PANDIT DEENDAYAL ENERGY UNIVERSITY

Laboratory Manual

Submitted to

Dr. Gangaprasad Pandey

Department of ICT Engineering
School of Technology

Submitted By: -

Name	Roll Number
Pratham Patel	22BIT003
Rushi Prajapati	22BIT013
Rudra Vyas	22BIT017
Darshan Desai	22BIT053
Heet Patel	22BIT057

Table of Content

- **Aim:** To design a 16-bit CPU based on instruction set using Logisim.

- **Assumptions:**

  - Assumption of infinite speed:

    It is not practical to assume the infinite speed as it creates problems in designers' thinking.

  - Assumption of infinite Memory:

    Storage is always finite and this is an issue in computer design. Hence memory also can't be assumed infinite.

  - Speed mismatch between memory and processor:

    The speed of memory and processor might not match sometimes. Either memory speed would be faster or processor speed would be faster. A mismatch between memory and processor leads to create problems in designing.

- Bug and error handling:

Bugs and errors may lead to the failure of the computer system. Hence handling bugs and errors is a huge responsibility for a computer designer.

- Multiple processors:

Designing a computer system with multiple processors leads to the huge task of management and programming.

- Multiple threads:

A computer with several threads should be able to multi-tasking and multiprocessing.

- Shared memory:

If there are several processes to be executed at a time then all the processes share the same memory space. It should be managed in a specific way so that collision does not happen.

- Disk access:

Disk management is the key to computer design. There are several issues with disk access. It may be possible that the system does not support multiple disk access.

- Better performance:

A designer always tries to simplify the system for better performance in reducing power and less cost.

- **Introduction:**

Procedure (in short):

- Make Registers
- Make ALU
- Make Data Path registers and ALU
- Make Control Units (Control Sequence Detector)
- Make all Registers and other logics
- Connect all the Circuits

- Connect the obtained circuit to RAM (Primary Memory)

● Working Principle:

✞ Central Processing Unit or CPU is the core control of computer which is also referred as brain of computer. CPU executes programmes and instructions to carry out operations that are been asked for. Accuracy, speed and efficiency of CPU is a base on which CPU's power is defined. With the progress in the field of computer and electronics, speed of CPU has been upgraded.

✞ Operation in CPU is done in three steps. Firstly, an instruction is read from memory. Then it is decoded, and the processor determines the process that it has to carry according to the order. Third, instruction and operation is carried out. Above written three stages are repeated in loop that starts with CPU fetching the instruction. These stages are referred as CPU's instruction cycle. Program counter is also there in CPU whose function is to keep track of which instruction to retrieve next. Address of memory region containing the next instruction to be executed is stored in counter, which is saved in register, which in turn is designated memory space within the CPU. After the fetch instruction cycle, the programme counter is incremented to point to the next instruction.

✞ Operations performed by CPU are:

- It executes instructions that carry out fundamental operations.
- Addition, Subtraction, Multiplication, Division are examples of arithmetic operations
- Memory operations are used to transfer data from one area to another.
- Logical operations set up a condition to test and then make a decision depending on the outcome.
- Control operations have an impact on other computer components
- These fundamental actions, when performed swiftly, enable a computer to perform a wide range of functions. The number of operations supported by a CPU is determined by its architecture.

✞ CPU and its Memory:

- The space where data and programmes are kept in a computer is referred as memory. Memory is not a component of CPU, yet it must interact closely with it in order to carry out operation. There are two kinds of memory: primary and secondary memory. CPU significantly relies on main memory to store programme instructions as well as the data that the instructions work on. Main memory is of a transient nature, and it only stores instructions and data for a programme while it is running. Hard drives and flash drives provide secondary memory, which is more permanent storage.
- The control unit, a component of PU, is in charge of shifting instructions and data from secondary storage into main memory prior to instruction execution. In addition, the control unit stores the outcomes of an instruction in secomdary stage

✞ Functions of a CPU in Computer:

- People commonly compare a computer's CPU to the human brain. This is a prime illustration because the CPU (central processing unit) is in charge of computer operation. It accomplishes this by executing computer algorithm instructions on data from various sources.
- The purpose of every computer is some form of data processing. The CPU aids data processing by performing fetch, decode, and execute functions on programmed instructions. These functions are often referred to as the instruction cycle when taken collectively. The CPU conducts data fetch and write functions in addition to the instruction cycle functions

✞ Functions of CPU related to Instruction Cycle:

- When a computer program executes, instructions are held in memory until they are executed. The CPU fetches the next instruction from memory using a program counter, which is stored in assembly code format. The instruction is decoded by the CPU into binary code that may be executed. After that, the CPU performs the operation, fetches or stores data, or adjusts the program counter to jump to a different instruction, as directed by the instruction.

✞ CPU Data Functions:

- The CPU may be instructed to execute an instruction that requires data while completing the execute function of the instruction cycle. Executing an arithmetic function, for example, demands the numbers that will be employed in the computation. There are instructions to retrieve data from memory and write data that has been processed back to memory in order to give the required data. The CPU's instructions and the data it works with are both stored in the same memory space. The CPU uses unique addresses to keep track of distinct memory regions.

✞ Micro-processors CPUs:

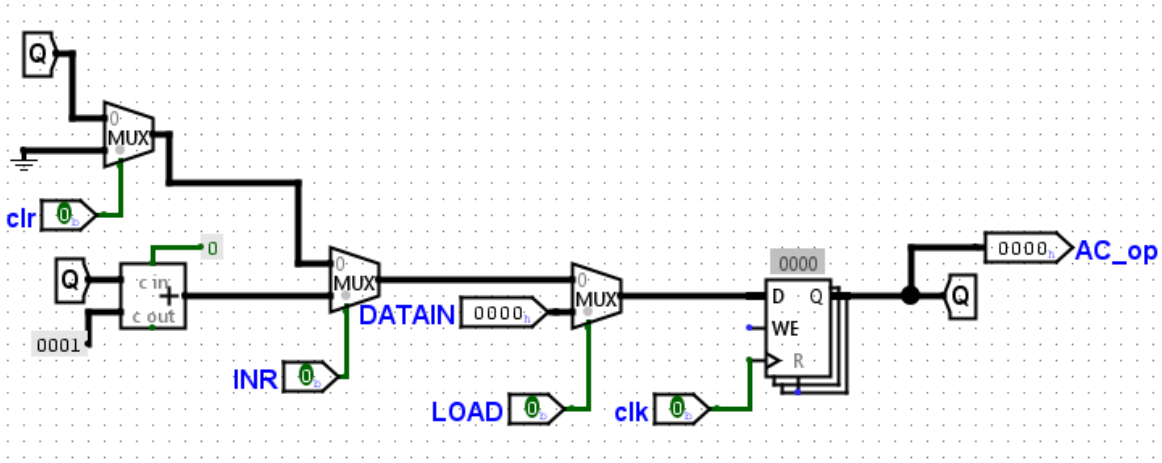
- A personal computer's microprocessor is a chip that includes all of the circuitry needed to control computer processes. It enables all CPU functions to be performed by a single chip, which is less expensive to produce and more reliable owing to the usage of integrated circuits. Prior to the development of microprocessors, a computer's CPU was located on a circuit board that included numerous chips coupled via integrated circuits. Many current processors include many CPUs on the same chip, referred to as cores.

