YEAR 3

GROUP PROJECT

ELECTRICAL/ELECTRONICS ENGINEERING

YEAR 3

MiCROPROCESSORS



**KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**COLLEGE OF ENGINEERING**

**DEPARTMENT OF ELECTRICAL/ELECTRONICS ENGINEERING**

****

**DESIGN OF A 32-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**

Processors are regarded as one of the most important devices in our everyday machines called computers. Before we start, we need to understand what exactly processors are and their appropriate implementations. 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.

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

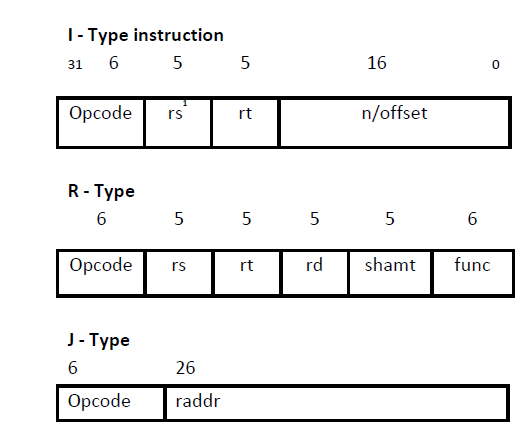
Currently, the efficacy of the RISC processors is generally accepted to be greater than that of their CISC counterparts. Before their execution the instructions are translated into RISC instructions in even the most popular CISC processors. The attributes mentioned above accentuate the design strength 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.

**32-BIT 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. As we know, implementing fewer instructions and addressing modes on silicon reduces the complexity of the instruction decoder, the addressing logic, and the execution unit. This allows the machine to be clocked at a faster speed, since less work needs to be done each clock period.

# Format



As mentioned before MIPS is a RISC microprocessor architecture. The MIPS Architecture defines 32-bit general purpose registers (GPRs). Register $r0 is hard-wired and always contains the value zero. The CPU uses byte addressing for word accesses and must be aligned on a byte boundary divisible by four (0, 4, 8, …). MIPS only has three instruction types: I-type is used for the Load and Stores instructions, R-type is used for Arithmetic instructions, and J-type is used for the Jump instructions as shown in Figure 1 which provides a description of each of the fields used in the three different instruction types.

MIPS is a load/store architecture, meaning that all operations are performed on perands held in the processor registers and the main memory can only be accessed through the load and store instructions (e.g lw, sw). A load instruction loads a value from memory into a register. A store instruction stores a value from a register to memory. The load and store instructions use the sum of the offset value in the address/immediate field and the base register in the $rs field to address the memory. Arithmetic instructions or R-type include: ALU Immediate (e.g. addi), three-operand (e.g. add, and, slt), and shift instructions (e.g. sll, srl). The J-type instructions are used for jump instructions (e.g. j). Branch instructions (e.g. beq, bne) are I-type instructions which use the addition of an offset value from the current address in the address/immediate field along with the program counter (PC) to compute the branch target address; this is considered PC-relative addressing.

The MIPS single-cycle processor performs the tasks of instruction fetch, instruction decode, execution, memory access and write-back all in one clock cycle. First the PC value is used as an address to index the instruction memory which supplies a 32-bit value of the next instruction to be executed. This instruction is then divided into the different fields shown above. The instructions opcode field bits [31-26] are sent to a control unit to determine the type of instruction to execute. The type of instruction then determines which control signals are to be asserted and what function the ALU is to perform, thus decoding the instruction. The instruction register address fields rs bits [25 - 21], rt bits [20 - 16], and rd bits [15-11] are used to address the register file. The register file supports two independent register reads and one register write in one clock cycle. The register file reads in the requested addresses and outputs the data values contained in these registers. These data values can then be operated on by the ALU whose operation is determined by the control unit to either compute a memory address (e.g. load or store), compute an arithmetic result (e.g. add, and or slt), or perform a compare (e.g. branch). If the instruction decoded is arithmetic, the ALU result must be written to a register. If the instruction decoded is a load or a store, the ALU result is then used to address the data

The MIPS implementation as with all processors, consists of two main types of logic elements: combinational and sequential elements. Combinational elements are elements that operate on data values, meaning that their outputs depend on the current inputs. Such elements in the MIPS implementation include the arithmetic logic unit (ALU) and adder. Sequential elements are elements that contain a hold state. Each state element has at least two inputs and one output. The two inputs are the data value to be written and a clock signal. The output signal provides the data values that were written in an earlier clock cycle. State elements in the MIPS implementation include the Register File, Instruction Memory, and Data Memory

**THE INSTRUCTION FETCH UNIT**

The function of the instruction fetch unit is to obtain an instruction from the instruction memory using the current value of the PC and increment the PC value for the next instruction.

