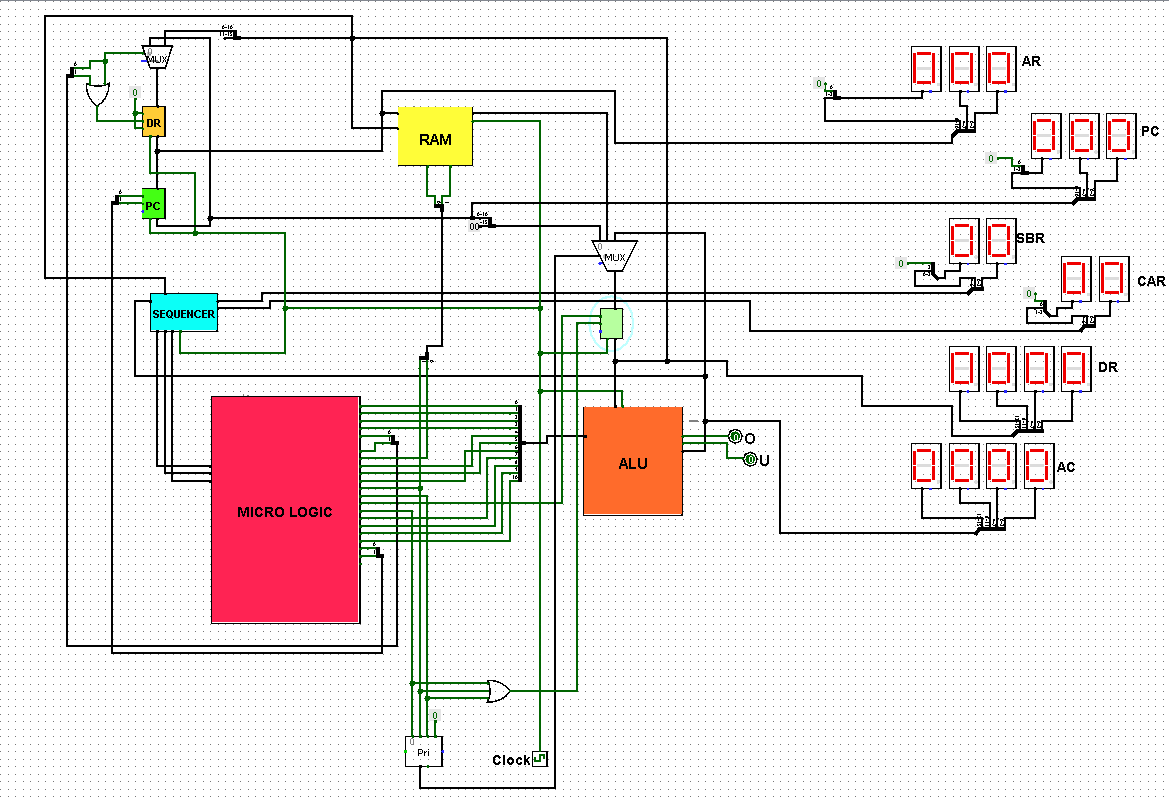
Logisim is an educational tool for designing and simulating digital logic circuits. With its simple toolbar interface and simulation of circuits as you build them, it is simple enough to facilitate learning the most basic concepts related to logic circuits. With the capacity to build larger circuits from smaller subcircuits, and to draw bundles of wires with a single mouse drag, Logisim can be used (and is used) to design and simulate entire CPUs for educational purposes.

## Features

* It is free! (Logisim is open-source.)
* It runs on *any* machine supporting Java 5 or later; special versions are released for MacOS X and Windows. The cross-platform nature is important for students who have a variety of home/dorm computer systems.
* The drawing interface is based on an intuitive toolbar. Color-coded wires aid in simulating and debugging a circuit.
* The wiring tool draws horizontal and vertical wires, automatically connecting to components and to other wires. It's very easy to draw circuits!
* Completed circuits can be saved into a file, exported to a GIF file, or printed on a printer.
* Circuit layouts can be used as "subcircuits" of other circuits, allowing for hierarchical circuit design.
* Included circuit components include inputs and outputs, gates, multiplexers, arithmetic circuits, flip-flops, and RAM memory.

**Benefit of logisim to use in the Implementation of computer 1.**

* Digital Circuit Design: Logisim provides a graphical interface for designing digital circuits using logic gates, multiplexers, decoders, flip-flops, and other digital components. This makes it well-suited for designing the various components of a hardwired computer, such as the ALU (Arithmetic Logic Unit), control unit, registers, and memory units.
* Visual Representation: Logisim allows designers to create circuits visually, which is particularly useful for understanding and debugging complex circuit designs. This is crucial when designing the intricate control logic and data pathways found in hardwired computer architectures.
* Simulation Capabilities : Logisim includes simulation features that enable the designer to test and verify the functionality of their digital circuits. This is essential for ensuring that the various components of a hardwired computer work correctly together before moving to physical implementation.
* Cost-effective Prototyping : Before committing to physical hardware implementation, Logisim allows designers to prototype and refine their designs virtually. This can save time and resources by identifying potential issues early in the design phase.
* Flexibility and Customization : Logisim supports custom component creation and allows for the creation of complex hierarchical designs. This flexibility is important when designing the unique architecture of a hardwired computer system.



**Instruction Cycle**

A program residing in the memory unit of a computer consists of a sequence of instructions. These instructions are executed by the processor by going through a cycle for each instruction.

In a basic computer, each instruction cycle consists of the following phases:

1. Fetch the instruction from memory.

2. Decode the instruction.

3. Read the effective address from memory.

4. Execute the instruction

Fetch

T0: AR ← PC

T1: IR ← M[AR], PC ← PC+1

Decode

T2: I ← IR[15] , Decode D0, D1, D2, D3…..D7 ← IR[14 - 12] , AR ← IR[11-0]

Indirect Address Fetching

T3: AR ← M[AR]

**Instructions**

Memory Reference Instructions

The control function in those operations should be written as DxTx: Micro Operation. These 7 Memory Reference Operations are

D0: AND

D1: ADD

D2: LDA

D3: STA

D4: BUN

D5: BSA

D6: SUB

AND operation

This is an instruction that performs the AND logic operation on bits in AC and the memory data at the effective address.