● Circuit and its Sub-Circuits:

Registers:

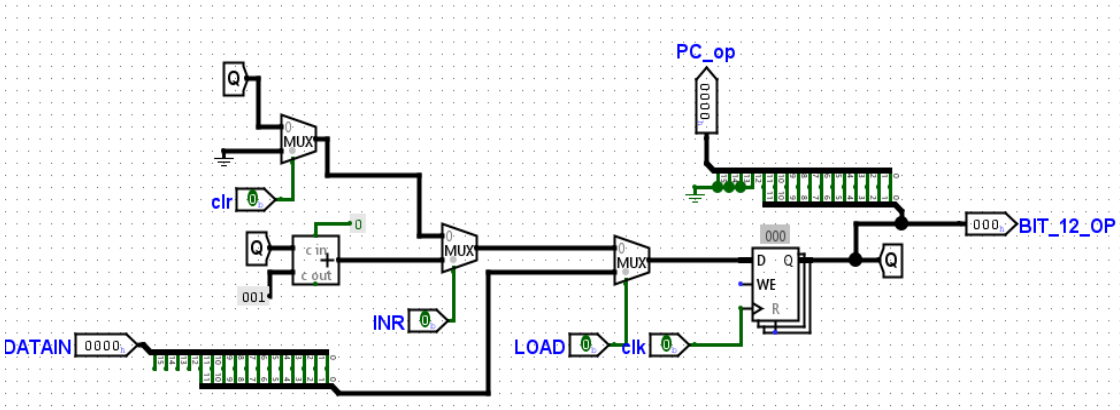
1. AR REGISTER:

AR(Address Register) : Contains the address of the memory location to be accessed, it is the only register connected for pointing the memory address in the RAM. This is a 12-Bit register as we have 12-bit address for memory location Address register is the CPU register that either stores the memory address from which data will be fetched to the CPU registers, or the address to which data will be sent and stored.



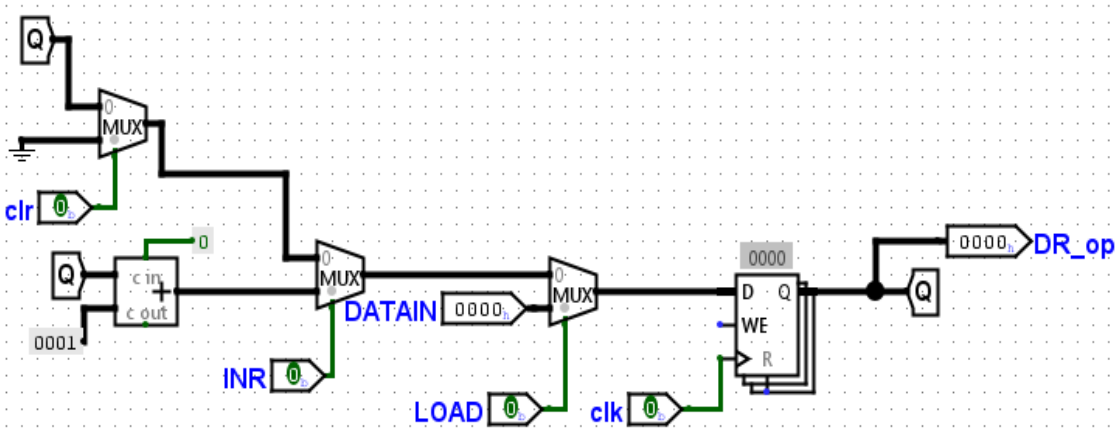
2. PC REGISTER:

PC(Program Counter) : The program counter, commonly called the instruction pointer in Intel x86 and Itanium microprocessors, and sometimes called the instruction address register, the instruction counter, or just part of the instruction sequencer, is a processor register that indicates where a computer is in its program sequence. Contains the address of the next instruction. This is also a 12-bit register as this also contains the address of the next instruction.



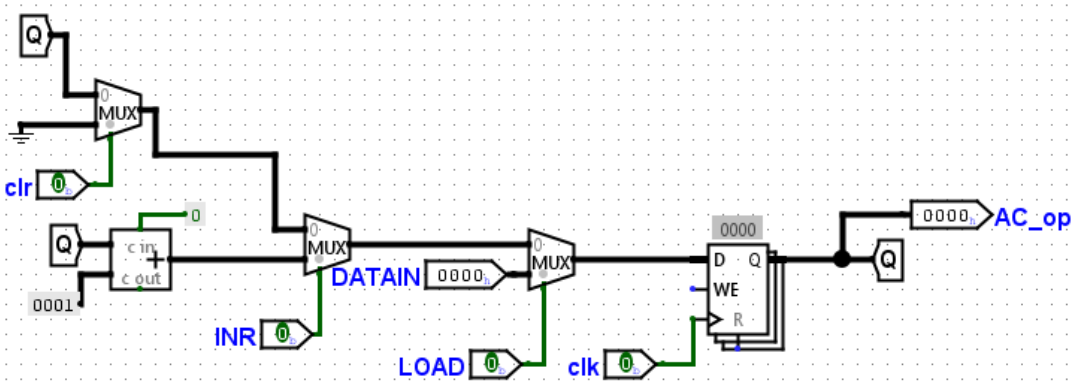
3. DR REGISTER:

DR(Direct Register) : Contains the data/operand for various operations to be performed on it, it is also connected to the ALU. This is a 16-bit register as the data size in this CPU is 16-bit. A memory buffer register or memory data register is the register in a computer's CPU that stores the data being transferred to and from the immediate access storage.



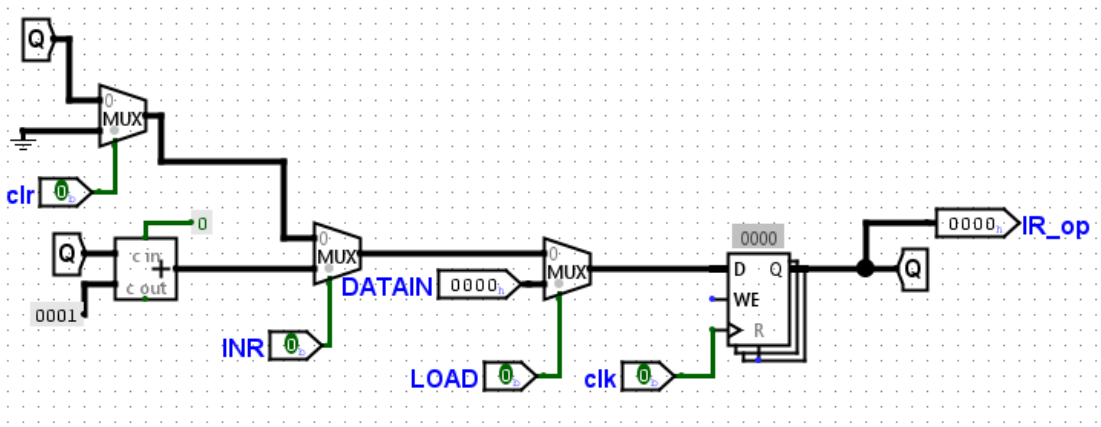
4. AC REGISTER:

AC(Accumulator) : This register is connected to the common bus through ALU, it is not directly connected to the common bus. All the results of the arithmetic operations are stored in this register. This is a 16-bit register as the data size in this CPU is 16-bit. In a computer's central processing unit, the accumulator is a register in which intermediate arithmetic logic unit results are stored.



