**Microprogrammed Control**

The function of the control unit in a digital computer is to initiate sequences of microoperations. The number of different types of microoperations that are available in a given system is finite. The complexity of the digital system is derived from the number of sequences of microoperations that are performed. When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired. Microprogramming is a second alternative for designing the control unit of a digital computer. The principle of microprogramming is an elegant and systematic method for con- trolling the microoperation sequences in a digital computer.

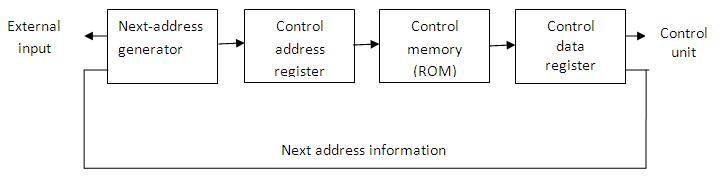
***1.Microoperations:*** -

* In computer central processing units, micro-operations (also known as a micro-ops) are detailed low-level instructions used in some designs to implement complex machine instructions (sometimes termed macro-instructions in this context).

***2.Micro instruction:*** -

* A symbolic microprogram can be translated into its binary equivalent by means of an assembler.
* Each line of the assembly language microprogram defines a symbolic microinstruction.
* Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD.

***3.Organization of micro programmed control unit: -***

* The general configuration of a micro-programmed control unit is demonstrated in the block diagram of Figure 4.1.
* The control memory is assumed to be a ROM, within which all control information is permanently stored.

***4.Address Sequencing: -***

* Microinstructions are stored in control memory in groups, with each group specifying a ***routine****.*
* To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computer instruction.

**Step-1:**

* An initial address is loaded into the control address register when power is turned on in the computer.
* This address is usually the address of the first microinstruction that activates the

instruction fetch routine.

* The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.
* At the end of the fetch routine, the instruction is in the instruction register of the

computer.

**Step-2:**

* The control memory next must go through the routine that determines the effective

address of the operand.

* A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.
* The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.
* When the effective address computation routine is completed, the address of the operand is available in the memory address register.

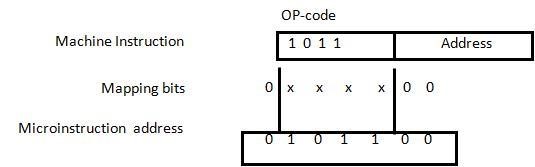
**Step-3:**

* The next step is to generate the microoperations that execute the instruction fetched from memory.
* The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.
* The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a ***mapping*** process.
* A mapping procedure is a rule that transforms the instruction code into a control memory address.

**Step-4:**

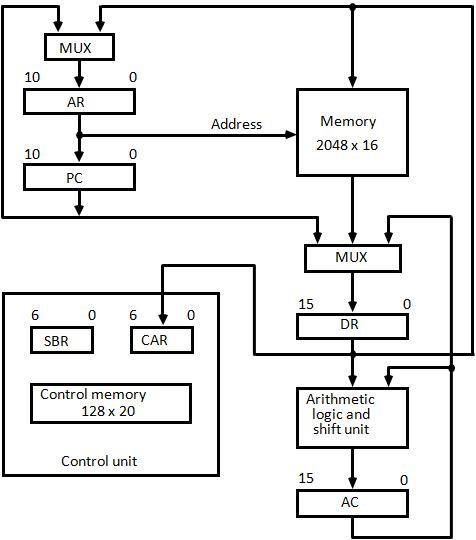
* Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.
* Micro-programs that employ subroutines will require an external register for storing the return address.
* Return addresses cannot be stored in ROM because the unit has no writing capability.
* When the execution of the instruction is completed, control must return to the fetch routine.
* This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.

***5.Mapping of an Instruction: -***



**Figure 4.3: Mapping from instruction code to microinstruction address**

***6.Computer Hardware Configuration: -***



**Figure 4.4: Computer hardware configuration**

The block diagram of the computer is shown in Figure 4.4. It consists of

1. ***Two memory units:***

Main memory -> for storing instructions and data, and

Control memory -> for storing the microprogram.