T0: AR ← PC

T1: IR ← M[AR], PC ← PC+1

T2 : I ← IR[15] , Decode D0, D1, D2, D3…..D7 ← IR[14 - 12] , AR ← IR[11-0]

D7’IT3: AR ← M[AR]

D0T4: DR ← M[AR]

D0T5: AC← AC ^ DR, SC← 0

ADD / Addition Operation

This is an instruction that performs the ADD logic operation on bits in AC and the memory data at the effective address.

T0: AR ← PC

T1: IR ← M[AR], PC ← PC+1

T2 : I ← IR[15] , Decode D0, D1, D2, D3…..D7 ← IR[14 - 12] , AR ← IR[11-0]

D7’IT3: AR ← M[AR]

D1T4: DR ← M[AR]

D1T5: AC← AC + DR, E ← Cout, SC← 0

LDA / Load Operation

This instruction transfers the memory data of an effective address into AC. Simply we do load operations in AC

T0: AR ← PC

T1: IR ← M[AR], PC ← PC+1

T2 : I ← IR[15] , Decode D0, D1, D2, D3…..D7 ← IR[14 - 12] , AR ← IR[11-0]

D7’IT3: AR ← M[AR]

D2T4: DR ← M[AR]

D2T5: AC ← DR, SC ← 0

STA / Store Operation

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

T0: AR ← PC

T1: IR ← M[AR], PC ← PC+1

T2 : I ← IR[15] , Decode D0, D1, D2, D3…..D7 ← IR[14 - 12] , AR ← IR[11-0]

D7’IT3: AR ← M[AR]

D3T4: M[AR] ← AC, SC ← 0

BUN / Branch Unconditional Operation

This instruction transfers the program to instruction at the effective address. The BUN instruction simply allows the user to jump or branch unconditionally, in which the PC transfers or jumps to another address. Simply skip or jump through some operations.

T0: AR ← PC

T1: IR ← M[AR], PC ← PC+1

T2 : I ← IR[15] , Decode D0, D1, D2, D3…..D7 ← IR[14 - 12] , AR ← IR[11-0]

D7’IT3: AR ← M[AR]

D4T4: PC ← AR, SC ← 0

BSA / Branch and Save Return Operation

This instruction is useful for branching to a portion of the program called a subroutine or procedure. In this operation, we branch into a new address and save there that from which address this instruction gets branched. After the execution of a specific program, the instruction branch/returns back to the initial program control.

T0: AR ← PC

T1: IR ← M[AR], PC ← PC+1

T2 : I ← IR[15] , Decode D0, D1, D2, D3…..D7 ← IR[14 - 12] , AR ← IR[11-0]

D7’IT3: AR ← M[AR]

D5T4: M[AR] ← PC, AR ← AR + 1

D5T5: PC ← AR, SC← 0

SUB / Subtraction Operation

This is an instruction that performs the SUB logic operation on bits in AC and the memory data at the effective address.

T0: AR ← PC

T1: IR ← M[AR], PC ← PC+1

T2 : I ← IR[15] , Decode D0, D1, D2, D3…..D7 ← IR[14 - 12] , AR ← IR[11-0]

D7’IT3: AR ← M[AR]

D6T4: DR ← M[AR]

D6T5: AC ← AC – DR , SC ← 0

Register Reference Instructions

Generally register reference operations have an Opcode of D7 and mode bit I = 0. The 12 operations of register reference is differentiated by their 12 bits. Any of the one bit is 1 in operation bit of register reference instruction.

The control function in those operations should be written as D7I’T3Bx: Micro Operation.

Where Bx refers to the bit number of operation bits(11-0) whose value is 1.

These 12 Memory Reference Operations are :

CLA:

D7I’T3B11: AC ← 0 Clear AC

CLE :

D7I’T3B10: E ← 0 Clear E

CMA:

D7I’T3B9: AC ← AC’ Complement AC

CME:

D7I’T3B8: E ← E’ Complement E

CIR:

D7I’T3B7: AC ← shr AC, AC(15) ← E, E ← AC(0) Circular Right

CIL:

D7I’T3B6: AC ← shl AC, AC(0) ← E, E ← AC(15) Circular Left

INC:

D7I’T3B5: AC ← AC + 1 Increment AC S

SPA:

D7I’T3B4: if (AC(15) = 0) then (PC ← PC+1) Skip if positive

SNA:

D7I’T3B3: if (AC(15) = 1) then (PC ← PC+1 Skip if negative

SZA:

D7I’T3B2: if (AC = 0) then (PC ← PC+1) Skip if AC is zero

SZE :

D7I’T3B1: if (E = 0) then (PC ← PC+1) Skip if E is zero

HLT:

D7I’T3B0: S ← 0 (S is a start-stop flip-flop) Halt computer

After all micro-operation D7I’T3Bx include SC ← 0. These all are the 12 Register Reference instructions, one can notice that these all operations are related to AC and E only, and in these operations, there is no access from a memory location or address calling. That is why this type of instruction is called register reference

Input / Output Reference Instruction

Generally Input / Output reference operations have an Opcode of D7 and mode bit I = 1. The 6 operations of Input / Output reference are differentiated by their 12 bits. Any of the one bit is 1 in (11-6) operation bit of Input / Output reference instruction.

The control function in those operations should be written as D7IT3Bx: Micro Operation.

Where Bx refers to the bit number of operation bits(11-6) whose value is 1.

These 6 Input / Output Reference Operations are :

**INP :**

D7IT3B11: AC(0-7) ← INPR, FGI ← 0 Input char. to AC

**OUT :**

D7IT3B10: OUTR ← AC(0-7), FGO ← 0 Output char. from AC

**SKI :**

D7IT3B9: if(FGI = 1) then (PC ← PC + 1) Skip on input flag

**SKO** :

D7IT3B8: if(FGO = 1) then (PC ← PC + 1) Skip on output flag

**ION :**

D7IT3B7: IEN ← 1 Interrupt enable on

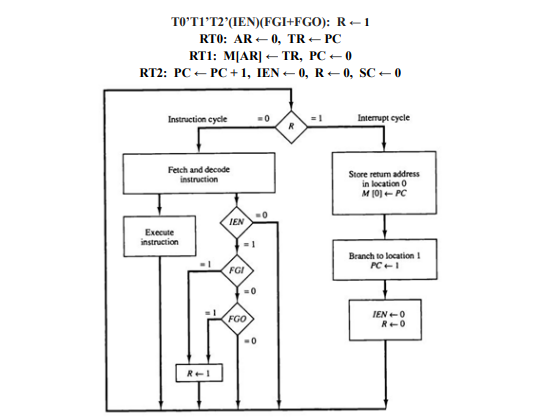
**IOF** :

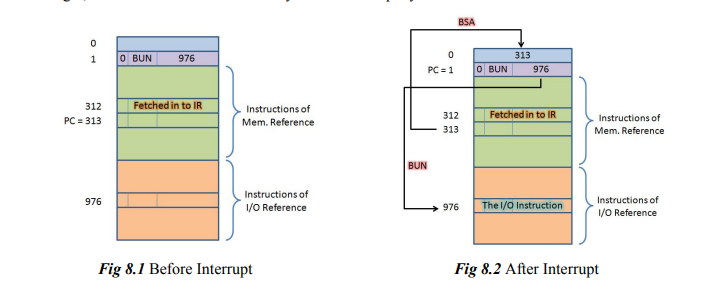
D7IT3B6: IEN ← 0 Interrupt enable off

After all micro-operation D7I’T3Bx include SC ← 0.