**THE INSTRUCTION DECODE UNIT**

The main function of the instruction decode unit is to use the 32-bit instruction provided from the previous instruction fetch unit to index the register file and obtain the register data values. This unit also sign extends instruction bits [15 - 0] to 32-bit.

**THE CONTROL UNIT**

The control unit of the MIPS single- cycle processor examines the instruction opcode bits [31 – 26] and decodes the instruction to generate nine control signals to be used in the additional modules. The RegDst control signal determines which register is written to the register file. The Jump control signal selects the jump address to be sent to the PC. The Branch control signal is used to select the branch address to be sent to the PC. The MemRead control signal is asserted during a load instruction when the data memory is read to load a register with its memory contents. The MemtoReg control signal determines if the ALU result or the data memory output is written to the register file. The ALUOp control signals determine the function the ALU performs. (e.g. and, or, add, sbu, slt) The MemWrite control signal is asserted when during a store instruction when a registers value is stored in the data memory. The ALUSrc control signal determines if the ALU second operand comes from the register file or the sign extend. The RegWrite control signal is asserted when the register file needs to be written. Table 1 shows the control signal values from the instruction decoded.

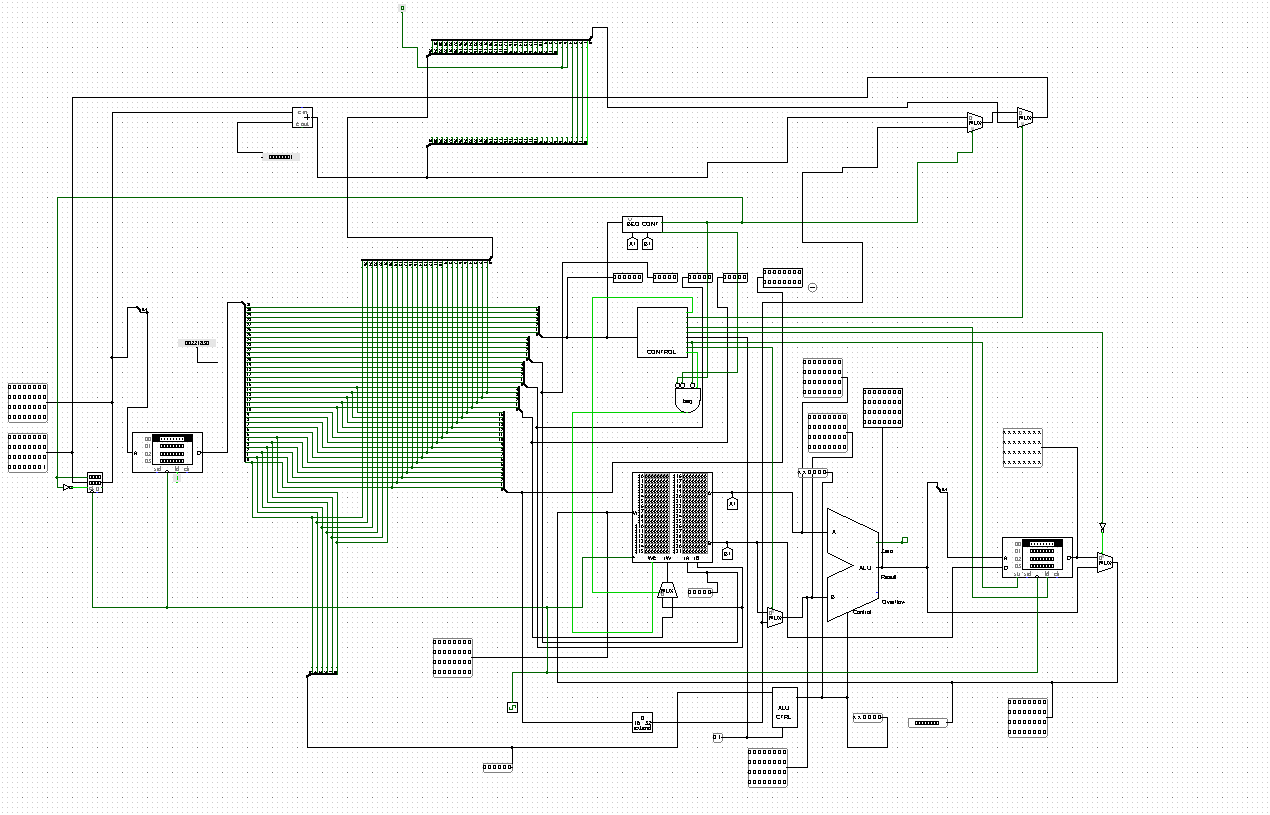