1. ***Six Registers:***

Processor unit register: AC(accumulator),PC(Program Counter), AR(Address Register), DR(Data Register)

Control unit register: CAR (Control Address Register), SBR(Subroutine Register)

1. ***Multiplexers:***

The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

1. ***ALU:***

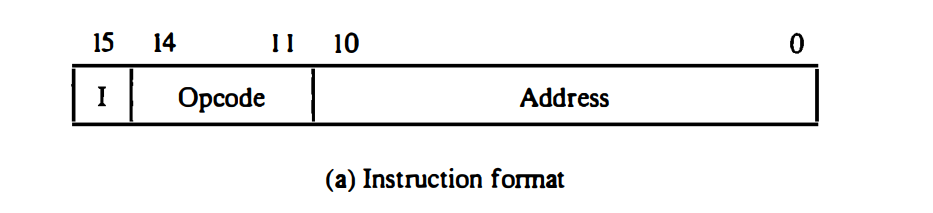
The arithmetic, logic, and shift unit performs microoperations with data from AC and DR and places the result in AC.

* DR can receive information from AC, PC, or memory.
* AR can receive information from PC or DR.
* PC can receive information only from AR.
* Input data written to memory come from DR,and data read from memory can go only to DR.

***7.Instruction Format: -***

The computer instruction format is depicted in Fig. It consists of three fields:

* 1-bit field for indirect addressing symbolized by I.
* 4-bit operation code (opcode).
* 11-bit address field.

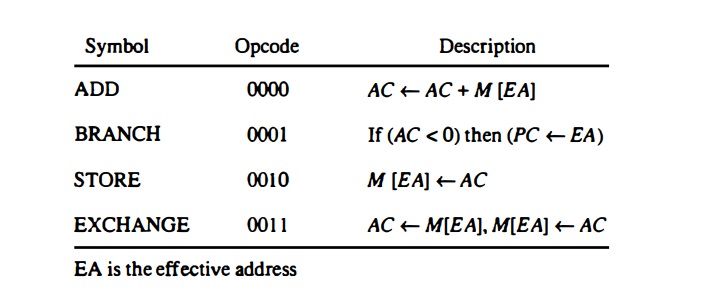


***The ADD instruction*** It adds the content of the operand found in the effective address to the content of AC.

***The BRANCH instruction*** causes a branch to the effective address if the operand in AC is negative. The program proceeds with the next consecutive instruction if AC is not negative. The AC is negative if its sign bit (the bit in the leftmost position of the register) is a 1.

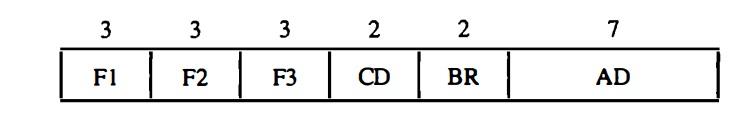
***The STORE instruction*** transfers the content of AC into the memory word specified by the effective address.

***The EXCHANGE instruction*** swaps the data between AC and the memory word specified by the effective address.



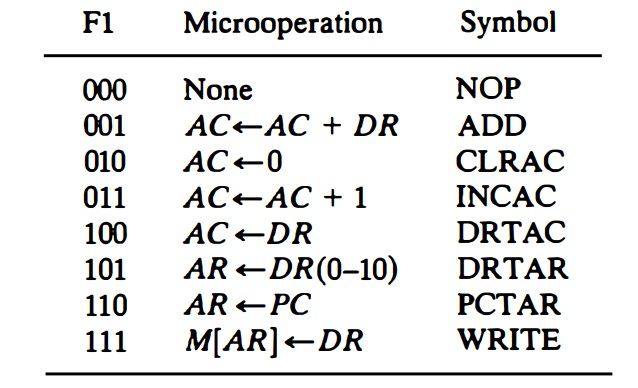
***8.Micro-instruction Format: -***

The micro-instruction format for the control memory is shown in Fig. The 20 bits of the microinstruction are divided into four functional parts. The three fields F1, F2, and F3 specify microoperations for the computer. The CD field selects status bit conditions. The BR field specifies the type of branch to be used. The AD field contains a branch address. The address field is seven bits wide, since the control memory has 128 = 27 words.

