

8-Bit Processor

23.03.2024

IMT2022 - Analog Circuits Project

Ishan Jha

Sreyas Janamanchi

Harshavardhan R

Pradyumna G

Aryan Mishra

Nupur Patil

Overview

A processor is defined as the logic circuitry that responds to and processes the basic instructions that drive a computer or any other electronic device. Processors are considered to be one of the most, if not the most, vital and essential integrated circuitry of the device. It is responsible for executing fundamental operations from loading/storing data in the memory, performing basic arithmetic or other kinds of operations, graphics rendering, etc.

Most modern processors employ sizes that are a power of two, for example 8, 16, 32 or 64 bits. We plan on implementing a processor in which the size of each instruction is 8 bits and can execute a finite set of operations. Any instruction that would be entered by the user would be first converted into low level machine language (1s and 0s) and this machine language instruction would be executed by the processors by scanning the operation code(opcode), the registers to be used, and the memory locations involved in order to execute that particular instruction.

Inspiration

Since the beginning of the Industrial Revolution, which started in 1760 in the United Kingdom, technology has taken various forms and has evolved as time progressed, carrying out desired tasks of the human race, faster and more efficiently. From the invention of the printing press by Johannes Gutenberg, which was able to print hundreds to thousands of copies of books in a single day, to the invention of the first computer by Charles Babbage in 1822 in order to solve large and difficult arithmetic operations and combinatorics for encoding and decoding messages, technology solved many problems of the human world quickly and more efficiently. Fast forward to the 21-century and now we have quantum computers which can, depending on the number of qubits, perform 10 billion operations a second. Processors (and in turn semiconductors) have played a vital role in increasing the number of operations a computer can perform. The significance of processors in modern digital systems and its relevance with the other courses in colleges is our main inspiration to make this project.

Specifications

Structure of instructions

The processor is going to be an 8-bit processor, which means that each instruction that is going to be passed to the processor is going to be an array of 1s and 0s of size 8. These 8-bits are further subdivided as follows to give us more information about the instruction :-

1	2	3	4	5	6	7	8
1	0	1	0	0	0	1	1

As shown above, we have an 8-bit instruction 10100011 which is divided into three parts:

1. **Operation Code** : The first 3 bits of the instruction, numbered **1 to 3** (both inclusive), represent the operation which we wish to perform. Those 3 bits are presented using a **light green colour**. Each operation has a unique operation code and since 3 bits are used to represent the operation code, there are a total of 8 operations which our processor can perform. In the above instruction, the operation with an op-code of **"101"** would be executed by the processor.
2. **Register Number** : The **4th** bit of the instruction marked in the **light blue colour**, represents the number of the register which is going to be the "Principal Register" (this term is explained later in the report) for that particular operation. Since only one bit is being used for this purpose, our processor has only 2 registers. In the above instruction, the register numbered **'0'** is being used, and there are only two registers numbered '0' and '1' available with the processor.
3. **Memory Location** : The last 4 bits of the instruction, numbered **5 to 8** (both inclusive) and marked in the **light pink colour**, are used to represent the memory location which is going to be used in the operation. Since 4 bits are being used to represent the memory location, it implies that there are going to be a total of 16 memory locations, with each location being able to save 8-bits long of data. In our project, these memory locations are going to be numbered from 0 to 15. In the

above instruction, the memory location **"0011"** is being accessed, which implies the memory location number 3 is being used.

Operation Set

As stated in the previous section, a total of 3-bits are being used in each instruction to represent which operation the processor is to perform. Each operation has its own unique number which is assigned to it, this unique number is called the "Operation Code" (which shall be referred to as op-code henceforth).

Since 3-bits are being used to represent each unique operation, there are a total of 8 unique op-codes available, which implies that our processor will be able to execute a total of 8 unique operations. Depending on the value of the op-code, the operation corresponding to that op-code will be executed. The list of operations and their op-codes are as follows :-

Operation Executed	Operation Code (Op-Code)
Addition	000
Subtraction	001
Load Immediate	010
Load Word	011
Store Word	100
Conditional Jump	101

1. **Load Memory** : This operation is used to load/pick any data from any memory location (numbered 0 to 15) and load it into any of the two registers (numbered 0 or 1), which is to be called the "Principal Register", depending on the user. For example take the following instruction:

011-1-0101

The first three bits represent the op-code for load memory operation(**011**). The fourth bit is the register bit, and its value is '1' and the last four bits, memory address bits, are '0101'. This instruction is used to load the contents of memory address number 5 (**0101**) into register number 1(**1**).

