



Bangladesh University of Engineering and Technology

CSE 306  
Computer Architecture Sessional  
Assignment 3  
Section: A2  
Group no.: 4

Group Members : 1805046  
1805047  
1805048  
1805052  
1805057

Date of Submission : August 17, 2022

# Introduction

MIPS (Microprocessor without Interlocked Pipelined Stages) is a RISC (Reduced Instruction Set Computer) ISA (Instruction Set Architecture). It was developed in 1985 by MIPS Computer Systems (now MIPS Technologies). The instructions of MIPS are fixed and rigid, which ensures regularity. For example, the *addition* and *subtraction* instructions always follow the following fixed formats:

Operation	Instruction	Action
ADDITION	add \$t1, \$t2, \$t3	\$t1 = \$t2 + \$t3
SUBTRACTION	sub \$t1, \$t2, \$t3	\$t1 = \$t2 - \$t3

Here, \$t1, \$t2, \$t3 are registers that hold values. To evaluate the expression,  $n = a + b - c$ , one set of instructions can be (assuming  $a, b, c$  are stored in \$t1, \$t2, \$t3 respectively and  $n$  will be stored in \$t0),

```
sub $t0, $t2, $t3    # $t0 = $t2 - $t3, i.e.,  $n = b - c$ 
add $t0, $t1, $t0    # $t0 = $t1 + $t0, i.e.,  $n = a + b - c$ 
```

According to MIPS instruction properties, arithmetic operations take only register values as operands, where the size of a register is 32 bits and there are 32 registers. Under these properties, MIPS provides different types of instruction formats for different types of operations:

- R-Format: To deal with arithmetic operations with registers

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Here, **op** := opcode to denote operation type and format type

**rs** := first source register

**rt** := second source register

**rd** := destination register

**shamt** := shift amount

**funct** := function code to denote specific variant of an operation

For example, consider the instruction, where the data in \$t1 and \$t2 get added and then saved into \$t0),

add \$t0, \$t1, \$t2

which is structured in MIPS as,

0	9	10	8	0	32
---	---	----	---	---	----

i.e.,

000000	01001	01010	01000	00000	100000
--------	-------	-------	-------	-------	--------

- I-Format: To deal with constants and data transfer operations

op	rs	rt	constant or address
----	----	----	---------------------

6 bits

5 bits

5 bits

16 bits

Here, rs := source register

rt := destination register

constant or address := the constant or the address to store/retrieve the value to/from

For example, consider the expression,  $a = A[3]$ , where  $a$  is a variable and  $A$  is an array; the corresponding instruction will be (assuming  $A$  is stored in \$t1, and  $a$  will be stored in \$s0),

lw \$t0, 12(\$t1)

sw \$t0, 0(\$s0)

which is structured in MIPS as,

35	9	8	12
43	8	16	0

i.e.,

100011	01001	01000	0000 0000 0000 1100
101011	01000	10000	0000 0000 0000 0000

- J-Format: To support long jump to a remote address

op	jumping address
6 bits	26 bits

For example, consider the instruction (assuming, address of **target** is 1200),  
**j target**  
 which is structured in MIPS as,

2	1200
---	------

i.e.,

000010	0000 0100 1011 0000
--------	---------------------

A datapath is built with elements that process data and addresses in the CPU such as registers, ALUs, MUXs, memories, and controls. The MIPS instructions are fed through a datapath to perform different instructions like addition, load/store, branching, or jump. A datapath, thus, is also the depiction of flow of data through different components of a computer while executing a certain instruction.