**Interrupt Cycle**

An interrupt flip-flop R is included in the computer. When R = 0, the computer goes through an instruction cycle. During the execution 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, the control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for the 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, the control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle. The interrupt cycle is an implementation of a branch and saves the return address operation. The return address available on the PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor register, a memory stack, or a specific memory location. Here we choose the memory location at address 0 as the place for storing the return address. Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.





The interrupt cycle is initiated after the last execution phase if the interrupt flip-flop R is equal to 1. This 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 to 1 can be expressed with the following register transfer statement.

**Construction of Basic Computer**

The basic computer consists of the following hardware components:

1. Memory 4096 X 16 Ram

2. Eight registers: AR, PC, DR, AC, IR, TR, OUTR, and INPR

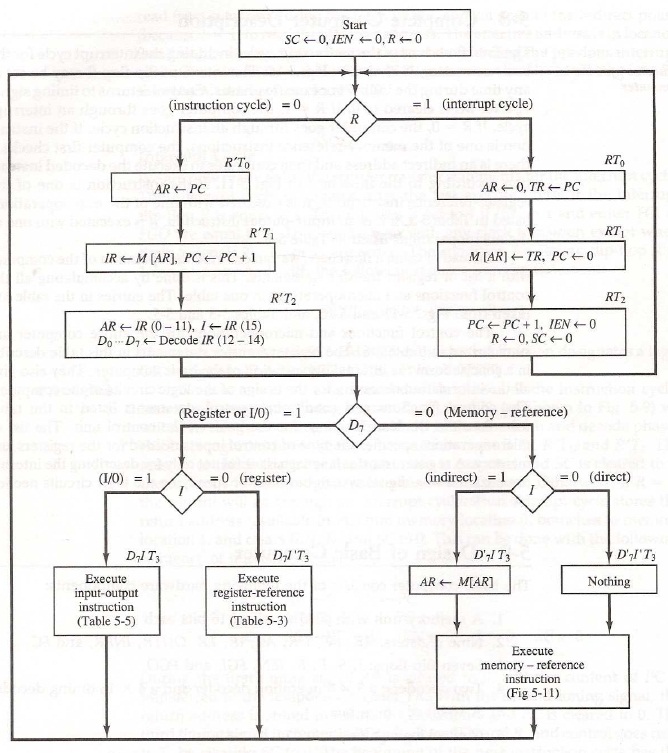
3. Seven flip-flops: I, S, E, R, lEN, FGI, and FGO

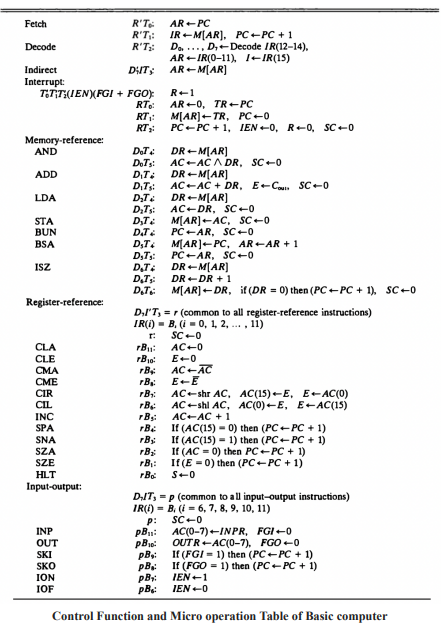
4. A 16-bit common bus

5. Two decoders: a 3 x 8 decoder for decoding opcode and a 4 x 16 timing decoder for SC

6. Control logic gates

7. Adder and logic circuit





**Control Logic**

Control logic is the combination of logic gates and circuits. These combinational circuits combine together to make a synchronously working computer.

The output of the control logic circuit is :

* Signals to control the inputs to 9 registers.
* Signals to control read and write input of the memory.
* Signal for S2, S1, and S0 to select a register for the bus.
* Signals for a flip flop.
* Signal to control the AC adder and logic circuit.

Control for Registers

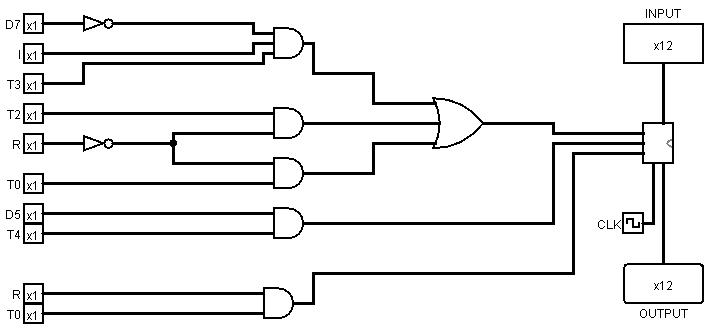
For individual registers there are load, clear, and increment controls for synchronously controlling the micro-operations in our basic computer. Those controls for 6 addresses and data registers are.