2. **Store Memory** : This operation is used to store the contents of any of the two registers (numbered 0 or 1), which is to be called the “Principal Register”, and store it into any memory location(numbered 0 to 15) depending on the user. For example take the following instruction:

001-0-1100

The first three bits represent the op-code for store memory operation(**001**). The fourth bit is the register bit, and its value is ‘0’ and the last four bits, memory address bits, are ‘1100’. This instruction is used to store the contents of register number 0(**0**) into the memory address number 12 (**1100**).

3. **Addition** : This operation is used to add the contents of the principal register to the contents of the other register and then store their result in the desired memory location. For example tak the following instruction:

000-1-0110

The first three bits represent the op-code for addition operation(**000**). The fourth bit is the principal register bit, and its value is ‘1’ and the last four bits, memory address bits, are ‘0110’. This instruction is used to add the contents of the other register (in this case register number 0) to the contents of the **principal register** (register given in the instruction, register number 1(**1**)) and then store the result in the memory address numbered 6 (**0110**)

4. **Subtraction** : This operation is used to subtract the contents of the secondary register from the contents of the principal register mentioned in the instruction and then store the result in the desired memory location. For example take the following instruction:

001-1-1111

The first three bits represent the op-code for the subtraction operation(**001**). The fourth bit is the principal register bit, and its value is ‘1’ and the last four bits, memory address bits, are ‘1111’. This instruction is used to subtract the contents of the other register (in this case register number 0) from the contents of the **principal register** (register given in the instruction, register number 1(**1**)) and then store the result in the memory address numbered 15 (**1111**)

5. **Load Immediate** : The first three bits represent the op-code for load immediate operation (**010**). The fourth bit is the principal register bit, and the last four bits are

the value which is to be stored in the given register. This instruction is used to load a value directly into the **principal register** (keep in mind this value can only be of length 4 bits).

010-1-0110

The first three bits represent the op-code for the load immediate operation (**010**). The fourth bit is the principal register bit, and its value is '1' and the last four bits, are used to represent the value which is to be stored in the register mentioned in the instruction. This instruction is used to load a specific value, in this case 6 (**0110**) subtract the contents of the other register (in this case register number 0) from the contents of the **principal register** (register given in the instruction, register number 1(**1**)) and then store the result in the memory address numbered 15 (**1111**)

Arithmetic Logical Unit (ALU)

The arithmetic logical unit is that part of the processor which performs all the arithmetic operations. It receives a signal, signifying which operation is to be performed, after the decoding of the 8-bit instruction, and then performs that particular operation using the "Principal Register" (and in some operations it also uses the "Secondary Register"). Since only one bit is used for indicating the register number, there are only 2 registers which are available for use in the processor and they are numbered 0 and 1. A register could be one of the following types:

1. **Principal Register** : The register number which is mentioned in the 8-bit instruction (the fourth bit of the 8-bit instruction represents the register number) is known as the "Principal Register".
2. **Secondary Register** : The other register, which is not mentioned in the 8-bit instruction is known as the "Secondary Register".

In the case of arithmetic operations where the order of operands does matter, that is, the operations which are not commutative (such as division and subtraction), the contents of the "Principal Register" are taken as the first operand (in case of subtraction it is the positive term, and in case of division it is the dividend) while the contents of the "Secondary Register" are taken as the second operand (in case of subtraction, the number after the minus(-) symbol, and in case of division it is the divisor).

In order to understand the above point let's take the help of an example. Let's say we load the value 7 into register number '0' and value 2 into register number '1'. Now we want to perform division. If we give the processor the instruction :

101-0-0001

It would assign register number '0' as the **Principal Register** and register number '1' as the **Secondary Register**. Now in division operation, the principal register acts as the dividend and secondary register acts as the divisor. So the value 3 ($7/2 = 3.5$, quotient is 3) is stored in the memory address numbered 1 (**0001**). However if we were to switch the principal register by providing the following instruction to the processor:

101-1-0001

Now the registers would switch roles, that is, register '1' is now the Principal Register and register '0' is the Secondary Register. So now the value 0 ($2/7 = 0.285$, quotient is 0) is stored in the memory address numbered 1 (**0001**).