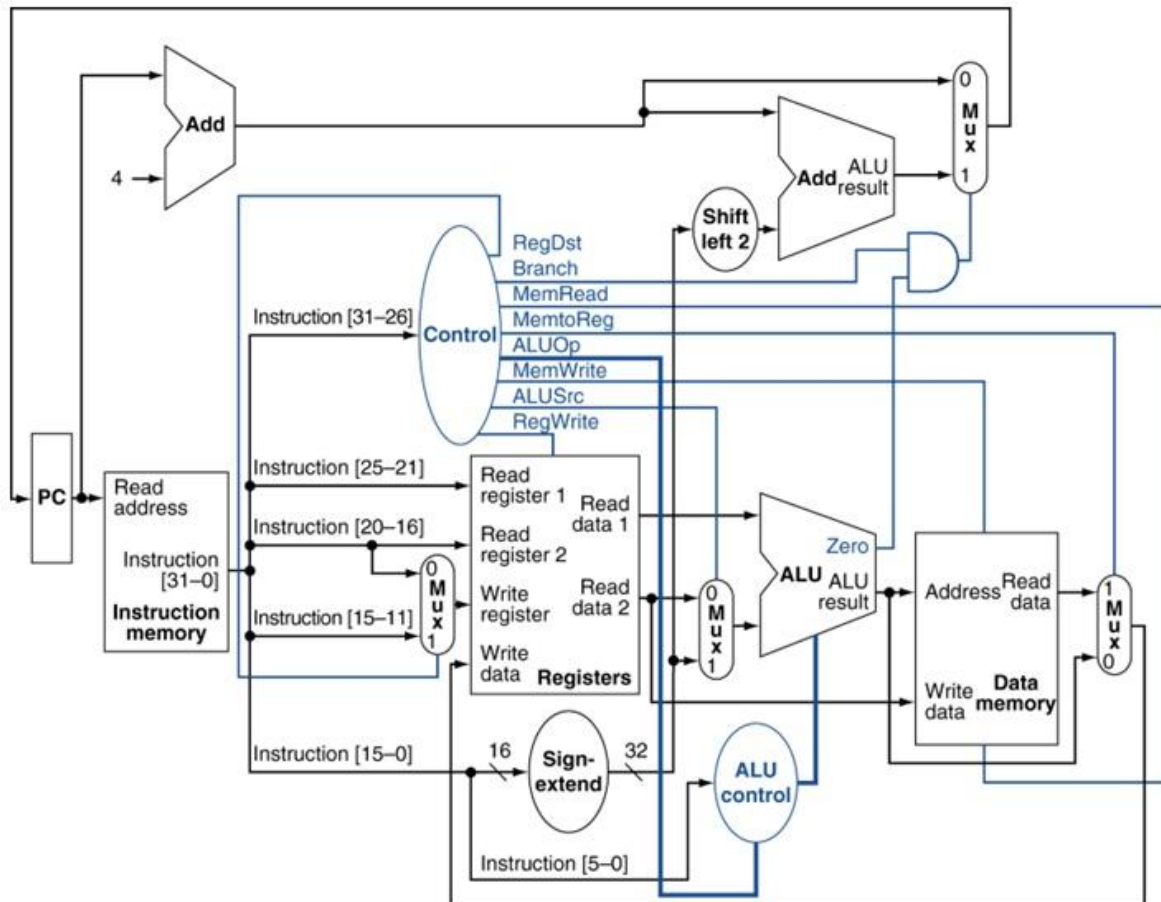


Fig. 1: A datapath with control

# Problem Specification

In this assignment, we are instructed to implement a modified and reduced version of the MIPS instruction set. Here, the Address bus will be of 8 bits, the data bus will be of 4 bits and there will be a 4-bit ALU. The reduced MIPS instructions will be 16-bit long with the following four formats:

- R-type:

op	rs	rt	rd
4 bits	4 bits	4 bits	4 bits

Here,  $rt :=$  second source register,  
 $rd :=$  destination register

- S-type:

op	rs	rt	shamt
4 bits	4 bits	4 bits	4 bits

Here,  $rt :=$  destination register  
 $shamt :=$  shift amount

- I-type:

op	rs	rt	constant or address
4 bits	4 bits	4 bits	4 bits

Here,  $rt :=$  destination register

- J-type:

op	jumping address	0
4 bits	8 bits	4 bits

Additionally,  $op$  stands for operation code or opcode to denote operation type and format type and for  $rs$  (first) source register.

