|  |  |
| --- | --- |
| CPU ile ilgili görsel sonucu  MiCROPROCESSORS  PROJECT | 16-BIT SINGLE CYCLE MIPS MICROPROCESSOR  Processors are regarded as one of the most important devices in our everyday machines called computers. Processor is an electronic circuit that functions as the central processing unit (CPU) of a computer, providing computational control. Processors are also used in other advanced electronic systems, such as computer printers, automobiles, and jet airliners, Calculators and etc.  GROUP PROJECT  COE 381 MICROPROCESSORS |

**KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**COLLEGE OF ENGINEERING**

**DEPARTMENT OF ELECTRICAL/ELECTRONICS ENGINEERING**

****

**DESIGN OF A 16-BIT SINGLE CYCLE MIPS MICROPROCESSOR**

The design team,

* Musah Ibrahim Ali 3555618

**ACKNOWLEDGEMENT**

We would like to thank the Almighty God for being our guide and protector throughout the project sessions. We would also express our gratitude to Dr. Jephthah Yankey for giving us this challenging yet intuitive and interesting project.

**INTRODUCTION**

Microprocessors & Microcontrollers are generally designed in the vicinity of two main computer architectures: Complex Instruction Set Computing i.e., CISC architecture and Reduced Instruction Set Computing i.e., RISC architecture. The concept of CISC is based on Instruction Set Architecture (ISA) design that redoubles performing further with several instructions utilizing changeable number of operands and an out spread variation of addressing modes in disparate locations in its Instruction Set. Thus, causing them to have varying execution time and lengths thereby authoritatively mandating an intricate Control Unit, which inhabits an immensely existent region on the chip. Compared with their CISC analogue, RISC processors typically support a minuscule set of instructions. A display that juxtaposes RISC processor with CISC processor, the number of instructions in a RISC Processor is low while the number of general-purpose registers, addressing modes, fixed instruction length and load-store architecture is more this in turn facilitates the execution of instructions to be carried out in a short time thus achieving higher overall performance

Currently, the efficacy of the RISC processors is generally accepted to be greater than that of their CISC counterparts. Before their execution the instructions are translated into RISC instructions in even the most popular CISC processors. The attributes mentioned above accentuate the design strength of RISC in the market for embedded systems known as "system-on-a-chip (SoC)". The premier microprocessors exhibiting reduced instruction set are SPARC, ARM, MIPS and IBM's PowerPC. RISC processor typically has load store architecture. This denotes there are two instructions for accessing memory which are a load instruction set to load data from the memory and store instruction set to Write Back (WB) the data into memory without any instructions.

Instruction Set Architecture (ISA) is an abstract model of a computer.

**THE MIPS MICROPROCESSOR**

The MIPS instruction set architecture (ISA) is a RISC based microprocessor architecture that was developed by MIPS Computer Systems Inc. in the early 1980s. MIPS is now an industry standard and the performance leader within the embedded industry. Their designs can be found in Canon digital cameras, Windows CE devices, Cisco Routers, Sony Play Station 2 game consoles, and many more products used in our everyday lives. By the late 1990s it was estimated that one in three of all RISC chips produced was a MIPS-based design. Architecture of MIPS RISC microprocessor includes, fix-length straightforward decoded instruction format, memory accesses limited to load and store instructions, hardwired control unit, a large general purpose register file, and all operations are done within the registers of the microprocessor.

# Format

## R-Type

|  |  |  |  |
| --- | --- | --- | --- |
| Op-Code  4 bit | rs  4 bit | rt  4 bit | rd  4 bit |

**I-Type**

|  |  |  |  |
| --- | --- | --- | --- |
| Op-Code  4 bit | rs  4 bit | rt  4 bit | Immediate  4 bit |

## J-Type

|  |  |
| --- | --- |
| Op-Code  4 bit | Target Address  12 bit |

In our Structure we have chosen 3 formats. If we deal with constant (like a = a + 4), we can’t deal with R-Type format. So, we need an immediate type format. For conditional statements, sometime we need to jump to a particular line or need to out of a loop. In that case, we need J-Type format.