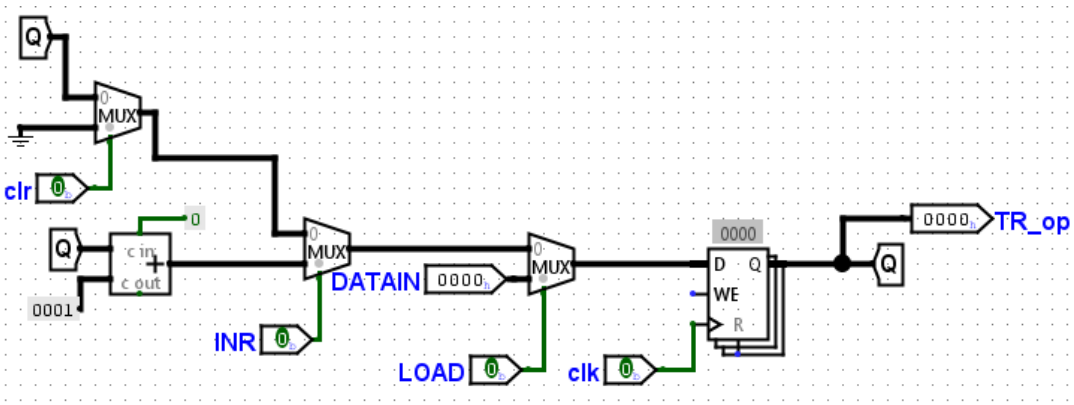
5. IR REGISTER:

IR(Instruction Register) : All the instructions are stored in this register to be decoded, this register is connected to the control and timing unit where the 4 MSBs are decoded. This is a 16-bit register as the data size in this CPU is 16-bit. The MSB is 'I' (Mode bit) the following 3 bits (14-12) are instruction bits followed by the remaining 12- bits (11-0) which are address bits.

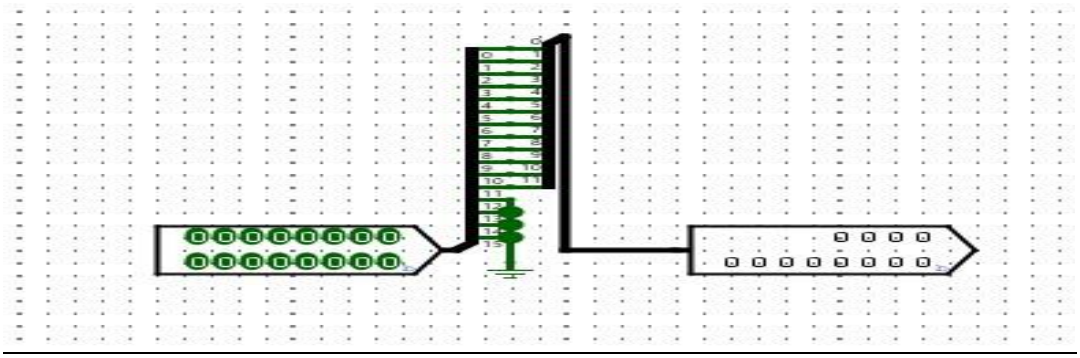


6. TR REGISTER:

TR(Temporary Register) : A temporary register is the only register that can be read and written more than once in a single instruction. Each temporary register has single-write and triple-read access. Therefore, an instruction can have as many as three temporary registers in its set of input source operands. This register is used to store temporary data like next program instruction during an interrupt cycle, in this project we aren't considering interrupt so this register isn't being used. This is a 16-bit register as the data size in this CPU is 16-bit. We have the same circuit for AC, DR,IR and Temporary Register.



7. 16 BIT TO 12 BIT CONVERTER:



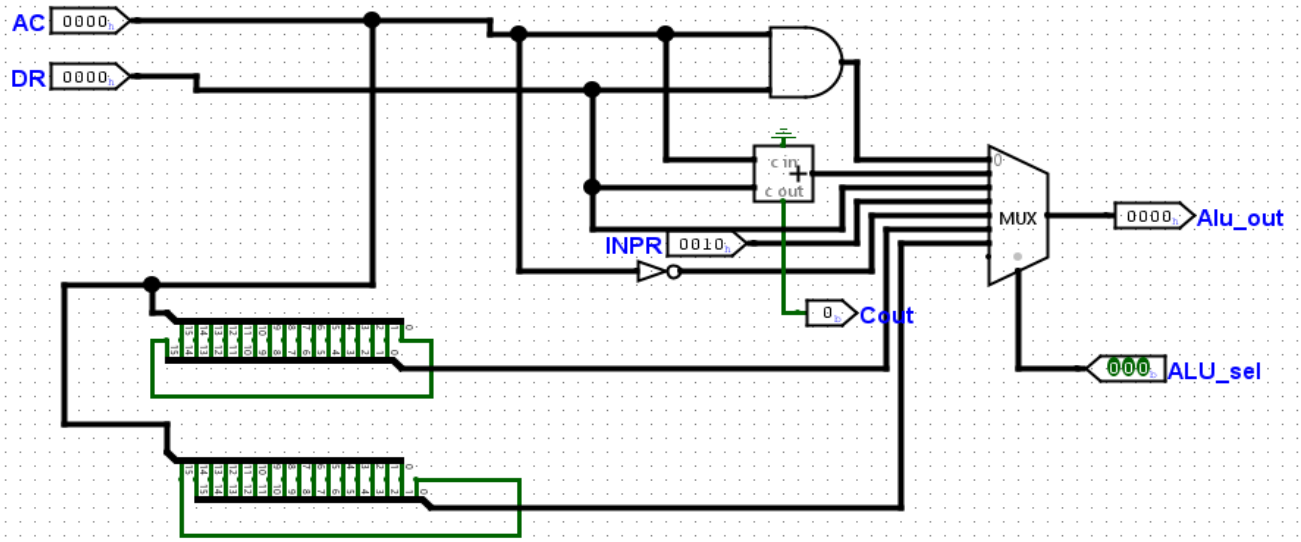
Here we have converted 16 bits to 12 bits using Splitter.

8. ARITHMETIC LOGIC UNIT (ALU):

The Arithmetic Logic Unit (ALU) is an essential component of a 16-bit CPU design. It performs arithmetic and logic operations on the data that is being processed by the CPU.The function of an ALU is to perform various arithmetic and logical operations on the data, such as addition, subtraction, multiplication, division, logical AND, logical OR, and bitwise operations, among others. The result of the operation is then stored in a register or memory location.

The ALU is responsible for carrying out these operations by receiving two inputs, performing the requested operation, and providing a single output. The inputs are usually stored in registers or memory locations, and the output is stored in a register or memory location as well.In a 16-bit CPU design, the ALU has to be able to handle 16-bit data, meaning it can perform arithmetic and logic operations on numbers up to 16 bits long. This is important because it allows the CPU to perform more complex operations than would be possible with an 8-bit ALU.