# Instruction Set

ID	Instruction	Type	Format	Op Code
E	and	Logic	R-type	0
I	sll	Logic	S-type	1
B	addi	Arithmetic	I-type	2
H	ori	Logic	I-type	3
G	or	Logic	R-type	4
M	sw	Memory	I-type	5
C	sub	Arithmetic	R-type	6
D	subi	Arithmetic	I-type	7
N	beq	Control	I-type	8
F	andi	Logic	I-type	9
P	j	Control	J-type	10
K	nor	Logic	R-type	11
O	bneq	Control	I-type	12
L	lw	Memory	I-type	13
A	add	Arithmetic	R-type	14
J	srl	Logic	S-type	15

# Circuit Diagram

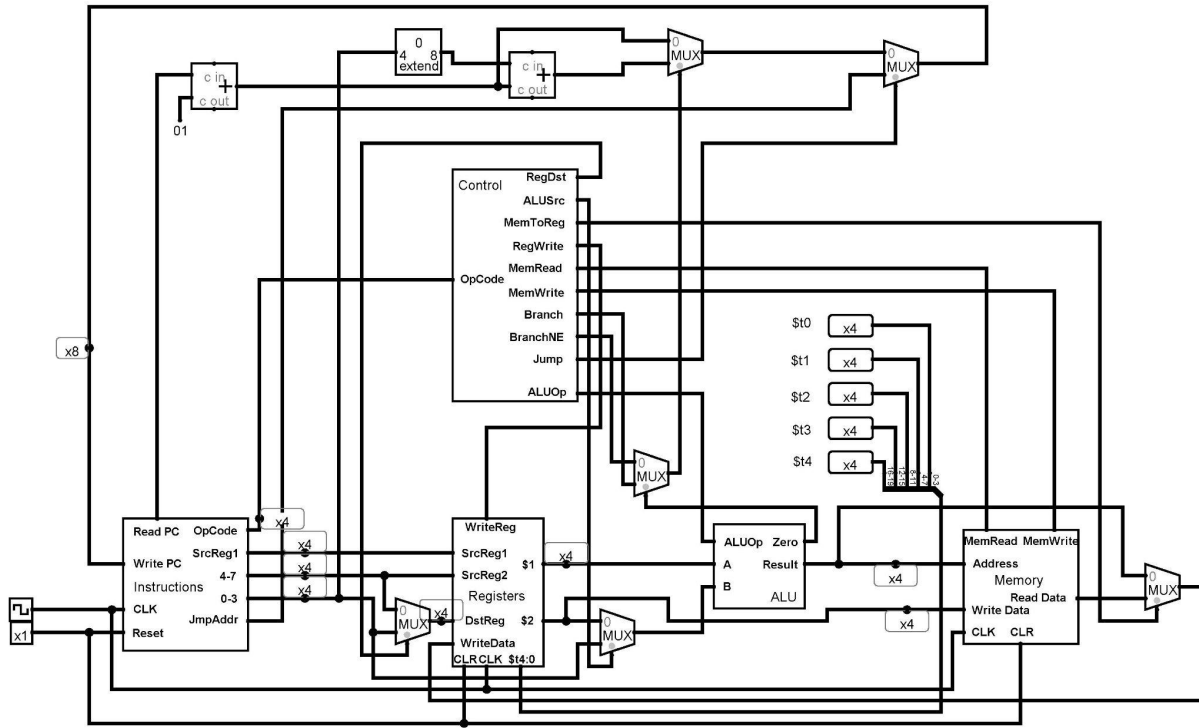


Fig. 2: Datapath for 4-bit MIPS

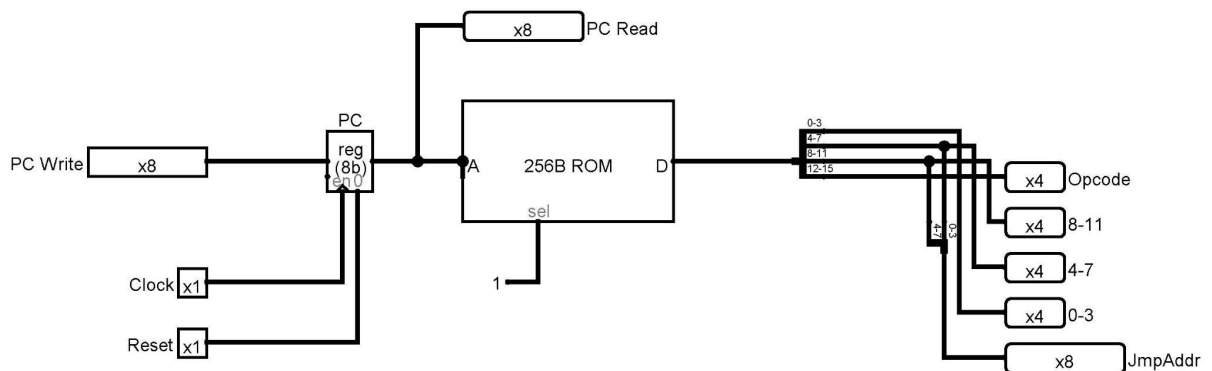


Fig. 3: Instruction used in 4-bit MIPS



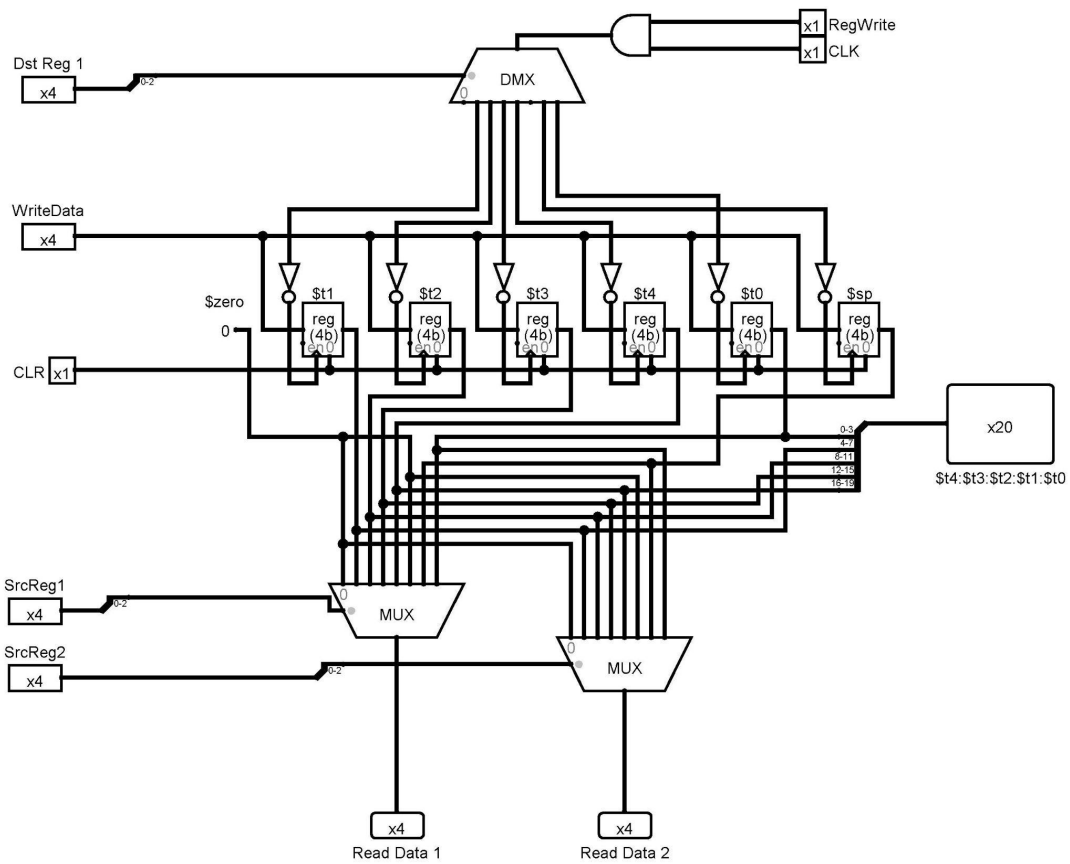