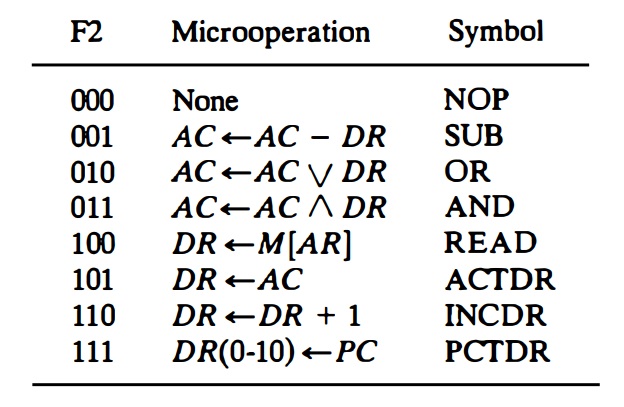
****

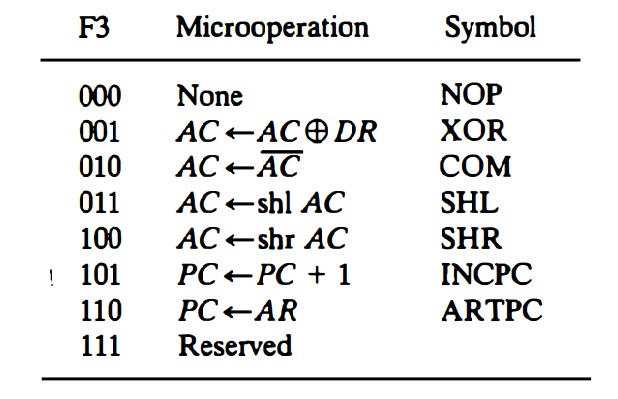
***Symbols and Binary Code for Microinstruction Fields:***

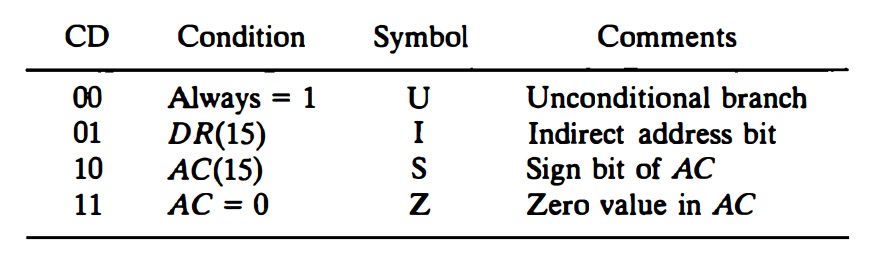
1. For F1 : -



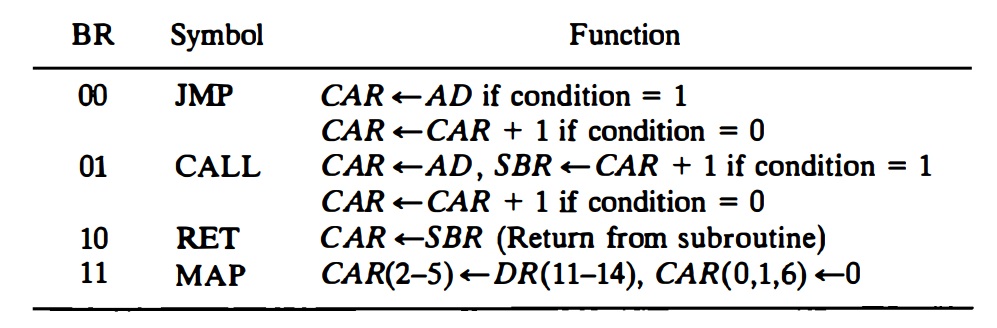
1. For F2 : -



1. For F3 : -
2. For CD : -



1. For BR : -



***9.Subroutines: -***

1. FETCH AND DECODE :

Micro-Operation: AR <- PC

DR <-M[AR], PC <- PC + 1

AR <- DR(0-10), CAR(2-5) <- DR(l l-14), CAR(0, 1,6) <-0

Micro-Instruction: PCTAR U JMP NEXT

READ , INCPC U JMP NEXT

DRTAR U MAP

1. INDIRECT :

Micro-Instructions: READ U JMP NEXT

DRTAR U RETURN

1. ADD :

Micro-Operation: DR🡨M[EA]

AC🡨DR+AC

Micro-Instruction: NOP I CALL INDIRECT

READ U JMP NEXT

ADD U JMP FETCH

1. STORE :

Micro-Operation: M[EA]🡨AC

Micro-Instruction: NOP I CALL INDIRECT

ACTDR U JMP NEXT

WRITE U JMP FETCH

1. EXCHANGE :

Micro-Operation: M[EA]🡨AC

AC🡨M[EA]

Micro-Instruction: NOP I CALL INDIRECT

READ U JMP NEXT

ACTDR,DRTAC U JMP NEXT

WRITE U JMP FETCH

1. BRANCH :

Micro-Operation: If (AC<0) then PC🡨EA

Micro-Instruction: NOP S JMP XYZ

NOP U JMP FETCH

XYZ:

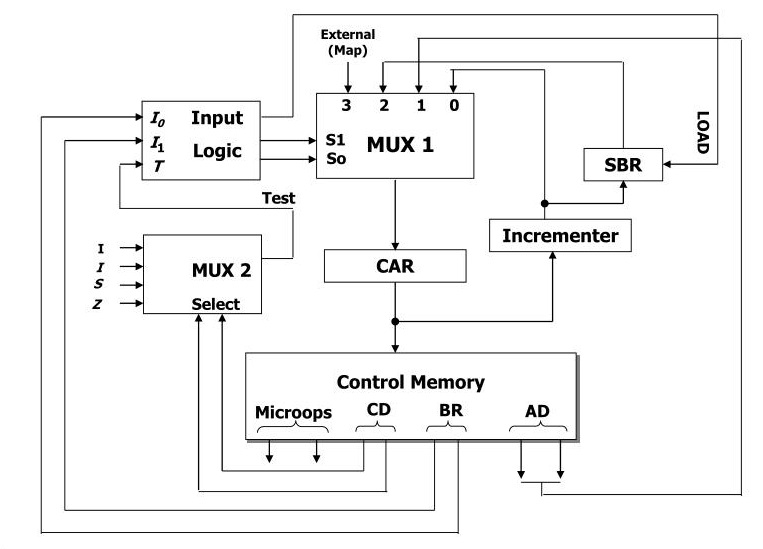
NOP I CALL INDIRECT

ARTPC U JMP FETCH

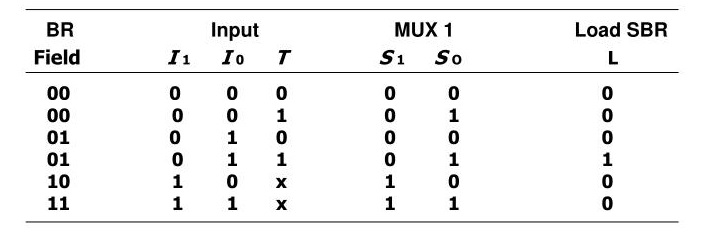
***Binary Microprogram for Control Memory (Partial)***

## 

***CONTROL UNIT***



INPUT LOGIC :