# Registers

|  |  |  |  |
| --- | --- | --- | --- |
| **Register Number** | **Name of the**  **Registers** | **Usage** | **Value Assigned (4**  **bit)** |
| 0 | $zero | Hard-wired to 0  (constant value 0) | 0000 |
| 1 | $t0 | Temporary | 0001 |
| 2 | $t1 | Temporary | 0010 |
| 3 | $t2 | Temporary | 0011 |
| 4 | $t3 | Temporary | 0100 |
| 5 | $s0 | Save | 0101 |
| 6 | $s1 | Save | 0110 |
| 7 | $s2 | Save | 0111 |
| 8 | $s3 | Save | 1000 |
| 9 | $s4 | Save | 1001 |
| 10 | $s5 | Save | 1010 |
| 11 | $gp | Global Pointer | 1011 |
| 12 | $sp | Stack Pointer | 1100 |
| 13 | $fp | Frame Pointer | 1101 |
| 14 | $ra | Return Address | 1110 |
| 15 | $a0 | Arguments to  functions | 1111 |

**Operations**

## R-Type Table

|  |  |  |
| --- | --- | --- |
| **Instruction Type** | **Instruction** | **Op-Code** |
| Arithmetic | add | 0000 |
| sub | 0001 |
| Logical | and | 0010 |
| or | 0011 |
| nor | 0100 |
| xor | 1110 |

**I-Type Table**

|  |  |  |
| --- | --- | --- |
| **Instruction Type** | **Instruction** | **Op-Code** |
| Data Transfer | lw | 0101 |
| sw | 0110 |
| Conditional Branch | beq | 0111 |
| mul | 1000 |
| Arithmetic | addi | 1001 |
| Logical | andi | 1010 |
| ori | 1011 |
| sll | 1100 |
| srl | 1101 |

**J-Type Table**

|  |  |  |
| --- | --- | --- |
| **Instruction Type** | **Instruction** | **Op-Code** |
| Unconditional Jump | J | 1111 |

# Operations’ Instructions

## add:

It adds the content of the source register 1 to the contents of the source register 2 and saves it in the destination register.

Operation: $s0 = $s0 + $t0 Syntax: add $s0, $s0, $t0

## sub:

It subtracts the content of the source register 2 from the contents of the source register 1 and saves it in the destination register.

Operation: $s0 = $s0 - $t0 Syntax: sub $s0, $s0, $t0

## and:

It does a bit by bit logical and operation between two source registers contents. Operation: $s0 = $s0 & $t0

Syntax: and $s0, $s0, $t0

## or:

It does a bit by bit logical or operation between two source registers contents.

Operation: $s0 = $s0 || $t0 Syntax: or $s0, $s0, $t0

## nor:

It does a bit by bit logical “nor” operation between two source registers contents. Operation: $s0 = ~ ($s0 | $t0)

Syntax: nor $s0, $s0, $t0

## slt:

It compares the contents of two source registers. If the content of 1st source register is less than the second source register, then it sets the value of destination register to 1, otherwise 0

Operation: if ($s0 < $t0)

$s0 = 1 Else $s0 = 0

Syntax: slt $s0, $s0, $t0

## lw:

It loads the contents of the memory specified by the offset and saves it into a destination register. Operation: $s0 = Memory [$s1 + offset]

Syntax: lw $s0, 2($s1)

## sw:

It stores the contents of a source register to the memory address specified by the offset. Operation: Memory [$s1 + offset] = $s0

Syntax: sw $s0, 2($s1)

## beq:

It checks if the content of the provided register is equal to the contents of another register or not. If equal jumps a relative number of instructions else continue.

Operation: if ($s0==$t0) jump to L (Label)

Else go to next line Syntax: beq $s0, $t0, L

## bne:

It checks if the content of the provided register is not equal to the contents of another register or not. If not equal, jumps a relative number of instructions else continue.

Operation: if ($s0 != $t0) jump to L (Label)

Else go to next line Syntax: bne $s0, $t0, L

## addi:

It can add direct value (constant) with a content of a source register. Operation: $s0 = $t0 + immediate value

Syntax: addi $s0, $t0, 4

## andi:

It does logical bit by bit and operation with a constant value and a content of a source register. Operation: $s0 = $t0 & immediate value

Syntax: andi $s0, $t0, 4

## ori:

It does logical bit by bit or operation with a constant value and content of a source register. Operation: $s0 = $t0 || immediate value

Syntax: ori $s0, $t0,

## srl:

This operation shifts the content of a register to right by a constant and stores shifted value to destination register.

Operation: $s0 = $t0 >> 4 Syntax: srl $s0, $t0, 4

## sll:

This operation shifts the content of a register to left by a constant and stores shifted value to destination register.

