National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Microprocessor Systems Assignment

4 Bit Micro-Controller

:

DEDICATION

This study is wholeheartedly dedicated to subject instructors Sir **Muhammad Imran Abeel** and **Mam Shaiza Malik**, who provided us with the necessary concepts, skills and are cooperative during duration of course.

Special thanks to our parents, who have been our source of inspiration and gave us the strength, who continually provided us with moral, spiritual, emotional and financial support.

To our brothers, sisters, relatives, mentors, friends and classmates who shared their words of advice and encouragement to finish this project.

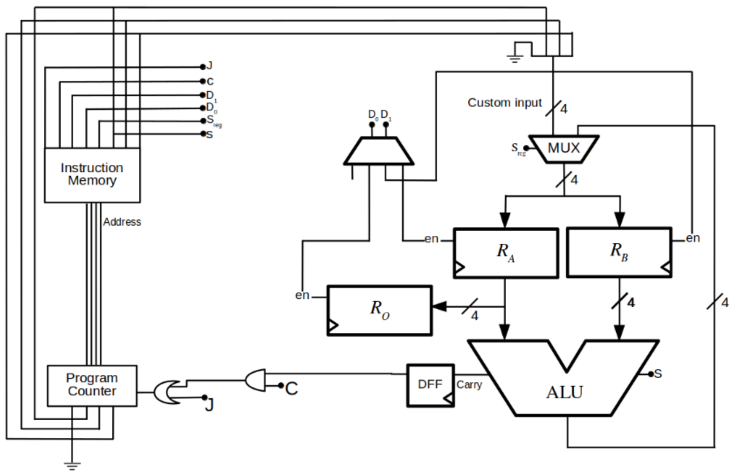
Lastly, I dedicate this work to the God Almighty against all odds, provided a place for my attachment and kept me safe and healthy throughout the period of my attachment.

CONTENTS

CHAPTER 1: INTRODUCTION

1.1 PROBLEM STATEMENT

Design 4-bit microprocessor using a simulator. There are 2 links given below, you can use as per your preference. The microprocessor should be able to demonstrate the flow of data and control signals. You should at least implement 5 instructions (MOV, ADD, JMP, IINC etc.) Understand the working of each component and its internals circuit that you use.



1.2 WHAT IS 4 BIT MICROCONTROLLER?

In general, a 4-bit microcontroller is a small computer on a chip that has a data width of 4 bits, meaning it can process and store data up to a maximum value of 15.

It typically includes a central processing unit (CPU), memory for storing data and instructions, input/output (I/O) ports for interfacing with external devices, and various other peripherals.

4-bit microcontrollers are often used in small-scale embedded systems and electronic projects due to their simplicity and low cost.

Due to their limited data width and processing capabilities, 4-bit microcontrollers are not suitable for complex applications and are generally replaced by more powerful microcontrollers or microprocessors in larger systems.

CHAPTER 2: ACHIEVED PROJECT

2.1 FEATURES

Our microcontroller is equipped to perform the following set of instructions:

- ✓ **loading** a value to a register
- ✓ **storing** data from a register to RAM
- ✓ **loading** data from RAM to a register
- ✓ **jumping** to a specified address in a program
- ✓ **arithmetic** operations such as adding and subtracting two registers, incrementing a register,
- ✓ **moving** value from one register to another,
- ✓ conditional jumps based on conditions, such as if the carry **flag** is one and if the zero flag is one.