Controls for AR:

**Load:** R’T0 + R’T2 + D7’IT3

**CLR:** RT0

**INC:** D5T4

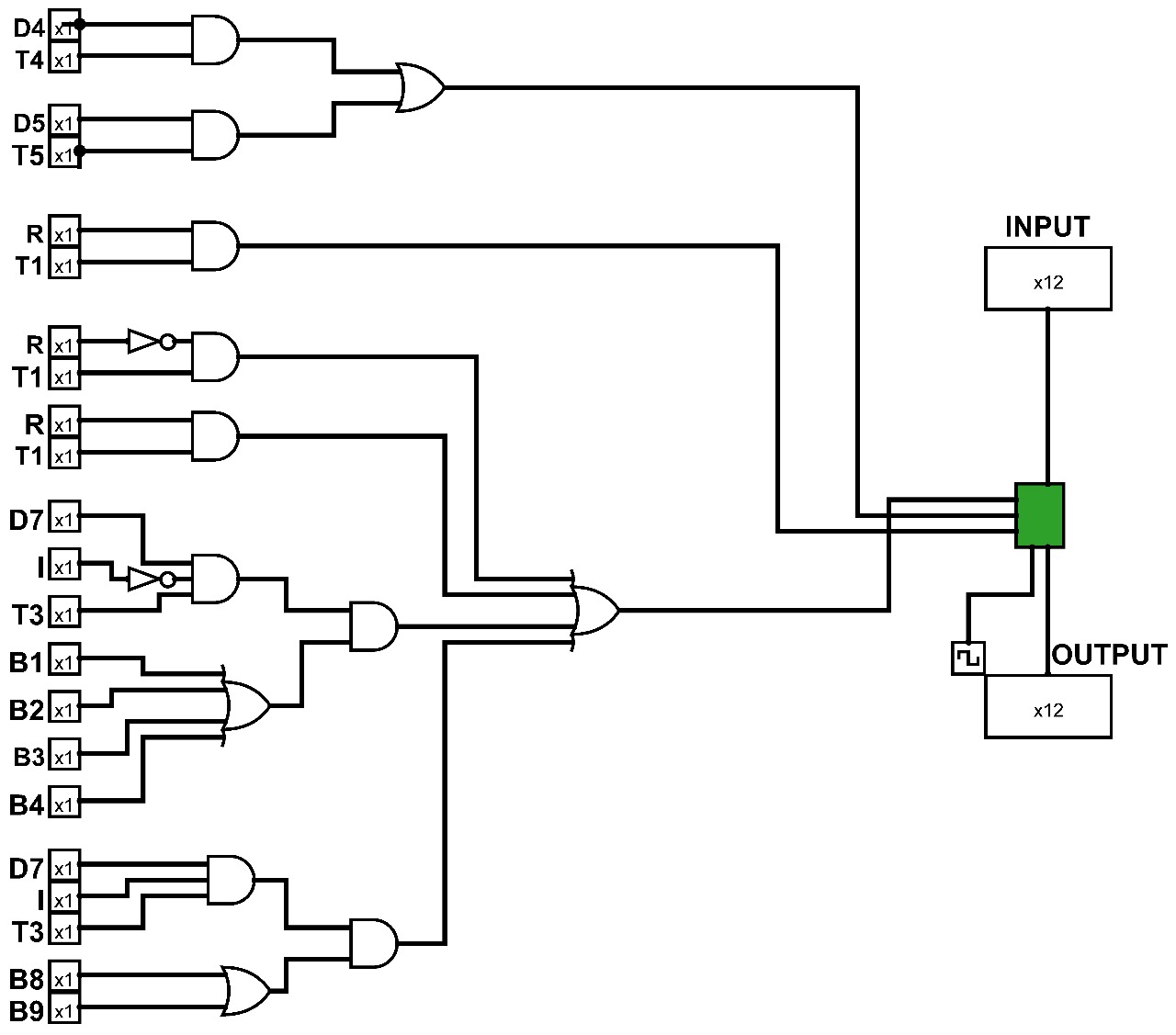


Controls for PC:

**Load:** D4T4 + D5T5

**CLR:** RT1

**INC:** R’T1 + RT2 + D7I’T3( B1 + B2 + B3 + B4)

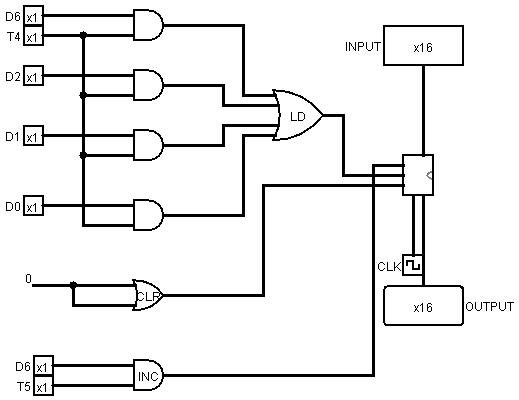


Controls for DR:

Load: D0T4 + D1T4 + D2T4 + D6T4

CLR: 0

INC: D6T5

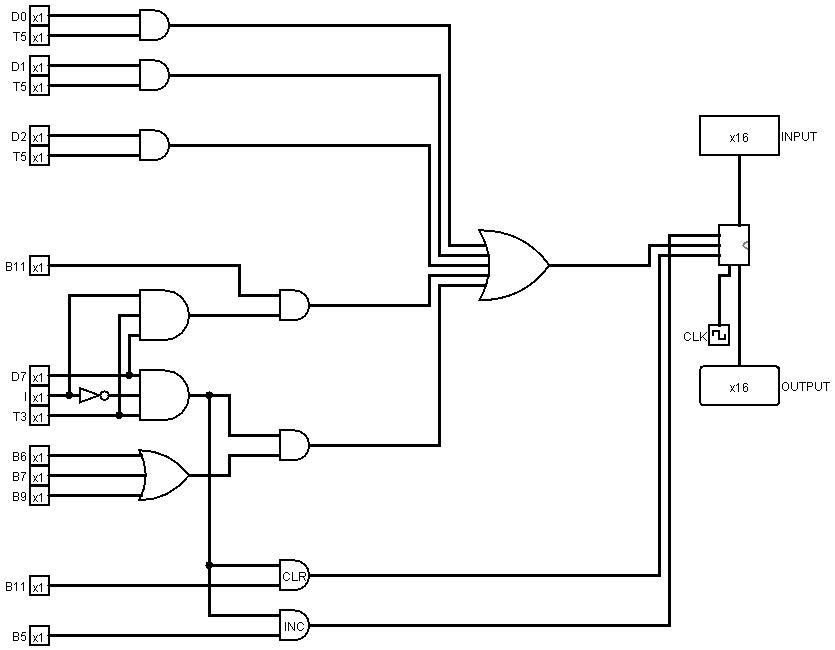


Controls for AC:

Load: D0T5 + D1T5 + D2T5 + D7I’T3( B6 + B7 + B9 )+ D7IT3B11

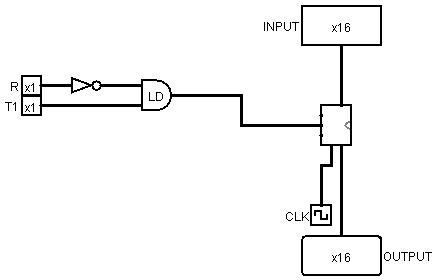
CLR: D7I’T3B11

INC: D7I’T3B5



Control for IR:

Load: R’T1

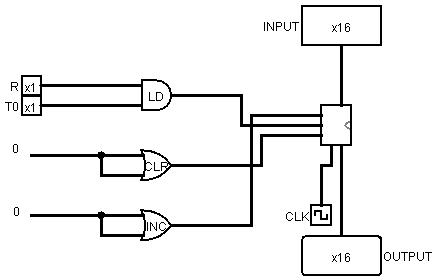


Control for TR:

Load: RT0

CLR: 0

INC: 0

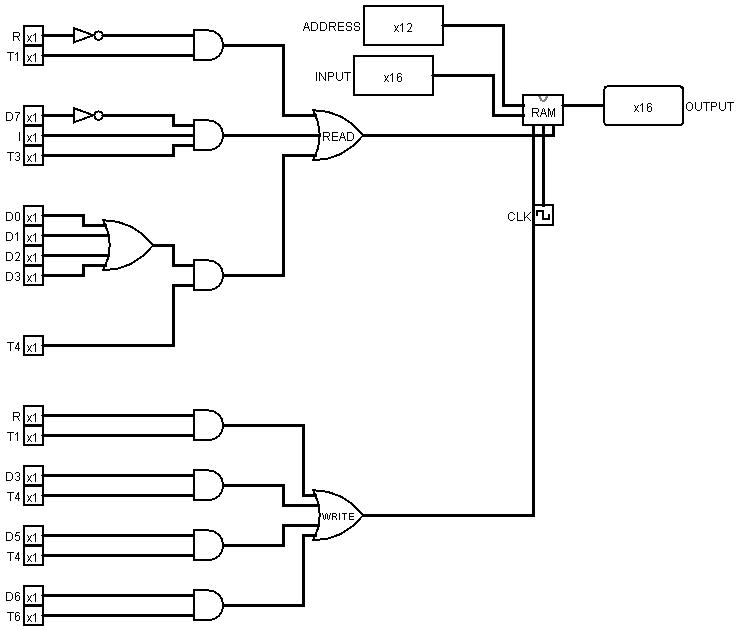


Control for RAM

For Ram there is read and write controls for synchronously controlling the micro-operations in our basic computer. Those controls for read and write operations in ram are:

Read: T4D0 + T4D1 +T4D2 + T4D6 + R’T1 + D7’IT3

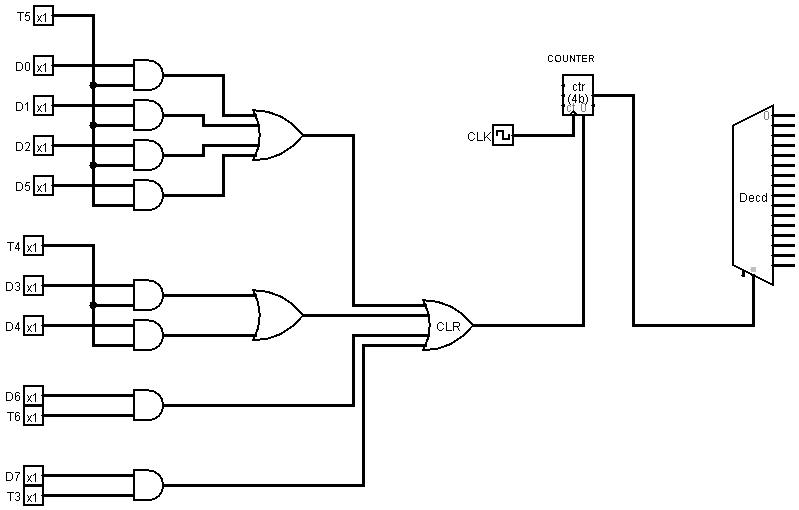
Write: RT1 + D3D4 + D5T4 + D6T6



Controls for Sequence Counter

There is only reset operation, in which Sequence counter changes to be T0. Those controls for resetting the value of the sequence counter are:

D0T5 + D1T5 +D2T5 + D5T5 + D3T4 + D4T4 + D6T6 + D7T3

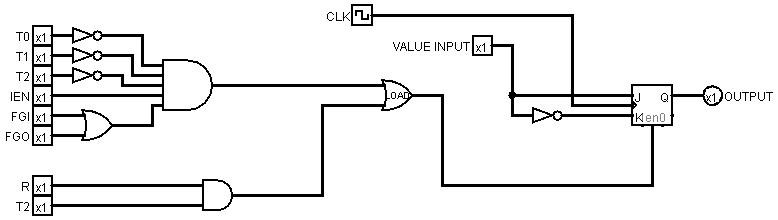


Controls for Flip Flop

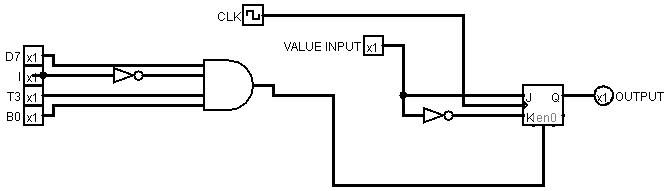
For Flip Flops there are set controls for synchronously controlling the micro-operations in our basic computer. Those controls for set value in flip flops are.

