

RESEARCH PROJECT SUBMISSION

Architecture Final Assessment

**Program: CESS**

***Course Code: CSE 112***

***Course Name: Computer Organization and Architecture***

***Examination Committee***

**Dr. Tamer Mostafa**

**Ain Shams University**

**Faculty of Engineering**

**Spring 2020 Semester**

**Student Personal Information**

Students’ Names & IDs: Sherif Ahmed Naiem Mohamed 18P6546

Yomna Hussein Mohamed Abd El Hamid 18P5794

Omar Mohamed Lotfy EL-Sayed 18P5606

Nour El-Din Mohamed Hesham 18P5806

Reda Mohsen Reda 18P5141

**Plagiarism Statement**

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

**Sherif Ahmed/ Yomna Hussein/ Omar Lotfy/ Nour El-Din Mohamed/ Reda Mohsen**

**Signature/Student Name:**

**3-6-2020**

**Date:**

**Submission Contents**

**01: Introduction to MIPS Code.**

**02: Registers.**

**03: Data Path of Different MIPS Instructions**

**04: Instruction Formats.**

**05: Coding & Testing.**

**01**

***First Topic***

**Table of Contents**

[I. Roles Distribution 4](#_Toc41933142)

[II. Introduction to MIPS code: 5](#_Toc41933143)

[A. What is Assembly Language? 5](#_Toc41933144)

[III. Register: 5](#_Toc41933145)

[IV. Data Paths of Different MIPS Instructions: 6](#_Toc41933146)

[B. ADD: 6](#_Toc41933147)

[C. LW: 7](#_Toc41933148)

[D. SW: 8](#_Toc41933149)

[E. BNE: 9](#_Toc41933150)

[F. J: 11](#_Toc41933151)

[V. Instruction Formats: 11](#_Toc41933152)

[G. R-Format: 12](#_Toc41933153)

[H. I-Format: 12](#_Toc41933154)

[I. J-Format: 12](#_Toc41933155)

[VI. List of References 13](#_Toc41933156)

# Roles Distribution

1. Yomna Hussien Mohamed

* In Report: Section IV.F, R-Format
* In Coding: I & J Format , BEQ execution code

1. Sherif Ahmed Naiem

* In Report: Section IV.C&D, I-Format
* In Coding: I & J Format, J execution code

1. Omar Mohamed Lotfy

* In Report: Section IV.B, J-Format
* In Coding: I & J Format, SLTI execution code

1. Nour El-Din Mohamed

* In Report: Introduction , Section IV.E
* In Coding: Class handling, Class Operations, R Format, Add execution code

1. Reda Mohsen

* In Report: Registers
* In Coding: Class instructions, Class print, R Format

**Note:** All the team first discussed the project carefully and then we divided the work among us and then after each team member had done his part we all gathered these parts and combined them together to produce this code that translate and execute MIPS instructions code.

# Introduction to MIPS code:

MIPS low level computing language (Assembly) basically alludes to the assembly language of the MIPS processor. The term MIPS is an abbreviation which represents Microprocessor without Interlocked Pipeline Stages, and it is a diminished operation set engineering architecture which was created by an association called MIPS Technologies.

MIPS is a load/store design; Load/store especially used to get to memory, all guidelines and instructions work on the registers.

Those computer instructions are represented in form of bits. Actually, this is the lowest possible level of representation for a program. Each instruction is equivalent to a single action of the CPU. In order that the machine understands this language, it is called machine language [1].

## What is Assembly Language?

A more significant level portrayal and one that is a lot simpler for people to utilize is called Assembly language. Assembly Language firmly associated with machine language, and there is normally a direct relationship to interpret programs written in Assembly into an executable and practical machine code by a [utility program](https://en.wikipedia.org/wiki/Utility_software) referred to as an [assembler](https://en.wikipedia.org/wiki/Assembly_language#Assembler)/compiler. The conversion procedure is alluded to assembly, as in gathering the source code. Assembly language as a rule has one line of code per machine instruction (1:1), yet remarks and explanations that are assembler directives, macros, and significant labels of program and memory areas are frequently additionally boosted [4].

# Register:

A processor register is one of a little set of information holding places that are a piece of the PC processor. A register may hold an instruction, a capacity address, or any sort of information, (for example, a sequence of bits or individual characters). A few guidelines determine enrolls as a component of the instructions. For instance, an instruction may determine that the contents of two characterized registers be included and afterward positioned in a predefined register.