Clearly it is evident that the value being stored in the memory address numbered 1, depends on the fact which register is set as the Principal register, as in the example when register number '0' was the principal register, the value being stored was 3, but when register number '1' was the principal register, the value being stored was 0.

In the case of arithmetic operations where the order of operands does not matter (commutative operations), it doesn't matter which register is set as the Principal Register or the Secondary Register. In order to understand this, let's take the help of an example similar to the one we took to explain our point for non-commutative operations.


Let's say we load the value 7 into register number '0' and value 2 into register number '1'. Now we want to perform addition. If we give the processor the instruction :

010-0-0001

It would assign register number '0' as the **Principal Register** and register number '1' as the **Secondary Register**. Now in addition operation, the principal register acts as the first operand and secondary register acts as the second operand. So the value 9 ($7+2 = 9$) is stored in the memory address numbered 1 (**0001**). Now even if we were to switch the principal register by providing the following instruction to the processor:

010-1-0001

The registers would switch roles, that is, register '1' is now the Principal Register and register '0' is the Secondary Register but the value being stored in memory address numbered 1 (**0001**) would remain the same, that is, 9 ($2+7 = 9$).



This example clearly shows that in the case of commutative operations, it doesn't matter which register is the Principal Register or the Secondary register, as the result that is going to be stored in the destination memory address is going to be the same either way because of the commutative nature of the operation.

Memory Structure

In the 8-bit instruction that is given to the processor, the last 4 bits are used to signify the memory address that is going to be involved in execution of that particular instruction. Since each memory location has a unique address and 4 bits are used for the address, that gives a total of 16 memory locations. In our project we shall be numbering these memory locations from 0 (0000) to 15 (1111) and they shall use flip-flops to store data.

A raspberry pi microprocessor will run a python code and will convert assembly language into low level machine language of 1s and 0s, and these low level machine code will be stored in an "Instruction Memory".

We also intend to display the data/value that is stored in any one of the memory locations as per the user's request while the processor is running. If time permits, we also intend on displaying this data with the help of Seven Segment Display.

Tentative List of Components

- Wires
- Breadboard
- Resistors
- Capacitors
- Bitwise Operation Gates IC
- Flip Flops
- 555 Timer Integrated Circuit
- Coders/Multiplexers and Decoders/Demultiplexers
- LEDs(various colours possibly)
- 4-bit D-type registers (74LS173)
- Static RAM (SRAM - 74LS219 / 74LS189)
- Binary Counter (74LS161A)
- 4-bit full adder (74HC83)

- EEPROM (AT28C16)
- Octal Bus Transceiver (74LS245)

Prices of Components & Links:

Serial Number	Component	Link	Price(per unit in Rs)
1	EEPROM	https://www.indiamart.com/proddetail/at28c16-21571525248.html	10
2	Octal Bus Transceiver (74LS245)	https://www.amazon.in/74LS245-74245-Octal-Bus-Transceiver/dp/B09NK4CW5L	49
3	Static RAM	https://sharvielectronics.com/product/74ls189-64-bit-ram-with-3-state-output-ic-dip-16-package/	139
4	4-bit D-type registers(74LS173)	https://www.amazon.in/74LS173-D-Type-Registers-3-State-Package-DIP-16/dp/B09HQQHYGQF	169
5	4-bit Synchronous Binary Counter	https://www.electroniccomp.com/74ls161-4-bit-synchronous-binary-counter-ic-74161-dip-14-package?gad_source=1&gclid=EALalQobChMI1O-Xxs-NhQMvFQp7Bx3	20

		LXAMeEAQYASABEgJ r8PD_BwE	
6	4-bit Full Adder	https://electroncart.in/74ls83-4-bit-binary-adder-ic/	199

Members of the Team

Serial Number	Name	Roll Number	Email ID
1	Ishan Jha	IMT2022562	Ishan.Jha@iiitb.ac.in
2	Sreyas Janamanchi	IMT2022554	Sreyas.Janamanchi@iiitb.ac.in
3	Harshavardhan R	IMT2022515	R.Harshavardhav@iiitb.ac.in
4	Pradyumna G	IMT2022555	Pradyumna.G@iiitb.ac.in
5	Nupur Patil	IMT2022520	Nupur.Patil@iiitb.ac.in
6	Aryan Mishra	IMT2022502	Aryan.Mishra@iiitb.ac.in