R Flag:

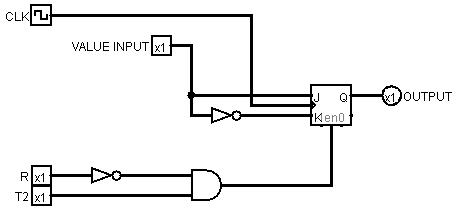
RT2 + T0’T1’T2’(IEN)(FGI + FGO)



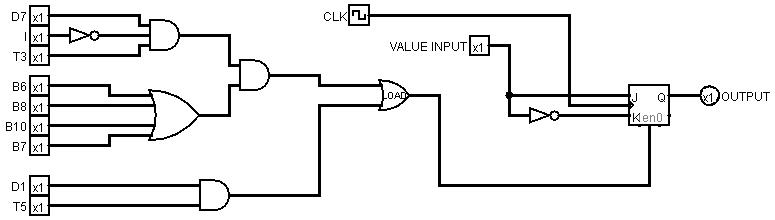
S Flag: D7I’T3B0



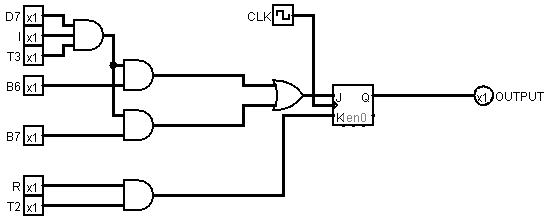
I Flag: R’T2



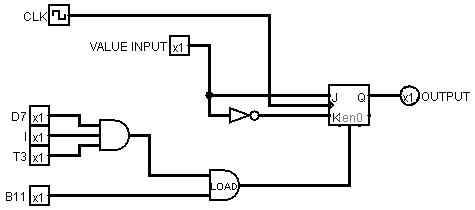
E Flag: D7I’T3(B6+ B7 +B8 + B10) + D1T5



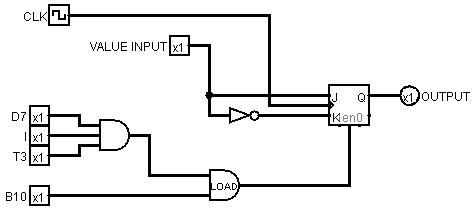
IEN Flag: RT2 + D7IT3(B6 +B7)



FGI Flag: D7IT3B11

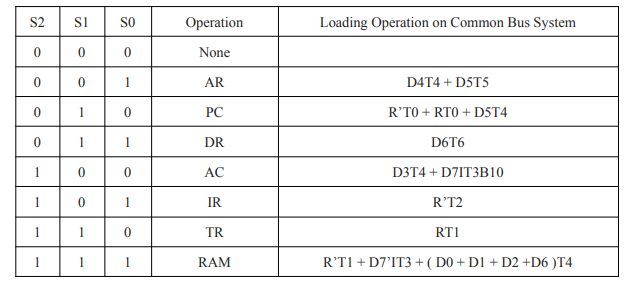


FGO Flag: D7IT3B10



Controls for Common Bus System

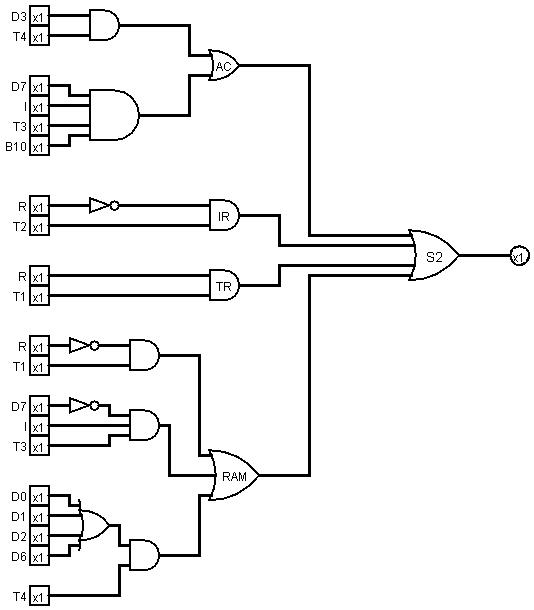
For a common bus system, it is necessary for controlling at which time where data should be loaded into it. This means at which time what should be the value of S0, S1, and S2 should be defined by some logic and combinational circuits those circuits are.



Controls for S2 :

(S2 = 1) at => CAC + CIR + CTR + CRAM

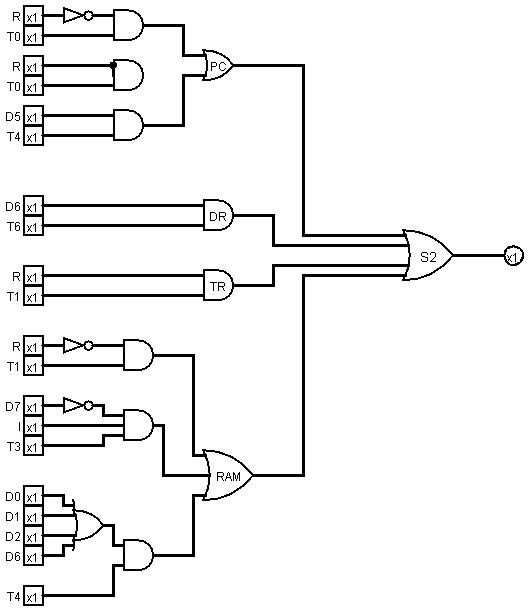
Controls for (S2 = 1) => D3T4 + D7IT3B10 + R’T2 + RT1 + R’T1 + D7’IT3 + ( D0 + D1 + D2 +D6 )T4



Controls for S1 :

(S1 = 1) at => CPC + CDR + CTR + CRAM

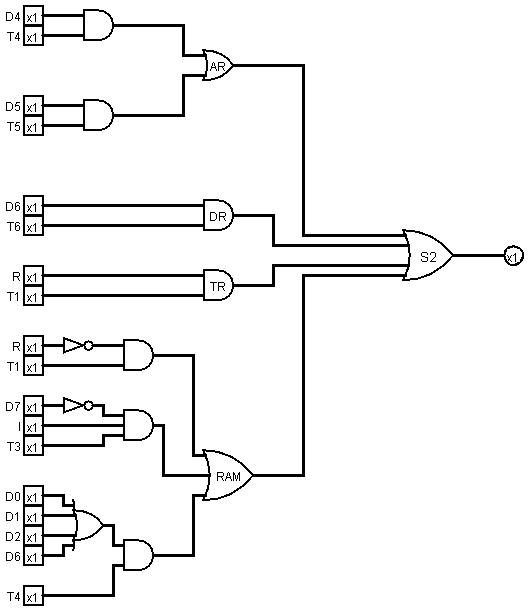
Controls for (S1 = 1) => R’T0 + RT0 + D5T4 + D6T6 + RT1 + R’T1 + D7’IT3 + ( D0 + D1 + D2 +D6 )T4



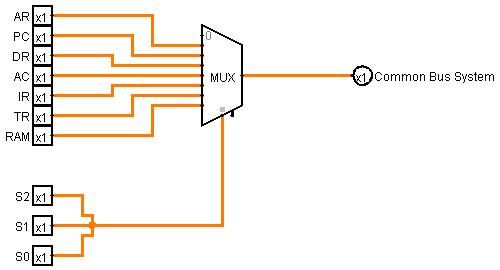
Controls for S0 :