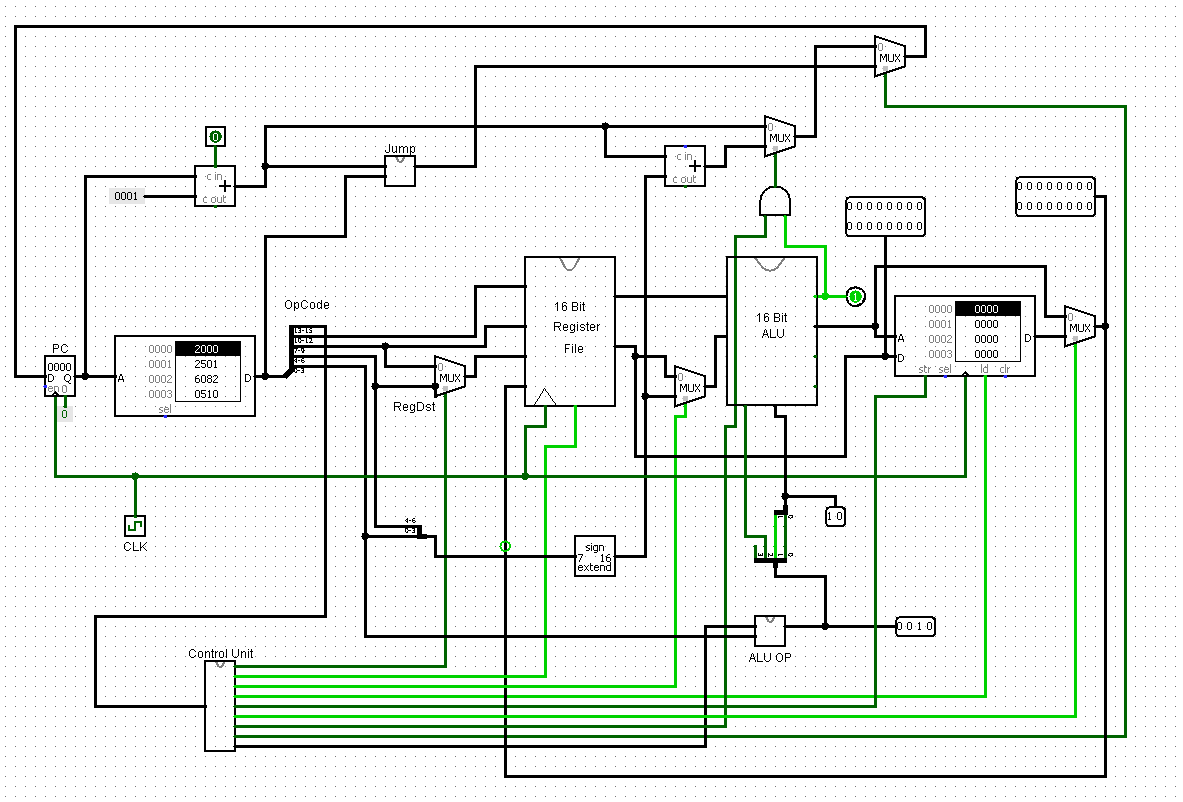
Operation: $s0 = $t0 << 4 Syntax: sll $s0, $t0, 4

## J:

It jumps a relative number of instruction/s specified by given label number. Operation: jump to Label

Syntax: J, L

**OUR MICROPROCCESSOR**

****

The design team, being the members of this group, decided to a 16-bit general purpose single cycle MIPS microprocessor. All components of the built system were modelled using logism.

Modern and advanced microprocessors can be designed to execute a lot more complex and vital task and computations. Including computations involving very large floating-point numbers. Others are designed to handle sophisticated computations in specific industries. Ours however lack the advanced and complex features to do complex computations due low level of expertise on our side.

Our microprocessor has some of the following features;

1. **Arithmetic Logic Unit**

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

1. **Registers**

A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. For example, an instruction may specify that the contents of two defined registers be added together and then placed in a specified register.

A register must be large enough to hold an instruction - for example, in a 64-bit computer, a register must be 64 bits in length.

1. **Random Access Memory (R.A.M)**

RAM (Random Access Memory) is the hardware in a computing device where the operating system (OS), application programs and data in current use are kept so they can be quickly reached by the device's processor. RAM is the main memory in a computer, and it is much faster to read from and write to than other kinds of storage, such as a hard disk drive (HDD), solid-state drive (SSD) or optical drive.