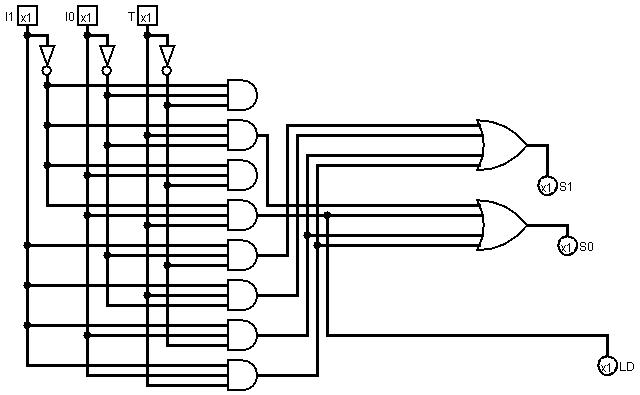


***CIRCUIT FOR INPUT LOGIC : -***

S1: I1I0’T’ + I1I0’T + I1I0T’ + I1I0T

S0: I1’I0’T + I1 ‘I0T + I1I0T’ + I1I0T

LD: I1 ‘I0T

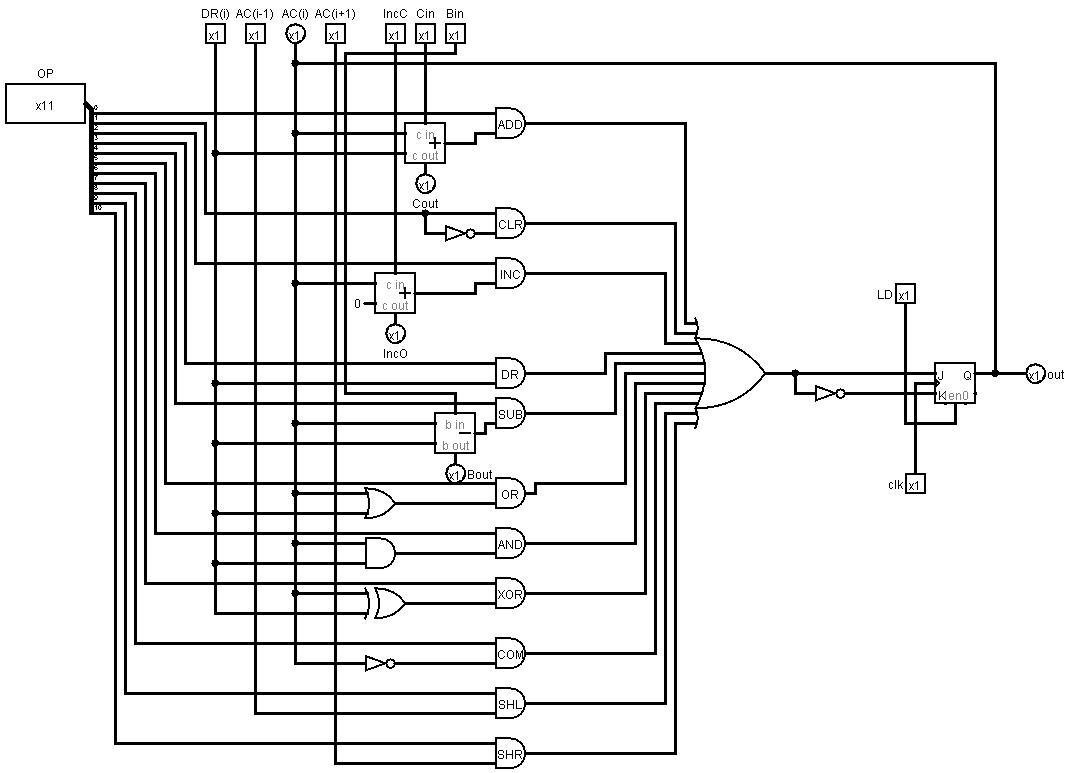