(S0 = 1) at => CAR + CDR + CIR + CRAM

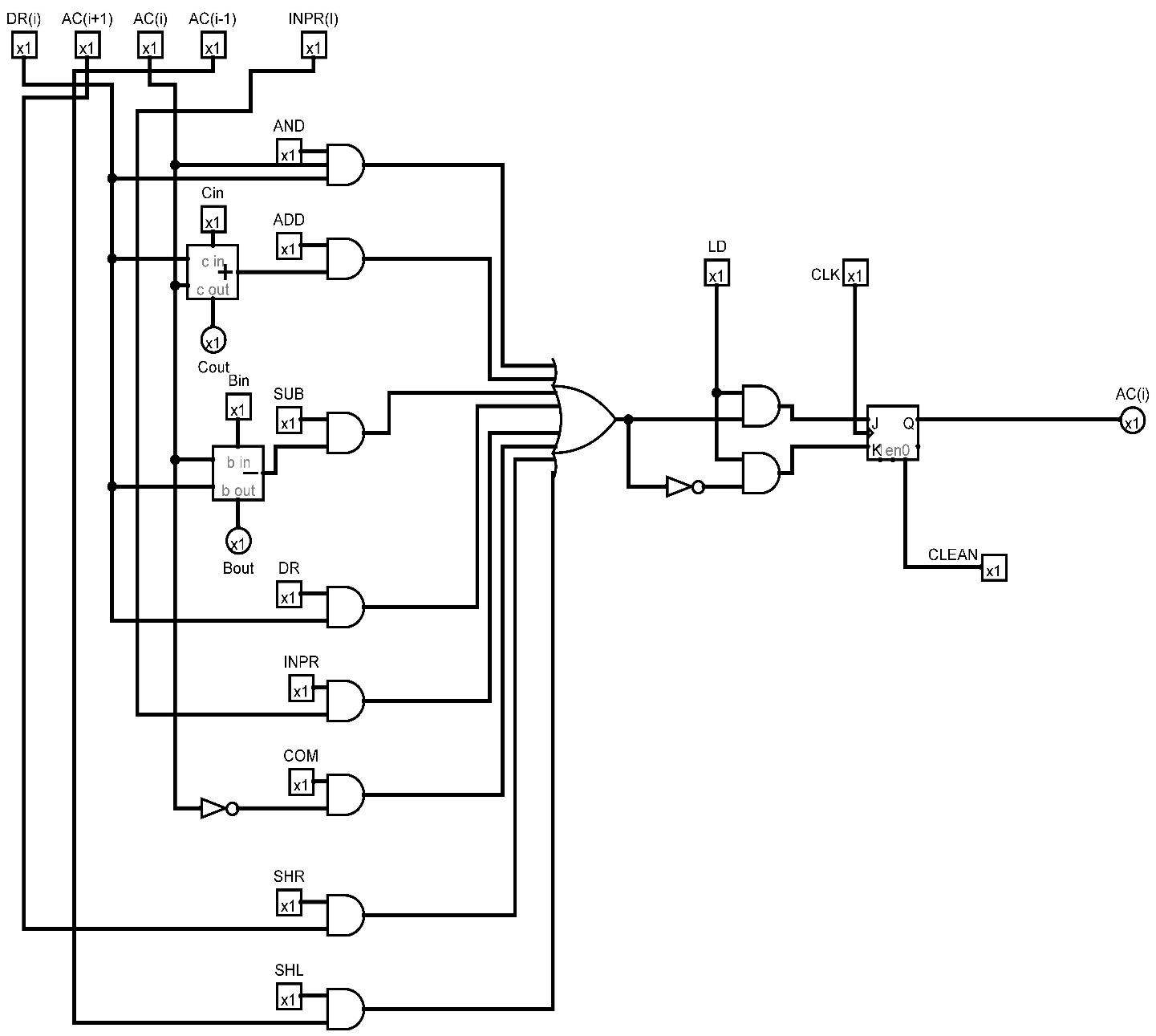
Controls for (S0 = 1) => D4T4 + D5T5 + D6T6 + R’T2 + R’T1 + D7’IT3 + ( D0 + D1 + D2 +D6 )T4