A register must be sufficiently large to hold a guidance - for example, in a [32-bit computer](https://searchdatacenter.techtarget.com/definition/64-bit-processor), a register must be 32 [bits](https://whatis.techtarget.com/definition/bit-binary-digit) in length. In some PC designs, there are smaller registers - for instance, half-registers - for shorter guidelines and instructions. According to the processor design and language rules, registers might be numbered or have discretionary names. A processor normally contains numerous index registers, also called address/index registers or registers of adjustment. The specific location of any entity in a PC incorporates the base, index, and relative locations, which are all put away in the index register. A shift register is another sort of registers. Bits enter the shift register toward one side and rise up out of the opposite end. Flip flops, also called bitable entries, store and procedure the information. General Purpose Registers (GPRs) are demonstrated with a dollar sign ($). The words SWORD and UWORD allude to 32-piece marked and 32-piece unsigned data types, respectively [2].

# Data Paths of Different MIPS Instructions:

Instruction in MIPS may vary according to the aim of the user, such like: ADD, ADDI, SUB, SLL,SRL,SRA, AND, ANDI, OR, ORI, NOR, STL, SLTI, LUI, LW, SW,BEQ,BNE, and J.

The previous instructions are classified according to their aim [1]:

* Arithmetic operations: ADD, ADDI, SUB,
* Logic operations: AND, ANDI, OR, ORI, NOR, SLL, SRL, SRA
* Compare instructions: SLT, SLTI
* Move instructions: LUI
* Load and store: LW, SW
* Branch instruction: BEQ, BNE, J

In this section, we will discuss the process of each instruction, and how it is executed in each clock cycle. This is acknowledged as the single-cycle design. We will recognize the constraints of the single-cycle design.

**How instructions are fetched**

Data and instructions are stored in memory. In order for the instruction to be executed, 1st it has to be read out of the memory. The address of the instruction that we are executing is saved in a unique program counter register in MIPS. This process is performed automatically, We don't have to search for the following instruction stored in the memory.

## ADD:

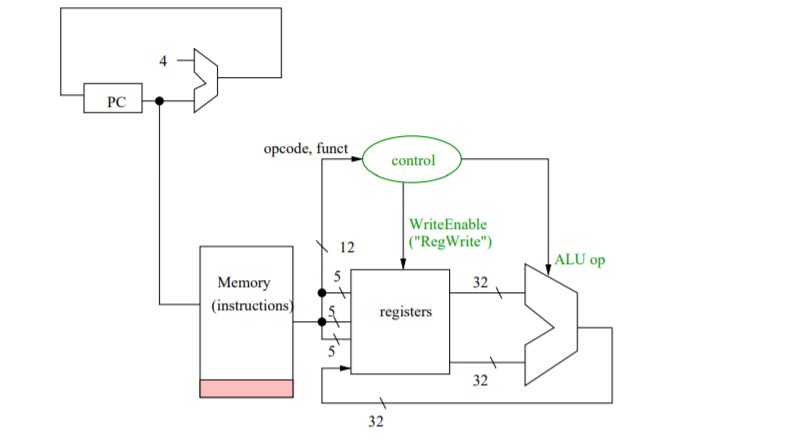
The opcode ﬁeld and function ﬁelds mutually encode that the operation is an addition. The three register ﬁelds of the instruction define which registers contain the operands and which register will take the output of the ALU. the opcode and function ﬁelds are used to control what data gets puts on what lines and when data gets written.

**what happens during an add instruction:**

* At the end of a clock cycle a new PC value is calculated and written.
* instruction is read from memory
* control signals for ALU operation are set.
* • RegData values are read from two registers and input to ALU.
* At the end of a clock cycle outcome, the write-register will carry the outcome.

All the above steps are done in a single cycle since the clock interval time can stabilize all circuits before the time interval is up.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Area | OP | RS | RT | RD | Shamt | Funct |
| Bits | 6 | 5 | 5 | 5 | 5 | 6 |

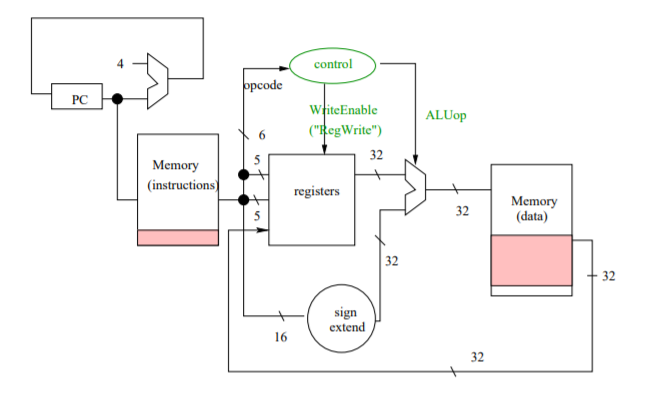


## LW:

**The steps for LW**:

* Fresh PC value is computed (it is written at next clock pulse)
* Instruction is read (“fetched”) from memory
* ReadReg and WriteReg picked and RegWrite is specified
* Base address for memory word is read from a register and fed to ALU
* Offset (16 bit immediate field) is sign-extended and fed to ALU
* Control signals are arranged for ALU execution.
* The result is sent to Memory after ALU calculates the sum of the base address and sign-extended offset.
* When the clock cycle reaches its end, the word that is obtained from the memory is written in a register.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Area | OP | RS | RT | Immediate |
| Bits | 6 | 5 | 5 | 16 |