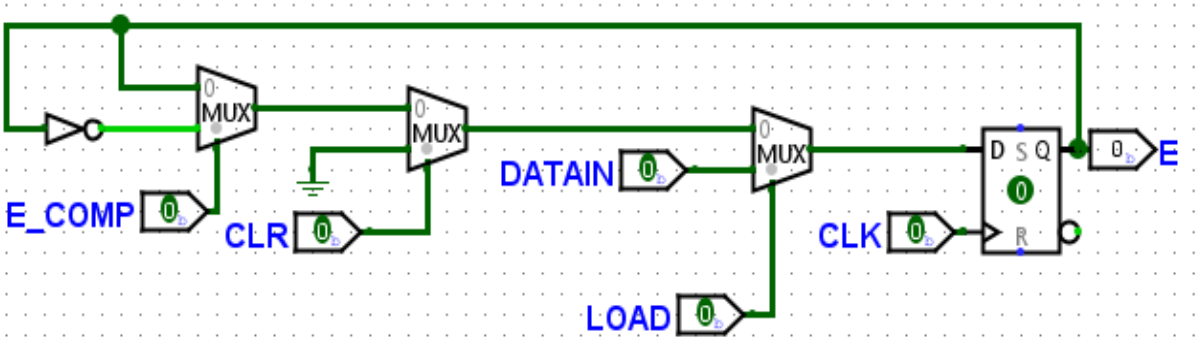
To summarize, the function of the ALU in a 16-bit CPU design is to perform arithmetic and logic operations on data, such as addition, subtraction, multiplication, division, logical AND, logical OR, and bitwise operations, among others. It can handle 16-bit data, allowing the CPU to perform more complex operations. we have combined arithmetic, logical and shift operations with the use of 8x1 mux



9. E-FF:

The E Register is for storing and performing operations on the carry bit that is generated by the addition operation in ALU.

It has control units Com_E (complement E), clr (clear), and load.



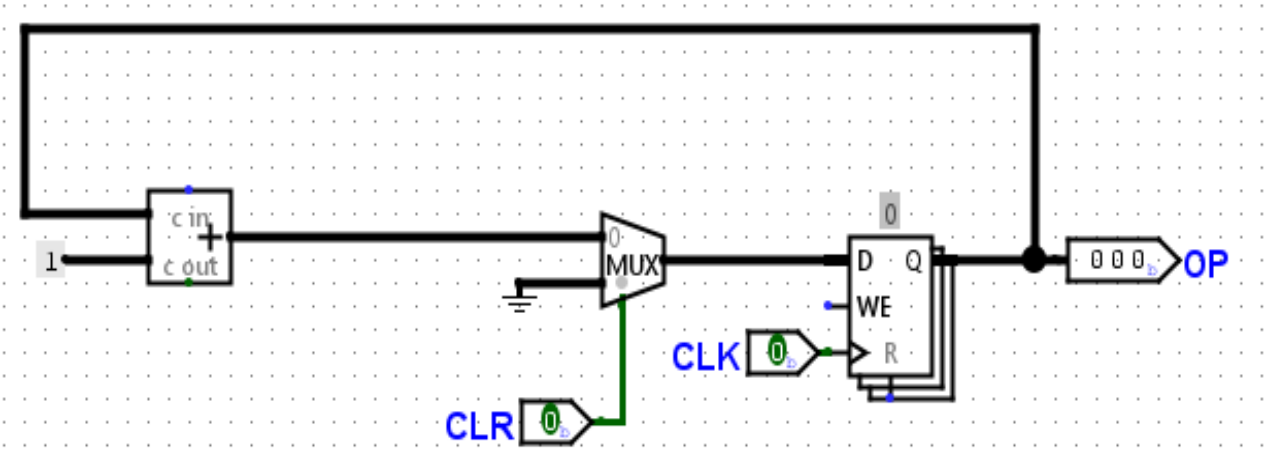
10. CONTROL SEQUENCE COUNTER:

In a 16-bit CPU design, the control-sequence counter is responsible for keeping track of the memory address of the next instruction to be fetched and executed.When a program is executed, the instructions are stored in memory, and the control-sequence counter points to the memory address of the first instruction. After the instruction is fetched and executed, the control-sequence counter is incremented to point to the next memory address that contains the next instruction. This process continues until the program is complete.

Here are the components commonly used to create a control sequence counter in Logisim using mux, decoder, flip-flop, and adder components:

1. **Clock:** A clock component generates a series of pulses that trigger the counter to count.
2. **Flip-flops:** Flip-flops are used to store the current count value. Depending on the number of bits required to represent the count value, multiple flip-flops can be used. For example, if a 4-bit count value is required, four D flip-flops can be used.
3. **Adder:** An adder component is used to add a constant value to the current count value. This is used to generate the next count value in the sequence. For example, if the counter needs to generate a sequence of odd numbers starting from 1, an adder can be used to add 2 to the current count value.
4. **Multiplexer:** A multiplexer component selects the appropriate output signal from multiple input signals based on the current count value. For example, a 4-to-1 multiplexer can select one of four input signals based on a 2-bit binary count value.
5. **Decoder:** A decoder component converts the binary count value into a corresponding output signal. For example, a 2-to-4 decoder can convert a 2-bit binary count value into a 4-bit output signal.
6. **Control logic:** The control logic manages the various inputs and outputs of the counter. This can be implemented using logic gates, such as AND, OR, and NOT gates.
7. **Input switches:** Input switches allow the user to manually set the initial count value, reset the counter, or load a new value into the counter.
8. **Output displays:** Output displays show the current count value or the output signal generated by the decoder.

These components can be combined and interconnected in various ways to create a control sequence counter that generates a sequence of output signals in a predetermined order based on the input signals it receives.