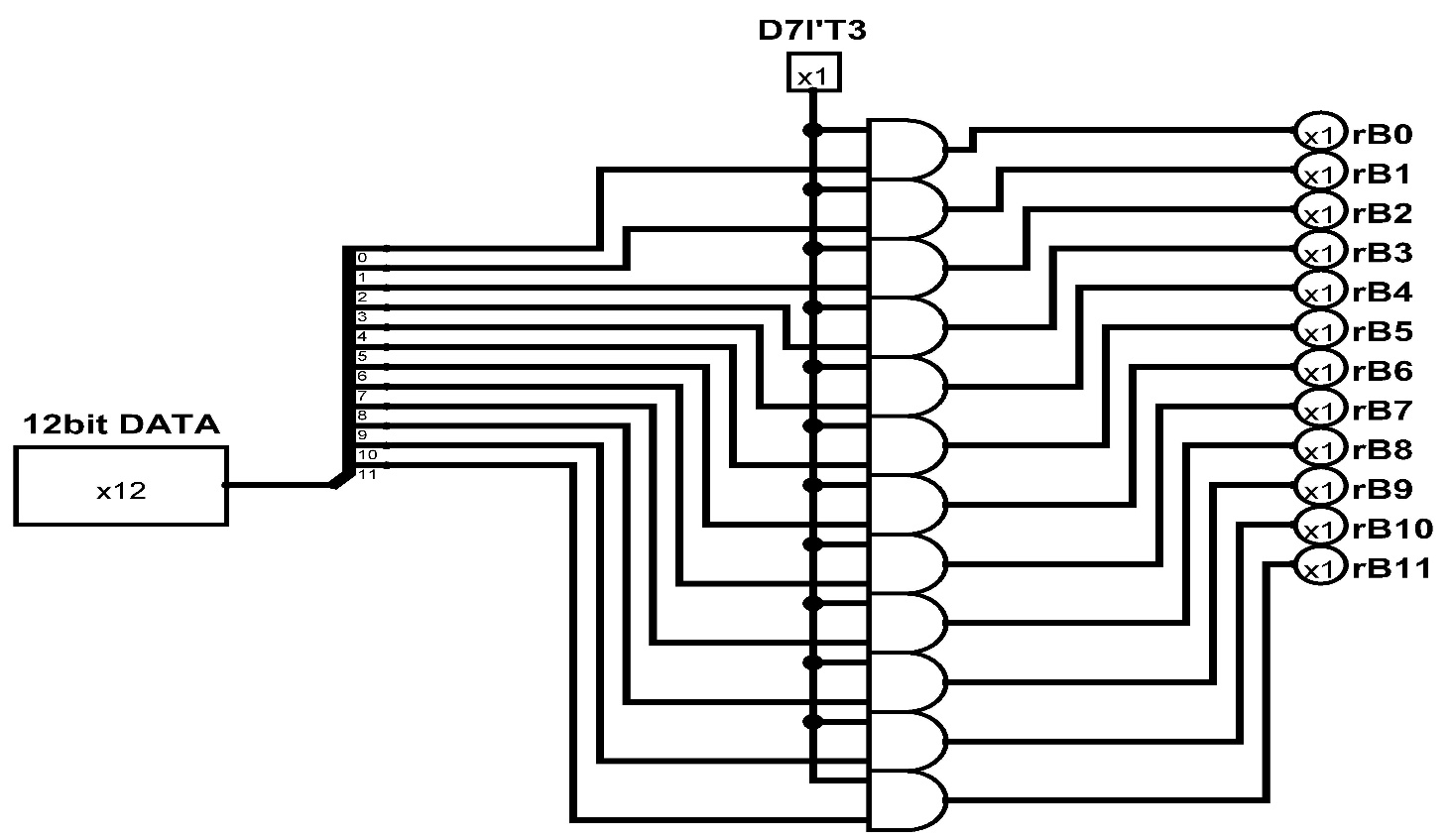


These all operations define whenever the values of S0, S1, S2 have to 1 / 0 so that one of the inputs of Multiplexer is taken as an input in the common bus



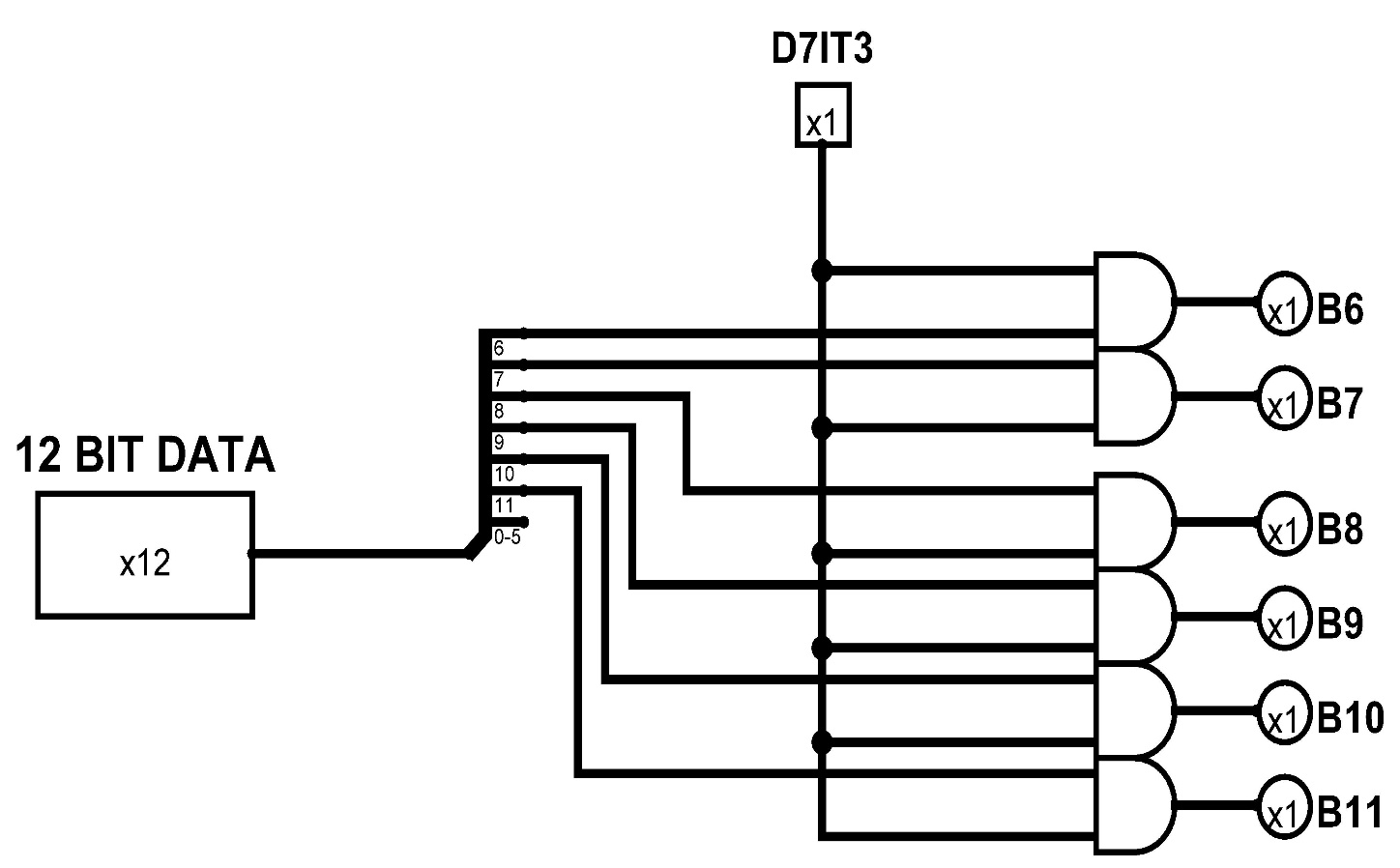
**Implementation of 1 – BIT ALU Block**

****

****

**Implementation of REGISTER REREFERANCE**

**Implementation of I / O REFERANCE**

****

CU , ALU AND REGISTER :

ALU is a digital circuit that provides arithmetic and logic operations. It is the fundamental building block of the central processing unit of a computer. A modern central processing unit(CPU) has a very powerful ALU and it is complex in design. In addition to ALU modern CPU contains a control unit and a set of registers. Most of the operations are performed by one or more ALUs, which load data from the input register. Registers are a small amount of storage available to the CPU. These registers can be accessed very fast. The control unit tells ALU what operation to perform on the available data. After calculation/manipulation, the ALU stores the output in an output register.