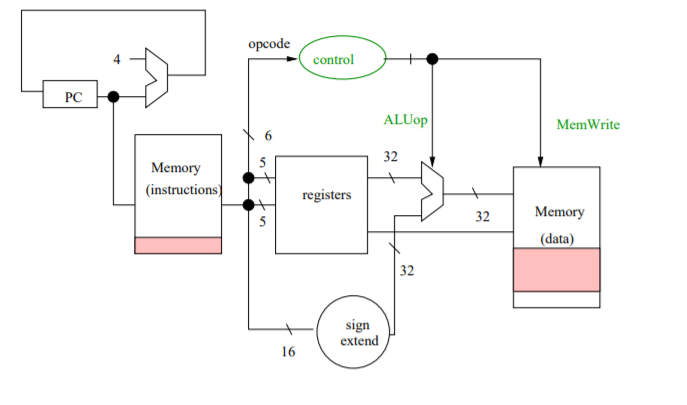


## SW:

**The steps for SW:**

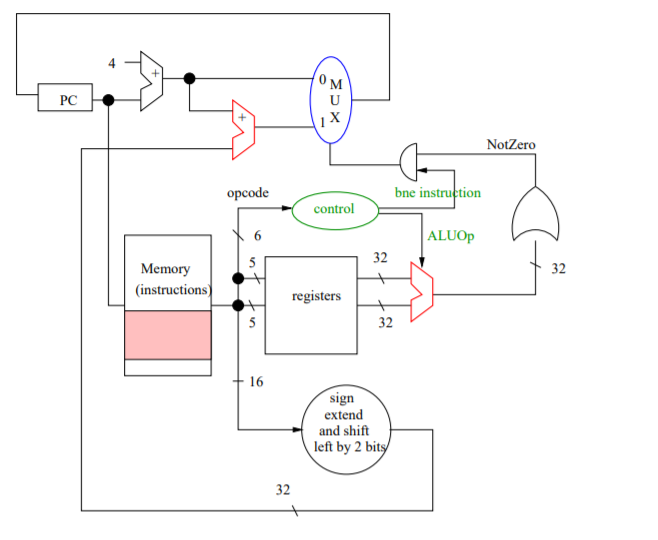
* fresh PC value is written at the subsequent clock pulse.
* instruction is brought from memory
* two ReadReg are chosen
* • base address for memory word is gathered from a register and supplied to ALU
* offset (16 bit immediate field) is sign-extended and fed to ALU
* control signals are arranged for ALU operation
* The result is transferred to Memory after ALU calculates the sum of the base address and sign-extended offset.
* When the clock cycle ends, the word is addressed into Memory.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Area | OP | RS | RT | Immediate |
| Bits | 6 | 5 | 5 | 16 |



## BNE:

Next how about we take a glance at the case that the instruction is a conditional branch, for instance, bne. This insstruction is more interesting as the computer probably will not take the value of PC+4. This instruction uses ALU, and performs a comparison operation; as if both arguments passed to the register are not equal, then the branching takes place according to the offset value. Otherwise, if both arguments are equal, then branching will not take place.  
See the figure underneath for the datapath. The multiplexor in the upper piece of the figure chooses somewhere in the range of PC+4 and PC+4+offset. The selector signal relies upon the 32 bit yield of the ALU, which has deducted the two registers determined by the bne guidance. This 32 bit ALU result is taken care of into a 32 bit OR GATE. The yield of the OR GATE is marked "Not Zero" in the figure. This NotZero signal is on if the two registers don't have similar arguments. Note that the branch is possibly taken if without a doubt the instruction is bne. In the figure, the NotZero signal is ANDed with a bne instruction control signal that demonstrates it is without a doubt a bne instruction. (I have named the branch control as "bne instruction" in the figure.) The branch is taken when both the "Not Zero" signal is ON and the branch control is on. For an ADD instruction or any other R-format instruction, the branch sign would be OFF (0). Additionally, notice that I have designed the circuit such a way, that the balance is comparative with the current instruction address in addition to 4, as opposed to the instruction address.



MIPS assumes that the two most minimal order bits of a specific address have got the value 0. In this way, in the 16 bit address field of a contingent branch, MIPS doesn't consider these two bits by continually setting them to 00. Moreover, the 16 bits field speaks to a balance of 4× the balance portrayal in double. That is the balance system is "moved" by two bits to one side before it is added to PC+4. no shift register is required for this. Moreover, two 0 wires are embedded straightforwardly, and the sign expand circuit reaches out from 16 to 30 bits, so altogether there are 32 bits.