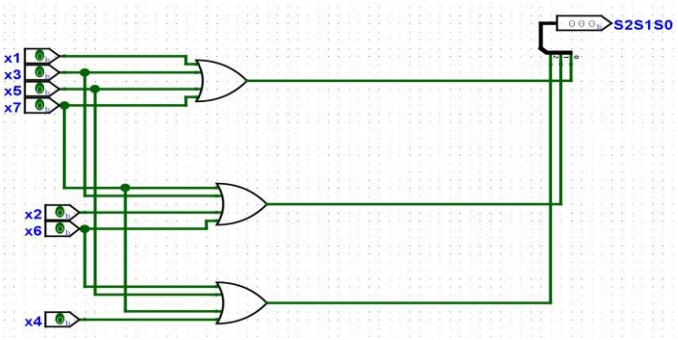


11. BUS SELECT GENERATOR:

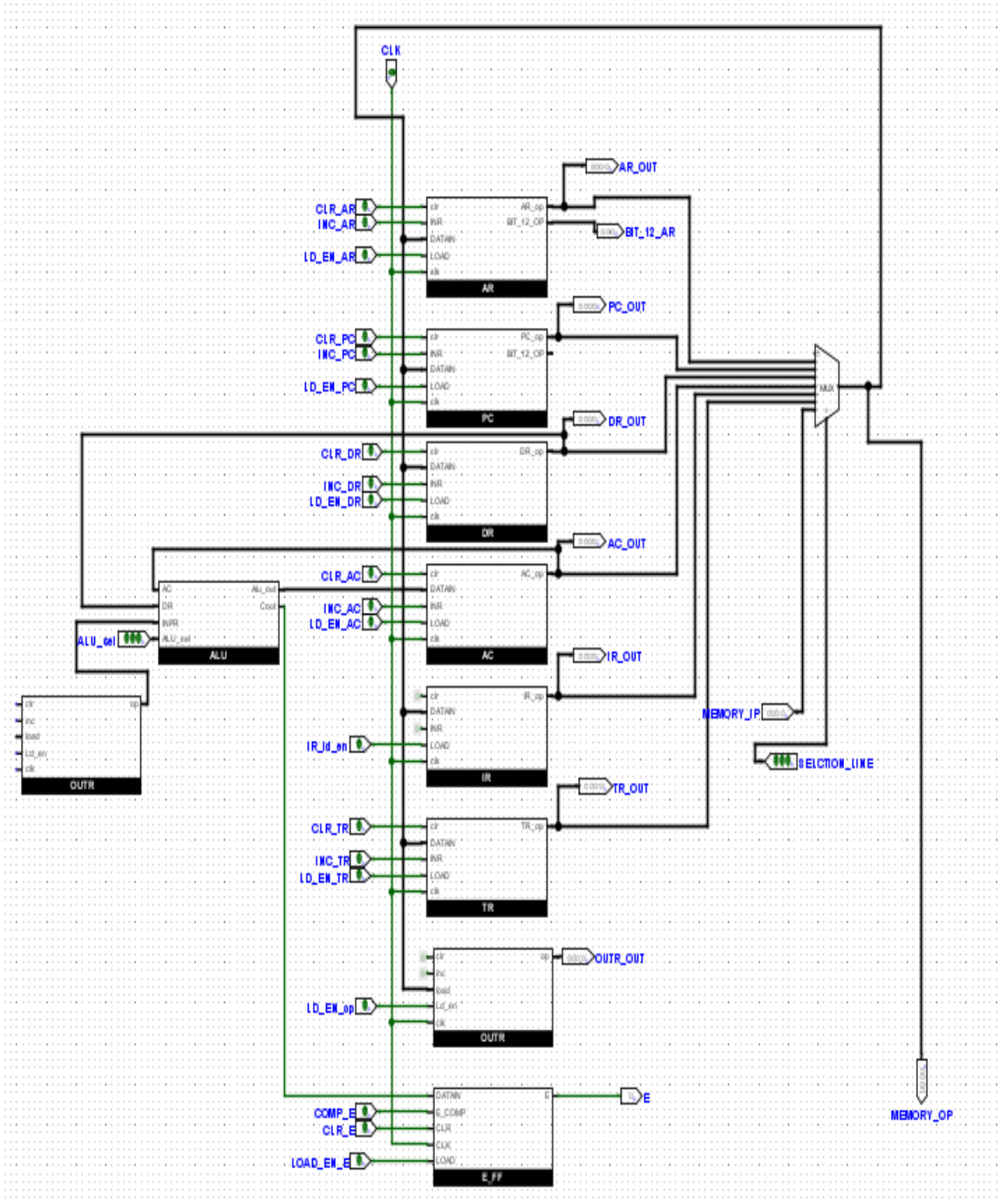
A 16-bit CPU typically requires a bus select generator to manage the routing of data and control signals between various components such as the CPU core, memory, input/output (I/O) devices, and other peripherals. Here's a simplified example of a bus select generator for a 16-bit CPU:

Inputs:
CPU address bus (16 bits): carries the memory address being accessed by the CPU.
CPU data bus (16 bits): carries the data being read from or written to memory or I/O devices.
Control signals (various): these include signals such as read/write (R/W), chip select (CS), and control signals for specific devices.

Operation:
The bus select generator receives the CPU address and data buses, as well as the control signals.
Based on the control signals, the bus select generator determines the type of operation (read or write), the target device (memory or I/O), and the specific address or port being accessed.
The bus select generator generates the appropriate control signals, such as R/W and CS, to enable the selected device to respond to the CPU's request. The bus select generator also routes the address and data on the appropriate buses, such as the memory address bus and memory data bus, to establish the correct communication path between the CPU and the selected device.


S2S1S0
x1
x3
x5
x7
x2
x6
x4

12.DATAPATH:


CLK
AR_OUT
CLR_AR
INC_AR
LD_EN_AR
BIT_12_AR
AR
PC_OUT
CLR_PC
INC_PC
LD_EN_PC
PC
DR_OUT
CLR_DR
INC_DR
LD_EN_DR
DR
AC_OUT
CLR_AC
INC_AC
LD_EN_AC
AC
ALU_sel
ALU
IR_OUT
MEMORY_IP
IR_ld_en
IR
SELCTION_LINE
TR_OUT
CLR_TR
INC_TR
LD_EN_TR
TR
OUTR_OUT
OUTR
LD_EN_op
COMP_E
CLR_E
LOAD_EN_E
E_FF
E
MEMORY_OP

A datapath is a collection of functional units such as arithmetic logic units (ALUs) or multipliers that perform data processing operations, registers, and buses.Along with the control unit it composes the central processing unit (CPU).A larger datapath can be made by joining more than one datapaths using multiplexers.

The general discipline for datapath design is to (1) determine the instruction classes and formats in the ISA, (2) design datapath components and interconnections for each instruction class or format, and (3) compose the datapath segments designed in Step 2) to yield a composite datapath.

Simple datapath components include *memory* (stores the current instruction), *PC* or program counter (stores the address of current instruction), and *ALU* (executes current instruction). The interconnection of these simple components to form a basic datapath

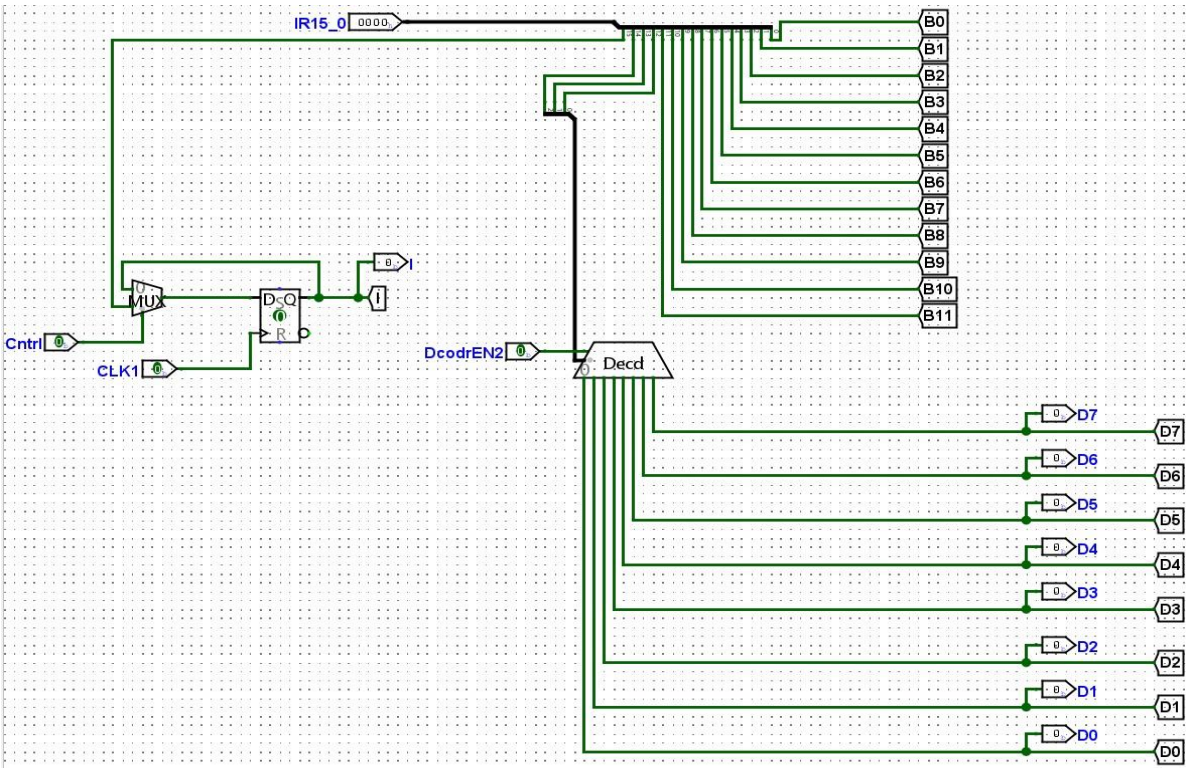
Such implementational concerns are reflected in the use of logic elements and clocking strategies. For example, with combinational elements such as adders, multiplexers, or shifters, outputs depend only on current inputs. However, sequential elements such as memory and registers contain state information, and their output thus depends on their inputs (data values and clock) as well as on the stored state. The clock determines the order of events within a gate, and defines when signals can be converted to data to be read or written to processor components (e.g., registers or memory).