Fig. 4: Registers used in 4-bit MIPS

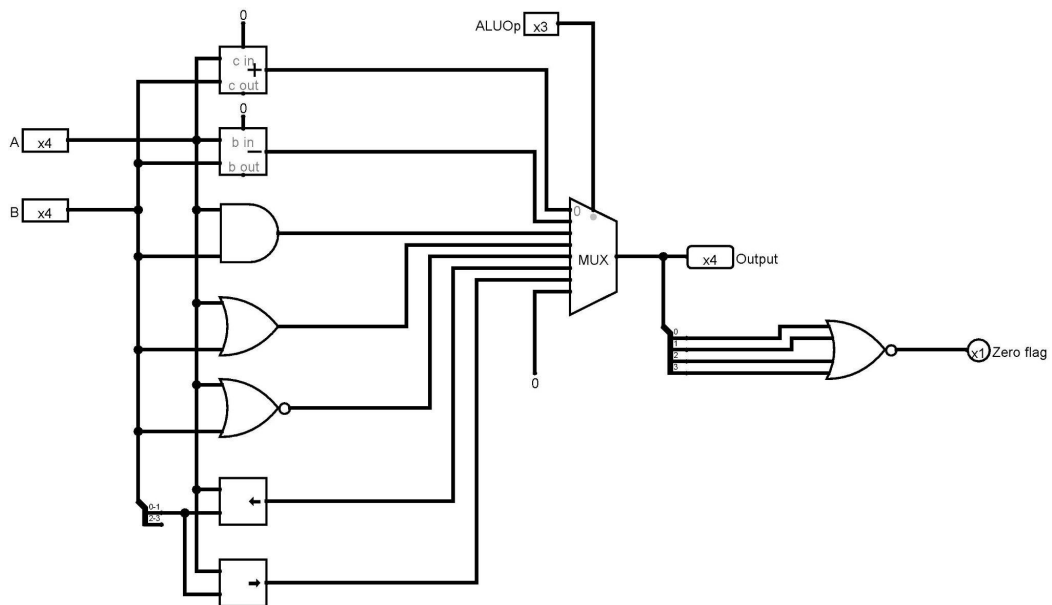


Fig. 5: ALU used in 4-bit MIPS

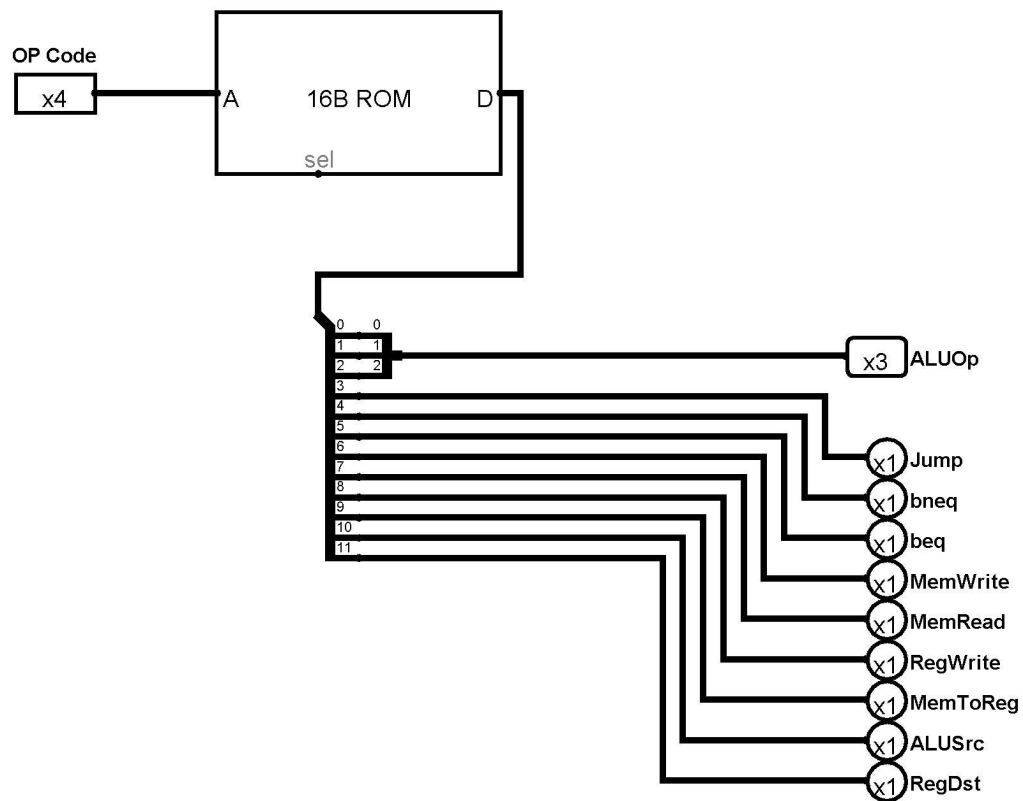


Fig. 6: Control used in 4-bit MIPS

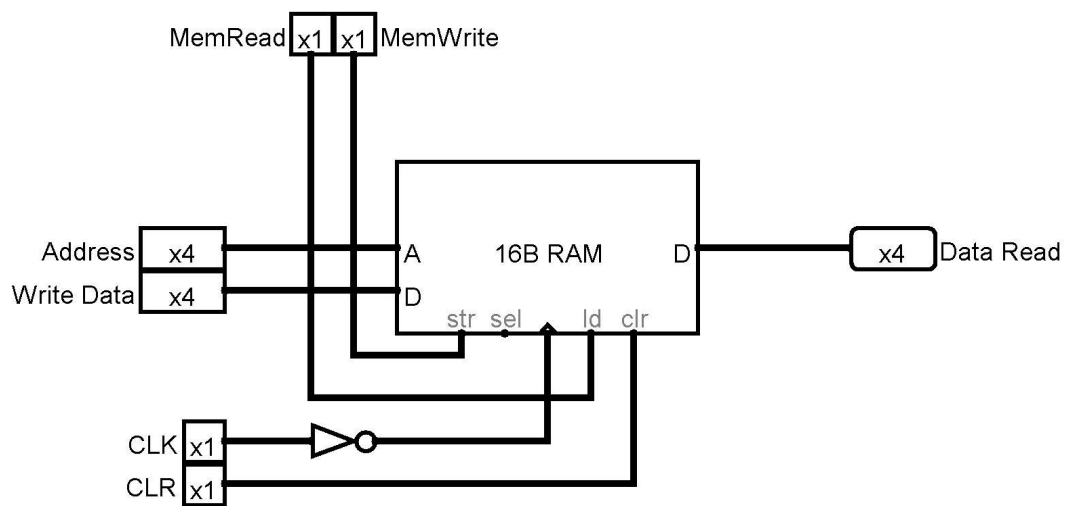


Fig. 7: Data Memory used in 4-bit MIPS

# Program Execution Instructions

## Step 1: Machine Code Generation

- Write the assembly code in input.asm
- Run the provided .py file to generate a output.hex file and a output.bin file

## Step 2: Copying Code Contents

- For *software simulation*, open the provided .circ file in logisim (preferable version: v2.7.1) and go to the instruction block and copy the contents of the output.hex file into ROM contents
- For *hardware execution*, burn the ATMEGA32 with the instruction.hex file generated by Atmel Studio by copying the contents of the output.hex file into the provided .cpp file

## Step 3: Run/Execute

- For *software simulation*, go to the main circuit (main.circ) and enter clock pulse to simulate
- For *hardware execution*, provide clock pulse with a push button to start execution

## ICs

Components	Details	Count
IC 7404	Not Gate	1
IC 7483	4-bit Adder	4
IC 74157	Quad 2-to-1 MUX	9
IC 74273	8-bit D Flip-Flop	1

ATMEGA32	Microcontroller	5
----------	-----------------	---

## Discussion

In this assignment, we were instructed to design, simulate and implement a modified and reduced version of the MIPS instruction set. The original MIPS does not have any S-format instructions, but in our reduced MIPS, a S-format was implemented. Additionally, other formats of instructions were also modified. For example, in the reduced R-format, there are no bits reserved for `shamt` or `funct`. The Instruction Memory, Registers, Data Memory, Control and ALU were replaced by ATMEGA32. IC 74273 was used as the Program Counter (`pc`). The `.py` file serves as a manual assembler. Stack Pointer (`$sp`) was implemented as a bonus.