2.2 TECHNICAL SPECIFICATIONS

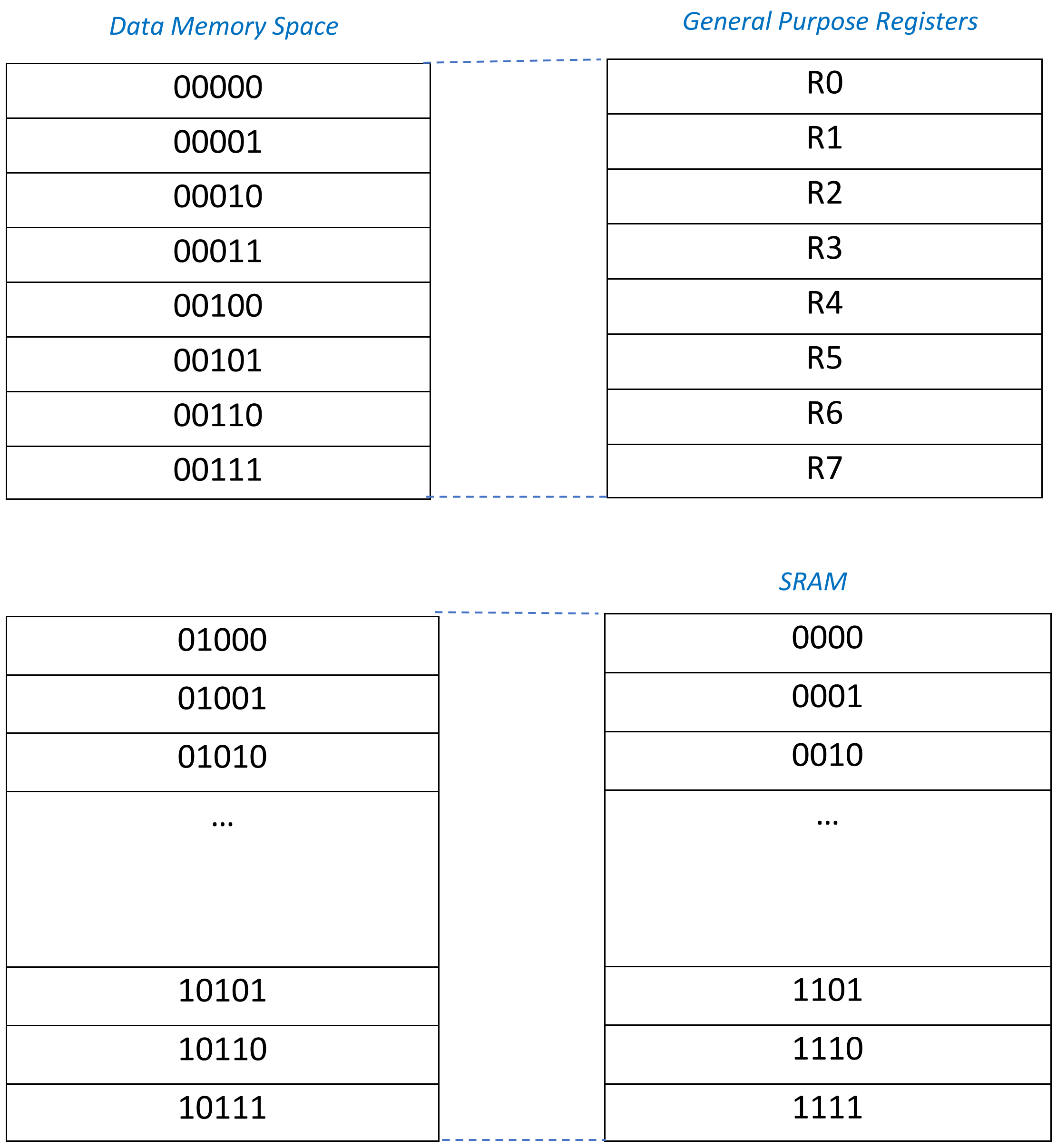
The architecture of our microcontroller follows the **Harvard machine model**, which has separate buses for data and instruction memory.

The microcontroller has a **4-bit** data bus and a **5-bit** address bus, allowing it to address up to 32 memory locations.

It has **8** general-purpose registers (GPRs) for storing data temporarily during the program execution. The microcontroller also includes a **4-bit** static random-access memory (SRAM) for data storage.

Additionally, it has **several flags** that are used to indicate certain conditions during program execution, such as carry, zero, overflow, negative, and sign. These flags are essential for implementing conditional jumps and performing arithmetic operations.