**EXECUTION UNIT**

The execution unit of the MIPS processor contains the arithmetic logic unit (ALU) which performs the operation determined by the ALUop signal. The branch address is calculated by adding the PC+4 to the sign extended immediate field shifted left 2 bits by a separate adder.

**THE DATA MEMORY UNIT**

The data memory unit is only accessed by the load and store instructions. The load instruction asserts the MemRead signal and uses the ALU Result value as an address to index the data memory. The read output data is then subsequently written into the register file. A store instruction asserts the MemWrite signal and writes the data value previously read from a register into the computed memory address.

**OUR MICROPROCCESSOR**

****

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.

## 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.

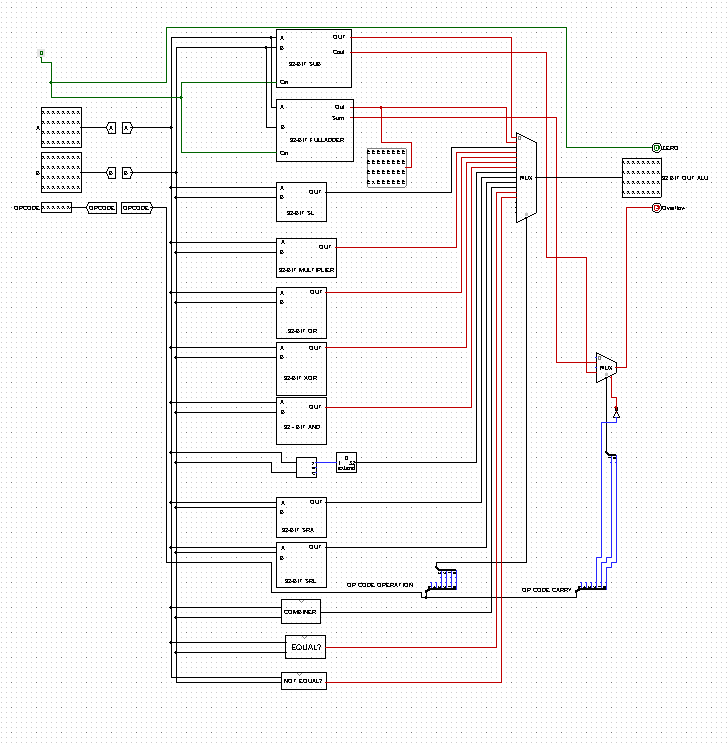
They work on based on digital logics. Every instruction is executed by a series of logic circuits that work on the principles of the different types of logic gates.

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**

**32-bit ALU**

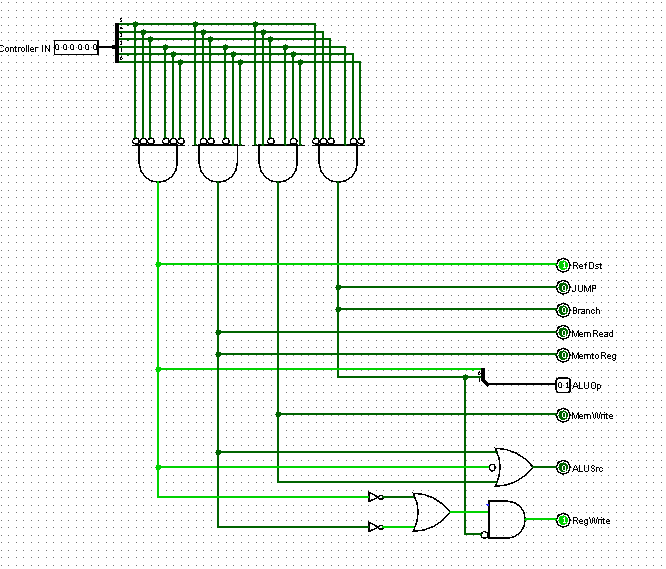


The logic of the 32-bit ALU is abstracted into the individual components as shown in the above image. Though abstracted the ALU operation remains relatively simple to follow.

## 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 32-bit. The result of the operation is sent into the multiplexer connected to our 32-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.