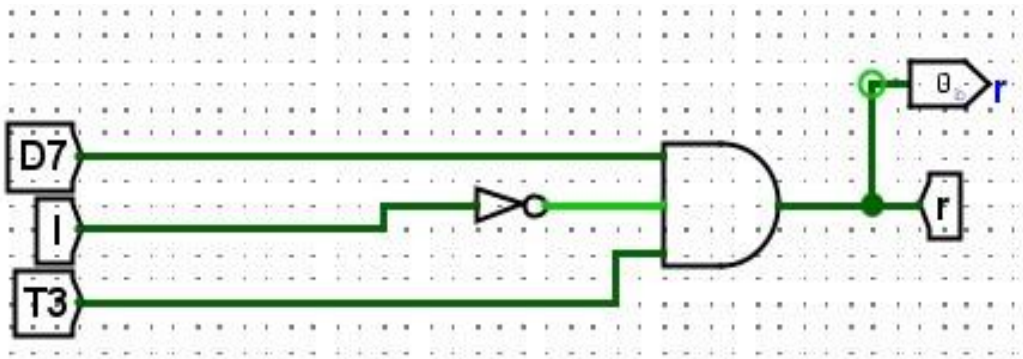
(9)Control Unit

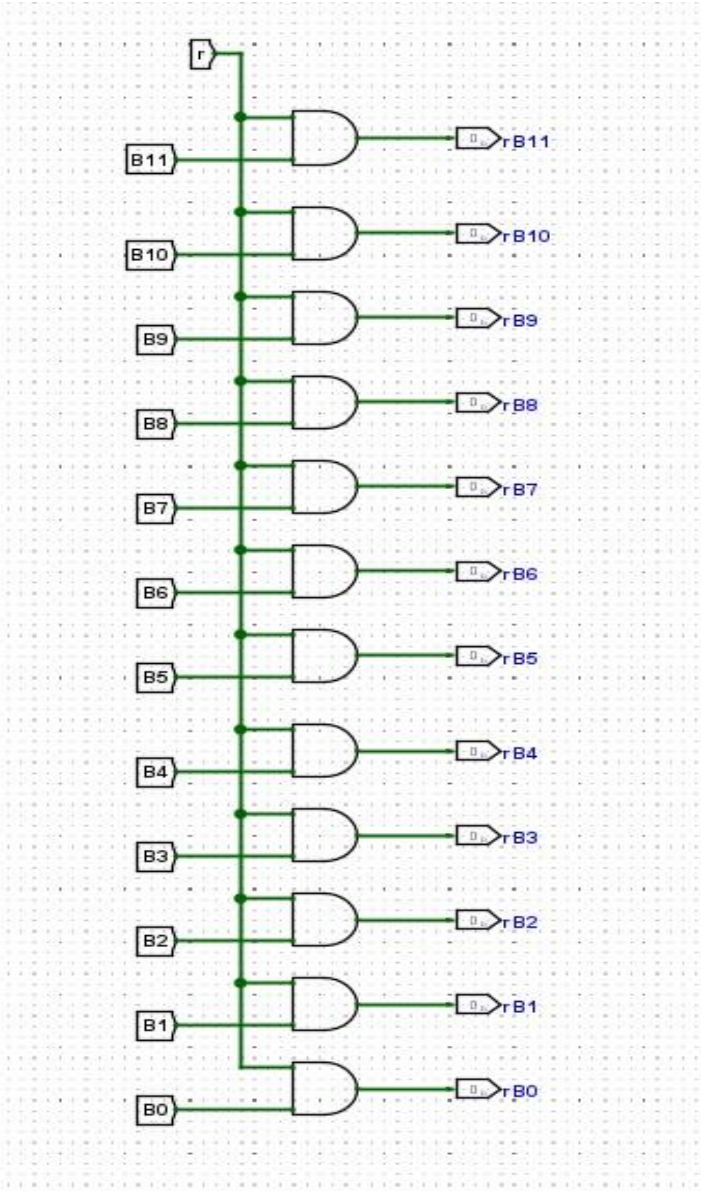
The control unit of a 16-bit CPU is responsible for managing the execution of instructions and coordinating the operations of various components within the CPU. It interprets instructions fetched from memory, generates control signals, and manages the timing and sequencing of CPU operations. Here's a high-level overview of the typical components and functions of a control unit in a 16-bit CPU:

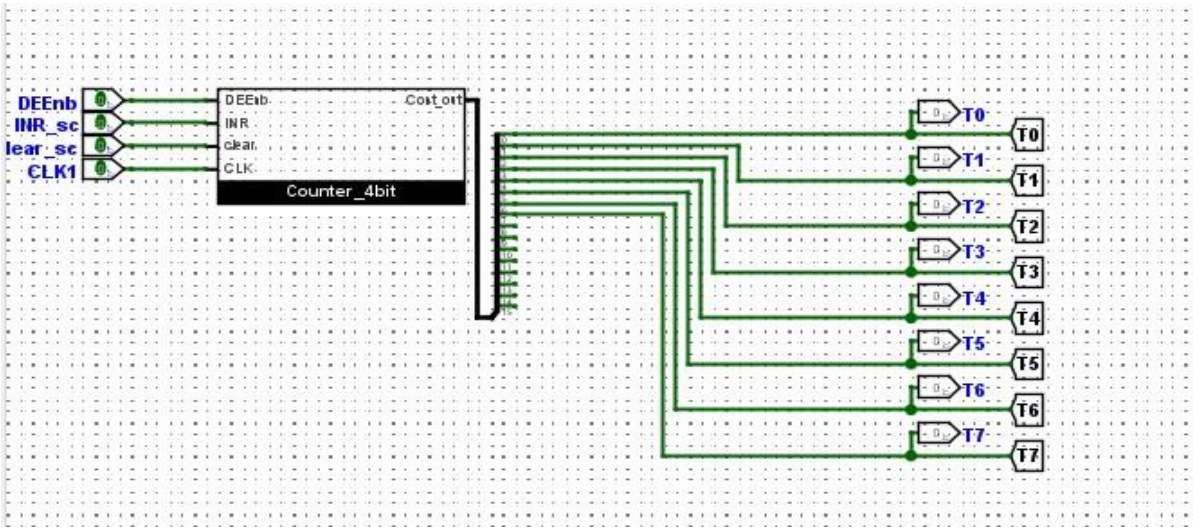
1. Instruction Decoder: Receives the instruction from the instruction fetch stage and decodes it to determine the type of operation to be performed. Determines the source and destination registers or memory addresses for data transfer and ALU operations. Generates control signals to control the operation of other CPU components, such as the ALU, register file, and data and address buses.
2. Control Logic: Generates control signals to coordinate the timing and sequencing of CPU operations. Controls the operation of various CPU components, such as the instruction fetch, instruction decodes, register file, ALU, and data and address buses.