2.3 DATA MEMORY MAP



INSTRUCTIONS AND THEIR RESPECTIVE CODES

Instruction	*OPCODE*
LDI	0101
STS	0010
LDS	0001
MOV	0110
ADD	0011
SUB	0100
JMP	0111
BRCS	1000
BREQ	1001
OUT	1010

2.4 HOW TO USE THE INSTRUCTIONS?

All the addresses used must be from the Data memory Space.

1) **LDI** (Load Immediate)
   - ✓ Opcode Reg, Value
     0101 00111, 1111
   - ✓ The above instruction will load the value 1111 to R7(Memory address 00111)
2) **STS** (Store to Space)
   - ✓ Opcode Memory Address, Reg Address
     0010 10101, 00001
   - ✓ Stores the value of R1(Memory address 00001) to the SRAM location 1101(Memory address 10101)
3) **LDS** (Load Data Space)
   - ✓ Opcode Reg Address, Memory Address
     0001 00001, 10101
   - ✓ Load the value of SRAM location 1101(Memory address 10101) to R1(Memory address 00001)

4) **MOV**
   - ✓ Opcode Reg Address, Reg Address
     0110 00001, 00111
   - ✓ Copies the value of R1(Memory address 00001) to R7(Memory address 00111)

5) **ADD**
   - ✓ Opcode Reg Address, Reg Address
     0011 00001, 00111
   - ✓ Adds the contents of R7(Memory address 00111) to R1(Memory address 00001) and store it in R1.

6) **SUB**
   - ✓ Opcode Reg Address, Reg Address
     0100 00001, 00111
   - ✓ The above instruction would subtract the contents of R7(Memory address 00111) from R1(Memory address 00001) and store it in R1.

7) **JMP**
   - ✓ Opcode FLASH ROM address
     0111 0000 0001
   - ✓ Jumps to the second instruction (Location) of the Flash Rom.

8) **BRCS** (BRanch if Carry is Set)
   - ✓ Opcode FLASH ROM address
     1000 0000 0001
   - ✓ Jumps to the second instruction (Location) of the Flash Rom only if the carry flag is set.

9) **BREQ** (BRanch If Equal)
   - ✓ Opcode FLASH ROM address
     1001 0000 0001
   - ✓ Jumps to the second instruction (Location) of the Flash Rom only if the zero flag is set.

10) OUT

- ✓ Opcode Data memory Space address
  1010 10111
- ✓ Displays the contents of SRAM location 1111 on the 7-segment display.

2.5 SOFTWARE USED

Logisim 2.7.1 is an open-source software used for designing and simulating digital circuits. It provides a graphical interface for building circuits using logic gates and other components. It's popular in both educational and professional settings for circuit prototyping and testing.

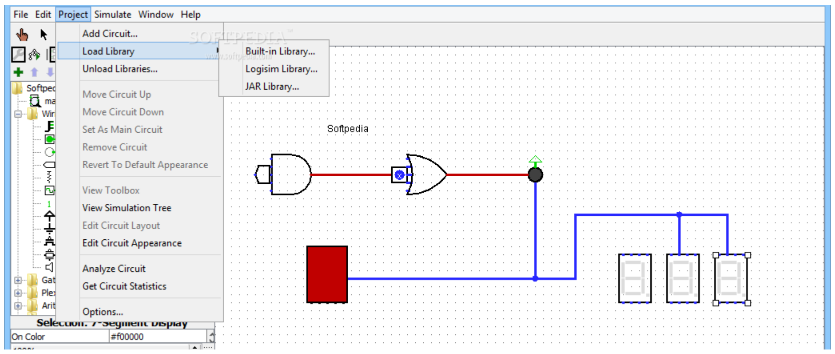
ADDING LIBRARIES:

When dividing a circuit design task into parts (collaboration), it is often useful to create and add libraries, in our case the .circ files pushed on individual github branches in Logisim. By doing so, individual team members can focus on creating specific components, which can then be shared with the rest of the team.