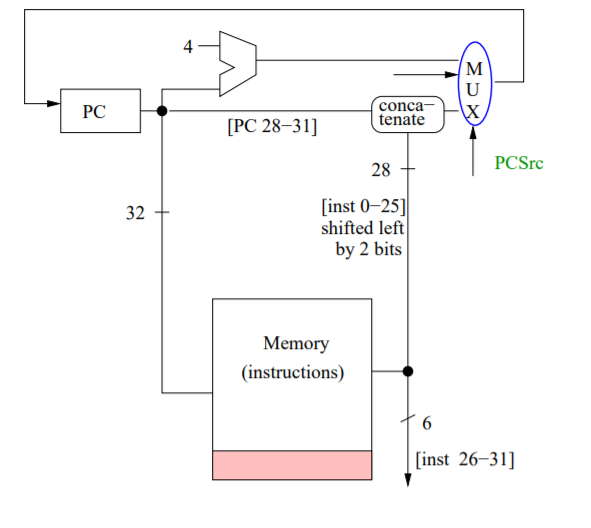
**Here are the means of the bne guidance:**

* PC holds the location of the present guidance
* Instruction is brought from memory
* PC+4 esteem is processed
* Estimation of PC+4 is added to sign-expanded/moved counterbalance
* ReadReg values are set
* Control signals are set up for deduction in ALU
* RegData values are perused from two registers into ALU
* ALU does deduction and 32 bit result is taken care of into goliath OR GATE, whose yield we call NotZero
* NotZero is ANDed with a bne control to choose either PC+4 or PC+4+offset, and chosen value is written into the computer

## J:

To apply the jump instruction we have to know the address of the following instruction. Due to the words are aligned in instructions, the 26-bit area is handled to define 2-27 bits. Bits 0 & 1 hold the value 00. While the higher 4 bits are defined by the current higher 4 bits of the program counter. MIPS instructions don't include 0x10000000. Therefore, instructions have 0000 as their most leading 4 bits. MIPS doesn’t regularly connect 0000 on the beginning of a jump instruction as there are other programs too. Instructions in other programs like kernel have addresses that go higher 0x80000000. Hence jump instructions in that program don't take 0000 as their higher 4 bits. Preferably, they hold (1000)2 as their higher 4 bits.

|  |  |  |
| --- | --- | --- |
| Area | OP | address |
| Bits | 6 | 26 |



# Instruction Formats:

Instruction format is a format which describes the layout of a certain instruction, in terms of divided parts. It includes opcode (6 bits), and the address dependent factors (26 bits).

Instruction Formats are divided into 4 essential formats, each format is responsible for certain instructions which are [3]:

## R-Format:

It includes op which is always zero (6 bits), rs which is the argument of the 1st register (5 bits), rt which is the argument of the 2nd register (5 bits), rd which is the destination register (5 bits), shamt-which is used only in case of shift instructions-(5 bits), func which represents the code for the operation to form (6 bits)🡪 32 bits in total.

*R-Format Instructions:* ADD, AND, NOR, OR, SLL, SLT, SRL, SRA [3].

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | op | rs | rt | rd | shamt | Func. |
| R-Format | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

## I-Format:

It includes op (6 bits), rs which is the argument of the 1st register (5 bits), rt which is the argument of the 2nd register (5 bits), and immediate address (16 bits) 🡪 32 bits in total.

*I-Format Instructions:* ADDI, ANDI, BEQ, BNE, LUI, LW, ORI, SLTI, SW [3].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | op | rs | rt | Immediate Address |
| I-Format | 6 bits | 5 bits | 5 bits | 16 bits |

## J-Format:

Includes op (6 bits), and a label address (26 bits) 🡪 32 bits in total

*J-Format Instructions:* J

|  |  |  |
| --- | --- | --- |
|  | op | Target address |
| J-Format | 6 bits | 26 bits |

# List of References

[1] Vollmar, K., & Sanderson, P. (2006, March). MARS: an education-oriented MIPS assembly language simulator. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education* (pp. 239-243)

[2] Flynn, M. J. (1995). *Computer architecture: Pipelined and parallel processor design*. Jones & Bartlett Learning.

[3] Gupta, S. A., Rau, B. R., Johnson, R. C., & Schlansker, M. S. (2002). *U.S. Patent No. 6,457,173*. Washington, DC: U.S. Patent and Trademark Office.

[4] Morrisett, G., Walker, D., Crary, K., & Glew, N. (1999). From System F to typed assembly language. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *21*(3), 527-568.