Random Access Memory is volatile. That means data is retained in RAM as long as the computer is on, but it is lost when the computer is turned off.

1. **Control Unit**

The control unit of the central processing unit regulates and integrates the operations of the computer. It selects and retrieves instructions from the main memory in proper sequence and interprets them so as to activate the other functional elements of the system at appropriate moments to perform their respective operations.

## How a Microprocessor Works

The operation of a microprocessor, though abstracted, is relatively simple to follow. The Microprocessor, can be somehow likened to the brain of a human being. As so, it is often known as the Brain of the computer.

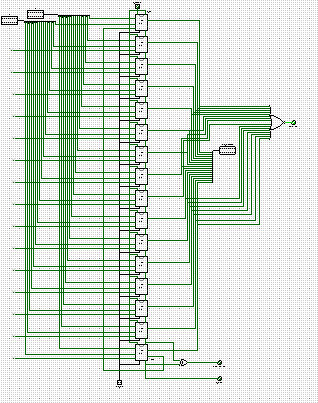
They work on based on digital logics. Every instruction is executed by a series of logic circuits that work on the principles of the different types of logic gates.

A microprocessor executes a collection of machine instructions that tell the processor what to do. Based on the instructions, a microprocessor does three basic things:

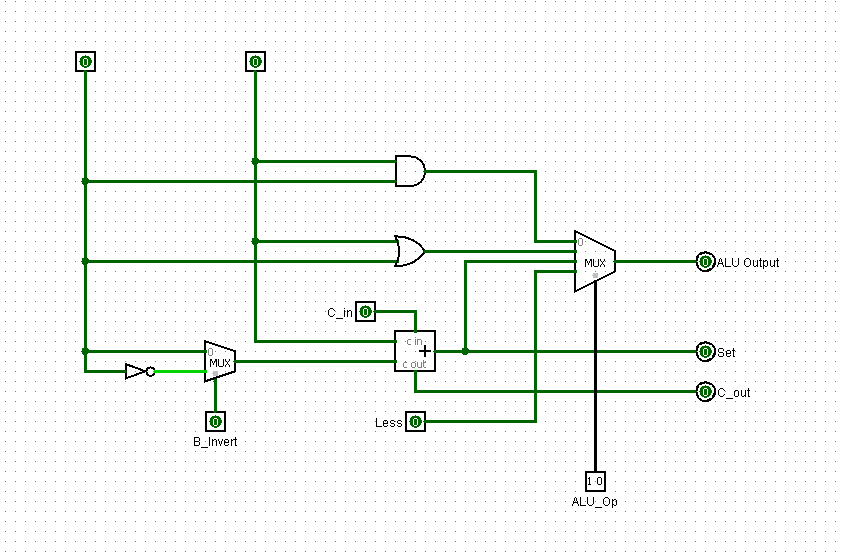
* Using its ALU (Arithmetic/Logic Unit), a microprocessor can perform mathematical operations like addition, subtraction, multiplication and division. Modern microprocessors contain complete floating-point processors that can perform extremely sophisticated operations on large floating-point numbers.
* A microprocessor can move data from one memory location to another.
* A microprocessor can make decisions and jump to a new set of instructions based on those decisions.