This approach saves time and effort while improving the overall quality of the design.

HOW TO ADD LIBRARIES:

Adding libraries in Logisim is a straightforward process that involves downloading the desired library file and importing it into the software.



In our case, download the **libraries.zip** and then unzip it.

Locate and **Logisim Library** under Project heading as shown in image, and add the downloaded libraries.

CHAPTER 3: THEORETICAL AND CONCEPTUAL EXPLANANTION

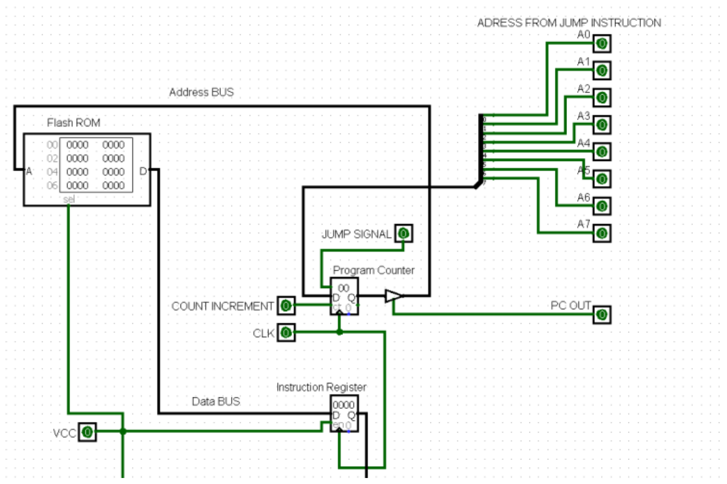
3.1 PROGRAM COUNTER

Why do we need PROGRAM COUNTER?

The job of a program counter is to keep track of the memory address of the next instruction to be executed, and to provide the processor with the necessary information to fetch and execute instructions from memory.

How is PC Implemented here?

The Program counter is connected to the Flash ROM via the Address bus (8-bit **Uni directional** bus), it generates the memory addresses of the instructions that are stored in the EEPROM.



3.2 FLASH ROM

Why do we need FLASH ROM?

The Flash ROM is a **non-volatile** memory that stores the program code. The program code is written to the flash PROM during the programming stage and cannot be modified during normal operation.

How is FLASH ROM implemented here?

The Program counter sends the memory address to the FLASH ROM, the ROM sends the contents stored at that specific address to the Instruction Register via the Data bus (**16-bit** unidirectional bus).

3.3 ARITHMETIC AND LOGIC UNIT and flag bits

Why do we need ALU?

The ALU is essential for executing programs and processing data, performing arithmetic and logic operations, and serving as the "**brain**" of the processor.

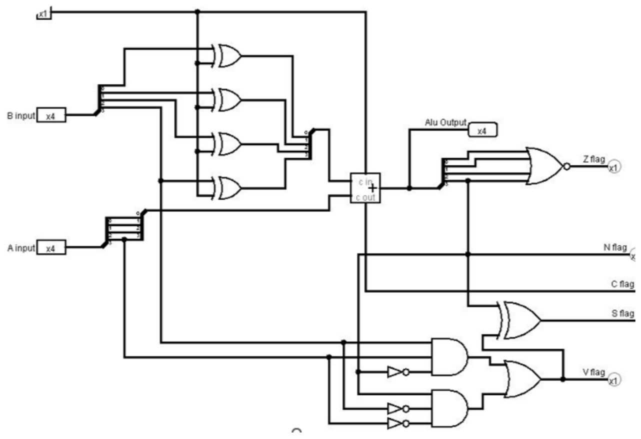
How ALU is implemented here?

The ALU is the part of our computer that performs **addition and subtraction** based on the provided control signal. In addition to performing addition and subtraction on the four input numbers, the ALU you described also has a unique feature where each bit of the second input number (B) is connected to an **XOR** gate with a control signal called **CTR**.