3. Clock and Timing: Generates clock signals that synchronize the operation of the CPU components. Controls the timing of instruction fetch, instruction decode, execution, and other operations to ensure proper coordination of CPU activities.
4. Sequencer Counter: Controls the sequencing of instructions, fetching instructions from memory, and incrementing the program counter to fetch the next instruction. Coordinates the flow of instructions and data between the instructions fetch, instruction decode, and execution stages of the CPU pipeline.

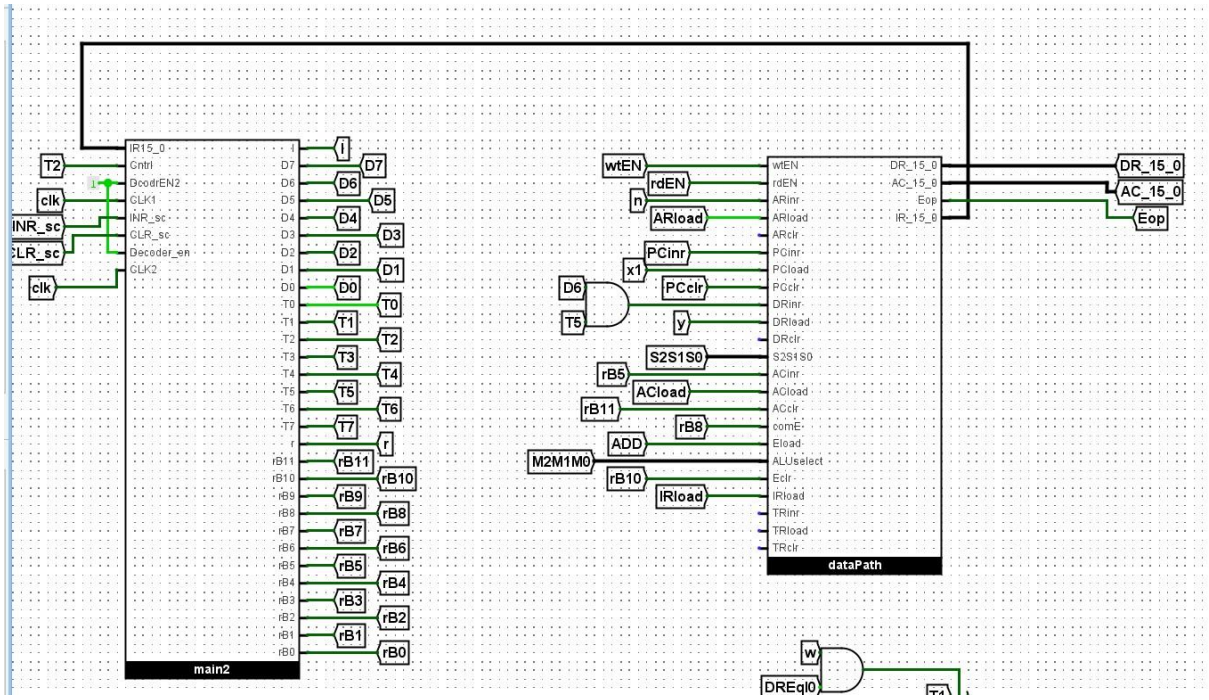


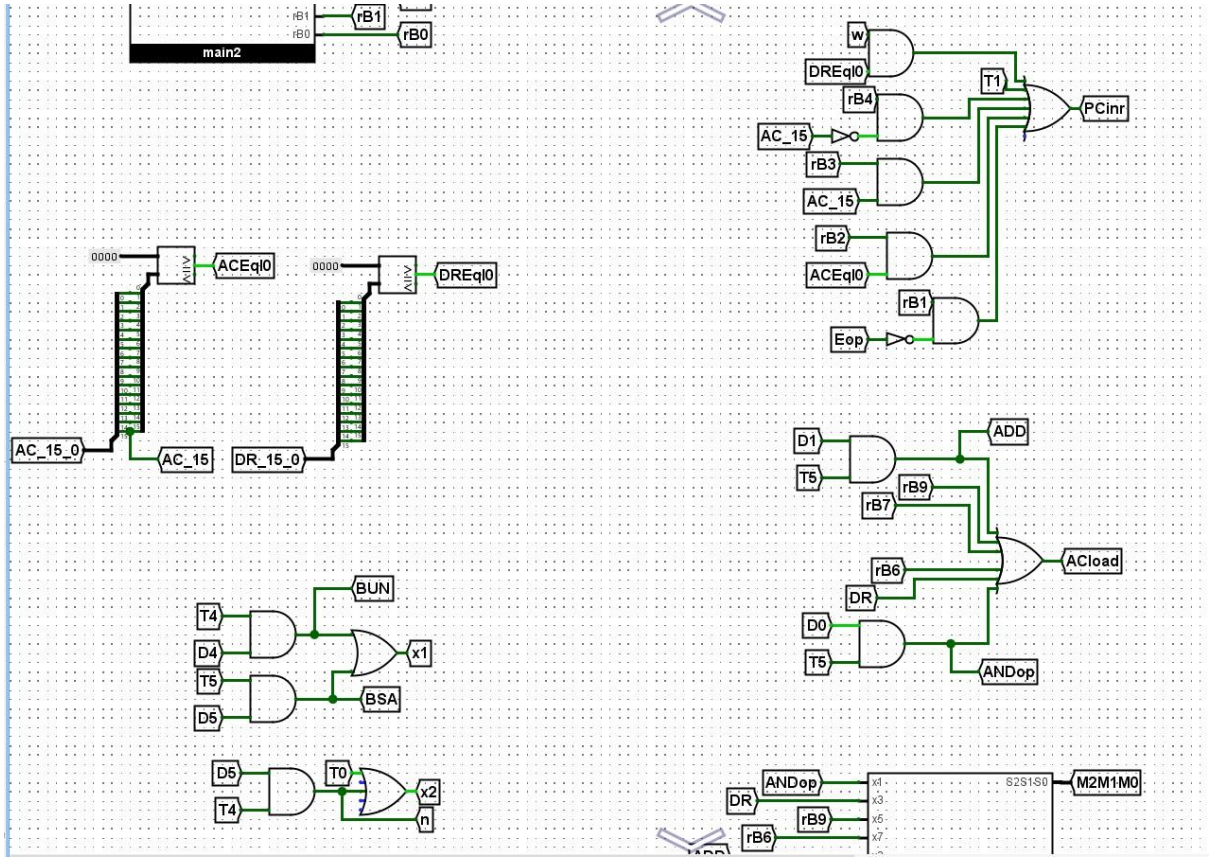


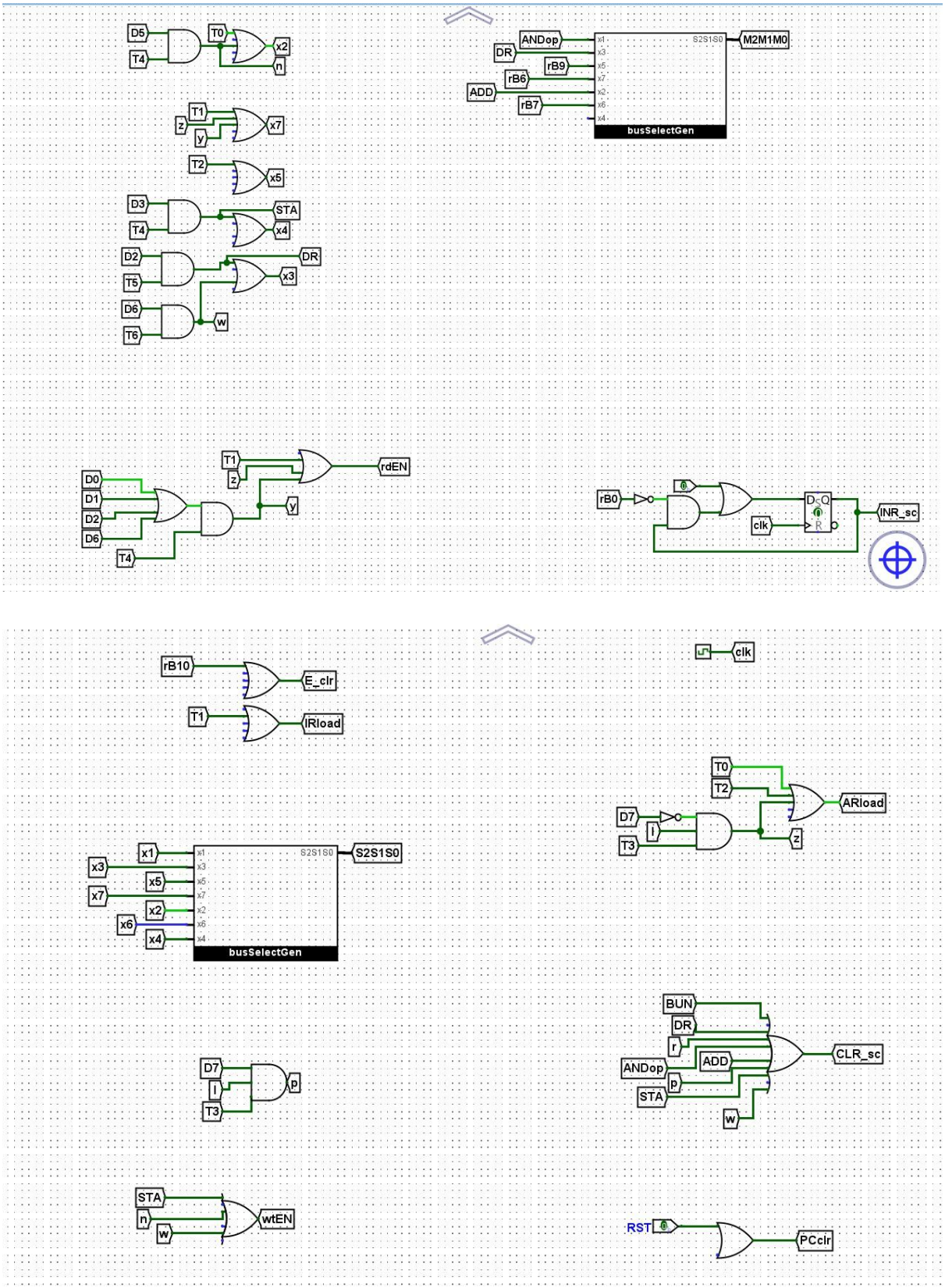

r
B11
B10
B9
B8
B7
B6
B5
B4
B3
B2
B1
B0
rB11
rB10
rB9
rB8
rB7
rB6
rB5
rB4
rB3
rB2
rB1
rB0


DEEnb
INR_sc
lear_sc
CLK1
DEEnb
INR
clear
CLK
Count_out
Counter_4bit
T0
T1
T2
T3
T4
T5
T6
T7

(10) Final(main) Circuit







Finally, we have connected data path, control unit and all memory reference instructions using tunnels.

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. The microoperations that execute this instruction is:

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 is 000. Two timing signals are needed to execute the instruction. The clock transition associated 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 C,., is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are
DIT,: DR <--- M[AR]
D1T5: AC <--- AC + DR, E <--- C,.,, SC <--- 0
The 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, SC <--- 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

TABLE 5-6 Control Functions and Microoperations for the Basic Computer

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$
Indirect	D_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:		
	$T_0'T_1'T_2'(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
Register-reference:		
		$D_7I'T_3 = r$ (common to all register-reference instructions)
		$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$

	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
Register-reference:		
		$D_7I'T_3 = r$ (common to all register-reference instructions)
		$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$