·

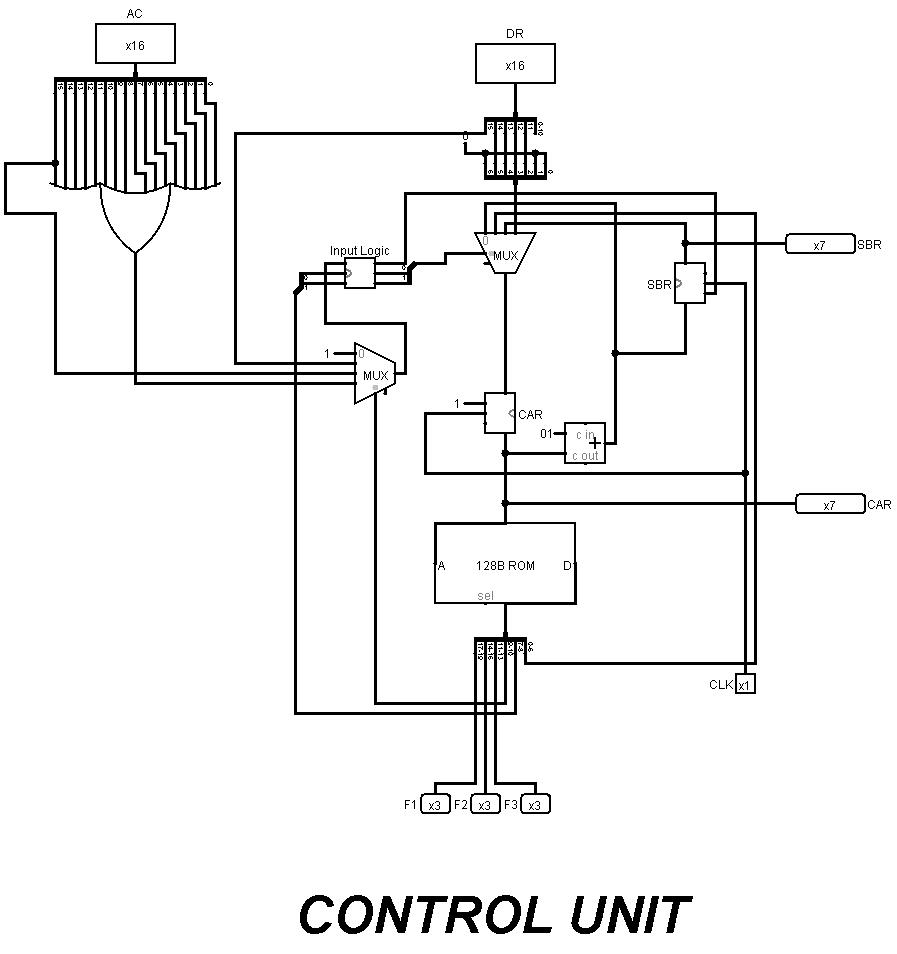
·

***ARITHEMATIC AND LOGICAL UNIT***

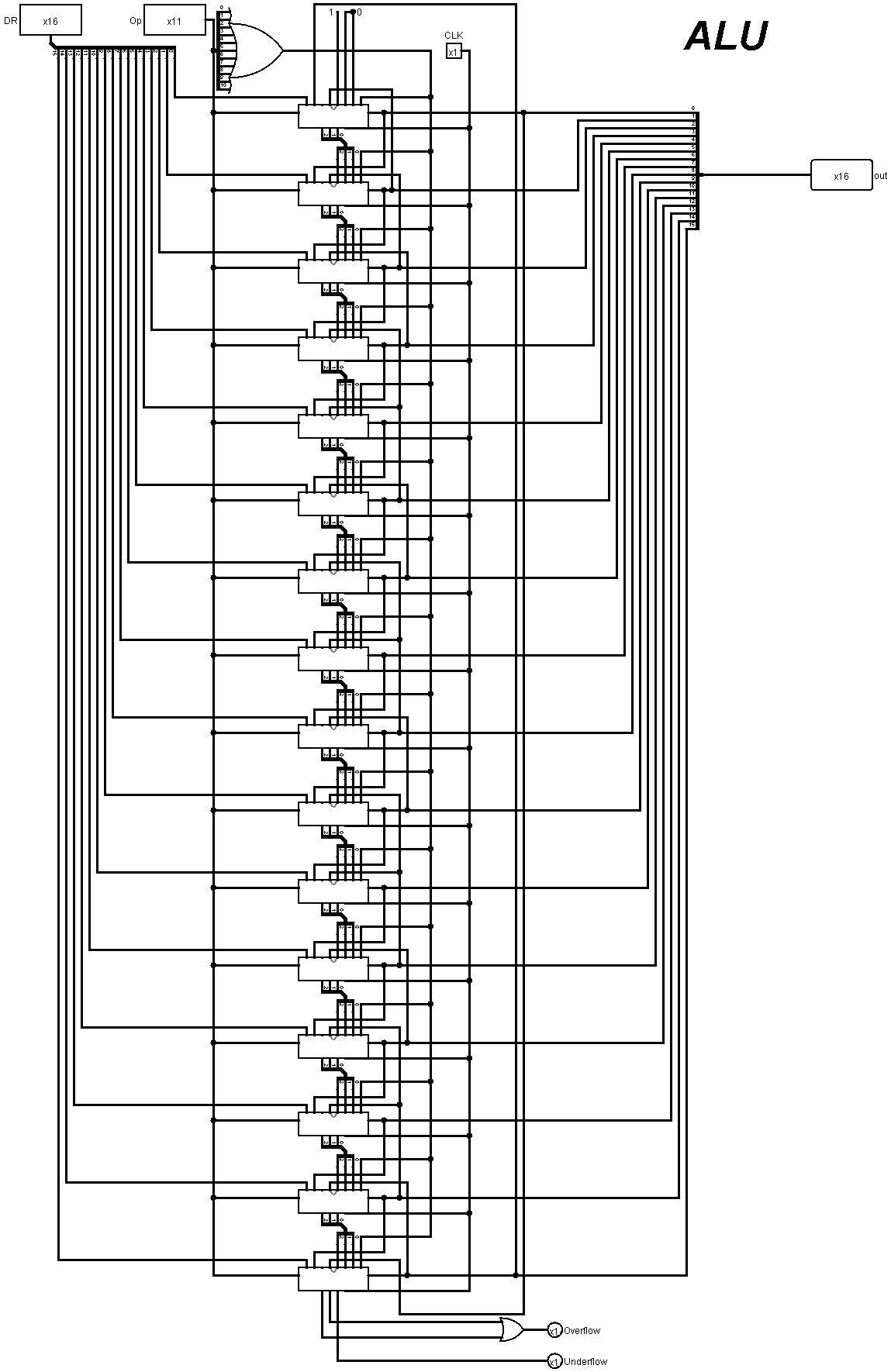
*1 Bit ALU is shown below* -



CONTROL UNIT PREVIEW



ALU PREVIEW



COMPUTER 2 PREVIEW