When CTR is set to 1, the XOR gate produces the one's complement of the bit. This means that each bit of B can be inverted when CTR is set, which can be useful for performing subtraction operations. If it is zero, then the B will not invert and will perform the addition.

Why we need FLAG BITS and How are they Implemented?

"Zero, Carry, Sign, and Negative flags"
are four essential flags to indicate the outcome of arithmetic and logical operations.



- The Zero flag (ZF) is set when the result of an operation is zero. This flag is useful for determining if a value or operation has produced an **output of zero**. The result is produced by connecting NOT to each bit of result and connected AND between them. If you convert it into a single gate, it will become **NOR.**
- The Carry flag (CF) is used to indicate if there was **a carry-out or borrow generated** during an arithmetic operation. The carry flag is also used in shift and rotate operations to hold the most significant bit that is shifted out.

- Overflow flag (OF)detects incorrect results due to limited bits in signed arithmetic. By this formula

V=S3.(~A3.~B3)+(~S3).(A3.B3)

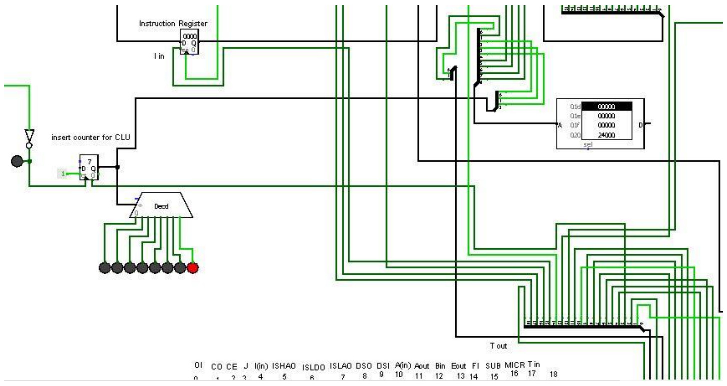
- The Sign flag (SF) is set when the result of an operation is negative. The sign bit of a binary number represents the sign of the number, and the sign flag indicates if the result of the operation is negative or positive. It is XOR of OVERFLOW flag and negative flag.

- The Negative flag (NF) is set when the result of an operation is negative. The negative flag is similar to the sign flag, but it is used in specific operations that involve signed numbers.IT is connected to last bit of operation.

3.4 CONTROL UNIT

Why do we need CLU?

The job of the CLU is to execute instructions fetched from memory. It does this by decoding the instructions (with the help of opcodes), generating control signals for the processor's components, and coordinating the transfer of data between the components. The control unit that each instruction is executed correctly and in the correct order.



How is CLU implemented here?

The CLU consists of an EPROM that receives the opcode of the instruction to be executed via the instruction register. Now the EPROM decodes this opcode with the help of the program that is stored in it and generates signals at the output that are necessary to execute that particular instruction.

3.5 GENERAL PURPOSE REGISTERS

Why do we need GPRs?