**Team Design ALU**

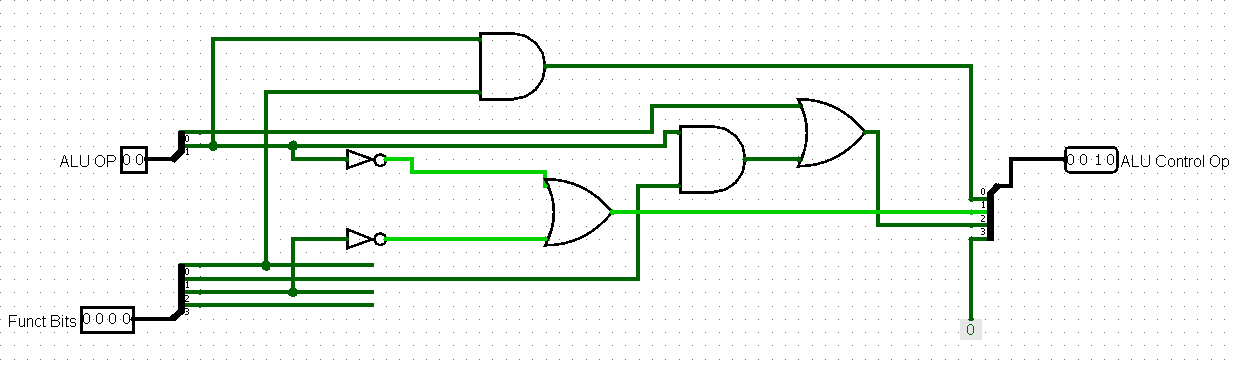
**16-bit ALU**



1-bit ALU



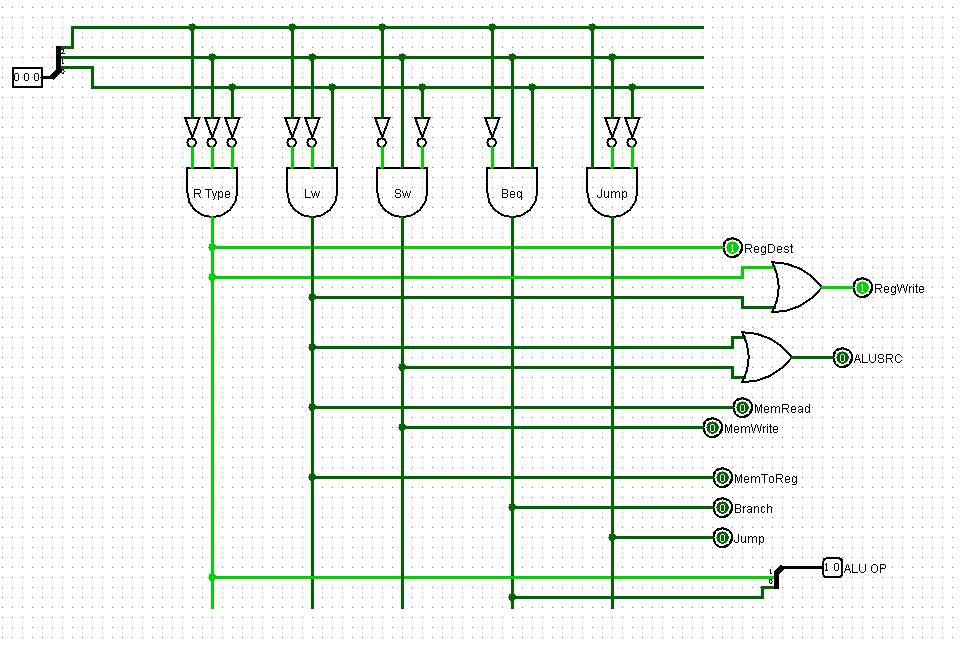
The logic of the 16-bit ALU is abstracted into the individual 1-bit ALU as shown in the above images. The team decided to build a more general purpose 1-bit ALU and then combine the 1-bit ALU to build a 16-bit ALU. The operation of the ALU is influenced by the input received from the ALU control shown below



## Operation of our ALU

The operation of the ALU is relatively simple and straight forward. Two inputs A and B are connected to the abstracted operation logic circuits. Once there is an input, the ALU executes that particular instruction and return a desired output in 16-bit. The result of the operation is sent into the multiplexer connected to our 16-bit register where it is written to.

**Team Designed Control Unit**



## Operation of the Control Unit

The operation of the Control unit is divided into three main parts. The fetch process, decode process and the execute process. The control unit is attached to the ALU (Arithmetic and Logic Unit) and several registers. (The registers mainly being memory address register MAR, instruction address register and other internal registers to keep data.)

**FETCH**

The first step the Control unit carries out is to fetch some data and instructions (program) from main memory then store them in its own internal temporary memory areas. These memory areas are called 'registers'. Fetch decode execute cycle. This is called the 'fetch' part of the cycle. For this to happen, the Control Unit makes use of a vital hardware path called the 'address bus'. The Control Unit places the address of the next item to be fetched on to the address bus.

Data from this address then moves from main memory into the control unit by travelling along another hardware path called the 'data bus'.

**DECODE**

The next step is for the control unit to make sense of the instruction it has just fetched. This process is called 'decode'.

The Control unit is designed to understand a specific set of commands. These are called the 'instruction set' of the Control unit. Each make of control unit has a different instruction set.

The Control unit decodes the instruction and prepares various areas within the chip in readiness of the next step.

**EXECUTE**

This is the part of the cycle when data processing actually takes place. The instruction is carried out upon the data (executed). The result of this processing is stored in yet another register. Once the execute stage is complete, the Control unit sets itself up to begin another cycle once more.