	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Input-output:		
		$D_7IT_3 = p$ (common to all input–output instructions)
		$IR(i) = B_i$ $(i = 6, 7, 8, 9, 10, 11)$
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

- **Result:**

This project provides the fundamental knowledge required to decipher the hardware digital processing unit. It explains how a computer's register transfer micro-operation works. It also describes the operation of the ALU, register, and multiplexer. As a result, it is possible to conclude that a 16-bit CPU was successfully designed.

- **Statement of skill development achieved:**

- Mastered using the simulation tool Logisim.
- Learned and understood the working of a simple hardwired control unit-based CPU.
- Learned to create and understand micro-operations.
- Learned to create control logics for various circuits.
- Learned about many sequential and combinational circuits during the project - Learned teamwork and coordination.
- Achieved time management and work management skills.

- Acquired knowledge of basic ALU.
- Gain an understanding in construction of CPU
- Learned how theoretical knowledge can be applied in the real world technology - Developed time management abilities.

- Learned how to construct basic register design used as the elementary computer unit - Learned implementation of registers and the micro-operations performed by them.
- Learnt the concept of series connection implementation of 12 and 16 bit load register in Logisim.
- Mastered time management and team synchronisation.

- Tasked with making logics for various registers and parts of CPU. - Gained knowledge on how to interpret instructions from instruction set - Implemented instructions in logisim simulator.
- Learned about control functions for several micro-operations. - Developed teamwork and coordination skills.

- Tasked with circuit design and developing Common Bus System.
- Learned how a tri-state buffer can prevent loading effects and its use In common bus.
- Learned how to make an efficient connection between various parts of the CPU.
- Applied theoretical knowledge of common bus in Logisim.

!! THANK YOU !!