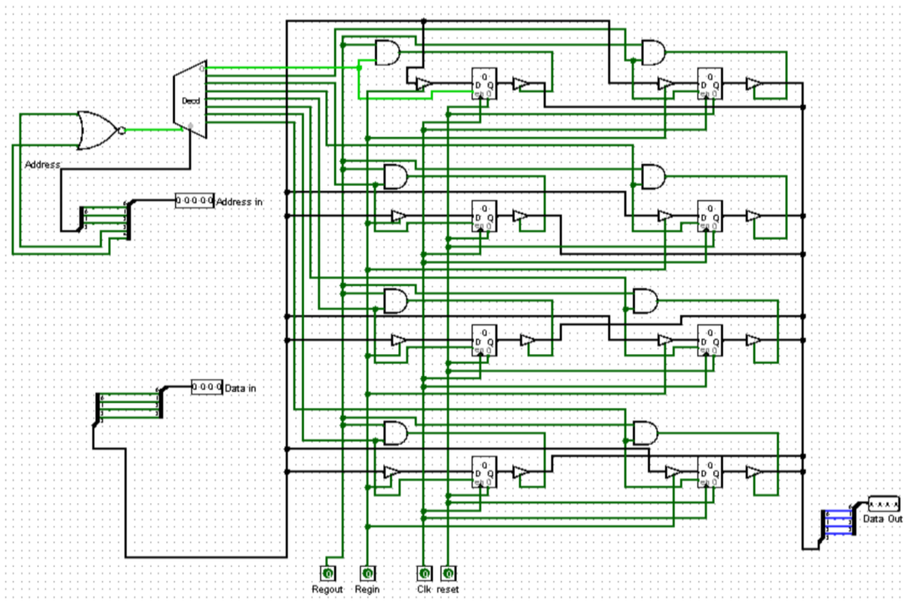
General Purpose Register are used for **temporarily** storing information.

The information can be of some value to be later used, address of the instruction to be jumped to and others depending on the instructions.

How GPRs are implemented here?

- ✓ In our application (4-bit microprocessor), **8 GPRs** have been created.
- ✓ All the GPRs are **4 bits wide**. To access a total of 8 registers, **3 bits** of address is required.
- ✓ The address bus for the data space of **4-bit** microcontroller is **5 bits.**

The lowest 8 of the possible addresses are used. The decoding circuitry of the GPRs is switched on only when there are one of the first 8 addresses on data space address bus. Control signals for the GPRs have been implemented for reading or writing to the GPRs (these are the shared control signals with SRAM).



3.6 SRAM

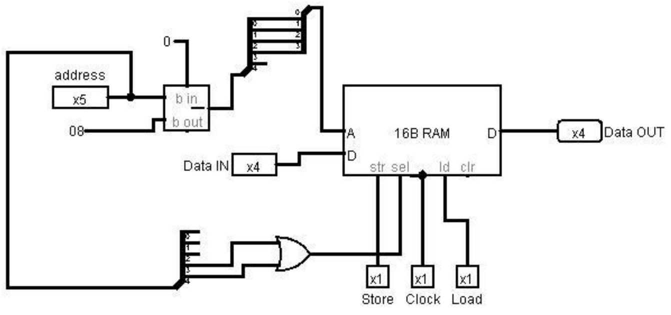
Why do we need SRAM?

Static Random Access Memory (SRAM) is a type of computer memory that is used for **temporary** storage of data and program code. Unlike other types of memory such as ROM or Flash memory, SRAM can be written and read **multiple** times, making it suitable for applications where data needs to be frequently accessed and updated.

SRAM is often used in microcontrollers and other embedded systems as it provides **fast** access to data and requires minimal power to operate.

4-bit SRAM is a type of static random-access memory that can store **4 bits** of data.

How is SRAM Implemented here?



- ✓ As discussed, before we have 8x GPRs, resulting in following address ranges

DATA SPACE	⟹	**00000 – 10100 (0–19)**
GPRs	⟹	**00000 – 00111 (0–7)**
SRAM	⟹	**01000 – 10100 (8–19)**

- ✓ From the incoming Address Bus, we need to subtract from **08** to get the Actual Ram Address.
- ✓ Most Two significant Bits of Address bus are used to Enable the SRAM.
- ✓ After subtracting, the most significant bit of the Address line becomes of no use, so we neglect that bit. In result we input 4 Bit Address line to SRAM.
- ✓ Pins **Store** (To get to RAM) and **Load** (To get from RAM) are controlled by the Control Unit.

CHAPTER 4: CONCLUSION

In conclusion, the 4-bit microcontroller project presented here provides insight into the architecture and capabilities of a simple yet powerful computing device.

This project is a valuable resource for those interested in the design and implementation of microcontrollers and embedded systems.

Additionally, the use of software tools such as Logisim 2.7.1 underscores their importance in the development and testing of such powerful electronic devices.