Bangladesh University of Engineering and Technology

June 20, 2021

CSE-306: Computer Architecture Sessional

Report on 8-bit MIPS Design and Simulation

Submitted By

Section: B1

Group: 03

1705068

1705069

1705070

1705071

1705074

Contents

1 Introduction **2**

2 Instruction Set **3**

3 Instruction Format **3**

4 Block Diagram **4**
4.1 Block diagram of an 8-bit MIPS processor 4

5 Circuit Diagram **5**
5.1 Complete Circuit diagram of an 8-bit MIPS 5

6 Description of MIPS Processor **5**

7 Block diagrams of the main components **7**
7.1 Instruction memory with PC 7
7.2 Register File 7
7.3 Data memory with the Stack 8
7.4 Control Unit 8
7.5 Control Unit Table 9

8 Description of Control Unit **10**

9 Push and Pop Instruction **11**

10 IC used with Count **12**

11 Simulator **12**

12 Discussion **13**

1 Introduction

MIPS is a reduced computer instruction set architecture which defines the interface between a user and a microprocessor. In this offline, each instruction executes in one clock cycle. For every instruction, the first two steps are identical.

1. Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.
2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require reading two registers.

After these two steps, the actions required to complete the instruction depend on the instruction class.The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.

In this assignment we designed a 8-bit MIPS which implements a custom instruction set.

2 Instruction Set

Instruction Type	**Instruction**	**Category**
I	addi	Arithmetic
I	subi	Arithmetic
I	andi	Logic
I	ori	Logic
I	sw	Memory
I	lw	Memory
I	beq	Control-conditional
I	bneq	Control-conditional
R	add	Arithmetic
R	sub	Arithmetic
R	and	Logic
R	or	Logic
R	sll	Logic
R	srl	Logic
R	nor	Logic
J	j	Control-unconditional

Table 1: Instruction Set

3 Instruction Format

R-type	OpCode 4-bits	Src Reg1 4-bits	Src Reg2 4-bits	Dst Reg2 4-bits	Shift Amount 4-bits

Table 2: R-type Instruction Set

I-type	OpCode 4-bits	Src Reg 4-bits	Dst Reg 4-bits	Address / Immediate 4-bits

Table 3: I-type Instruction Set

J-type	OpCode 4-bits	Target Jump Address 4-bits	0 4-bits	0 4-bits

Table 4: J-type Instruction Set

4 Block Diagram

4.1 Block diagram of an 8-bit MIPS processor

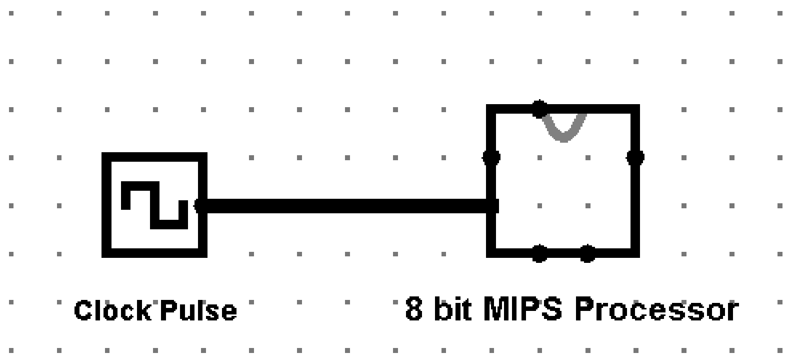


Figure 1: Block Diagram of MIPS

5 Circuit Diagram

5.1 Complete Circuit diagram of an 8-bit MIPS

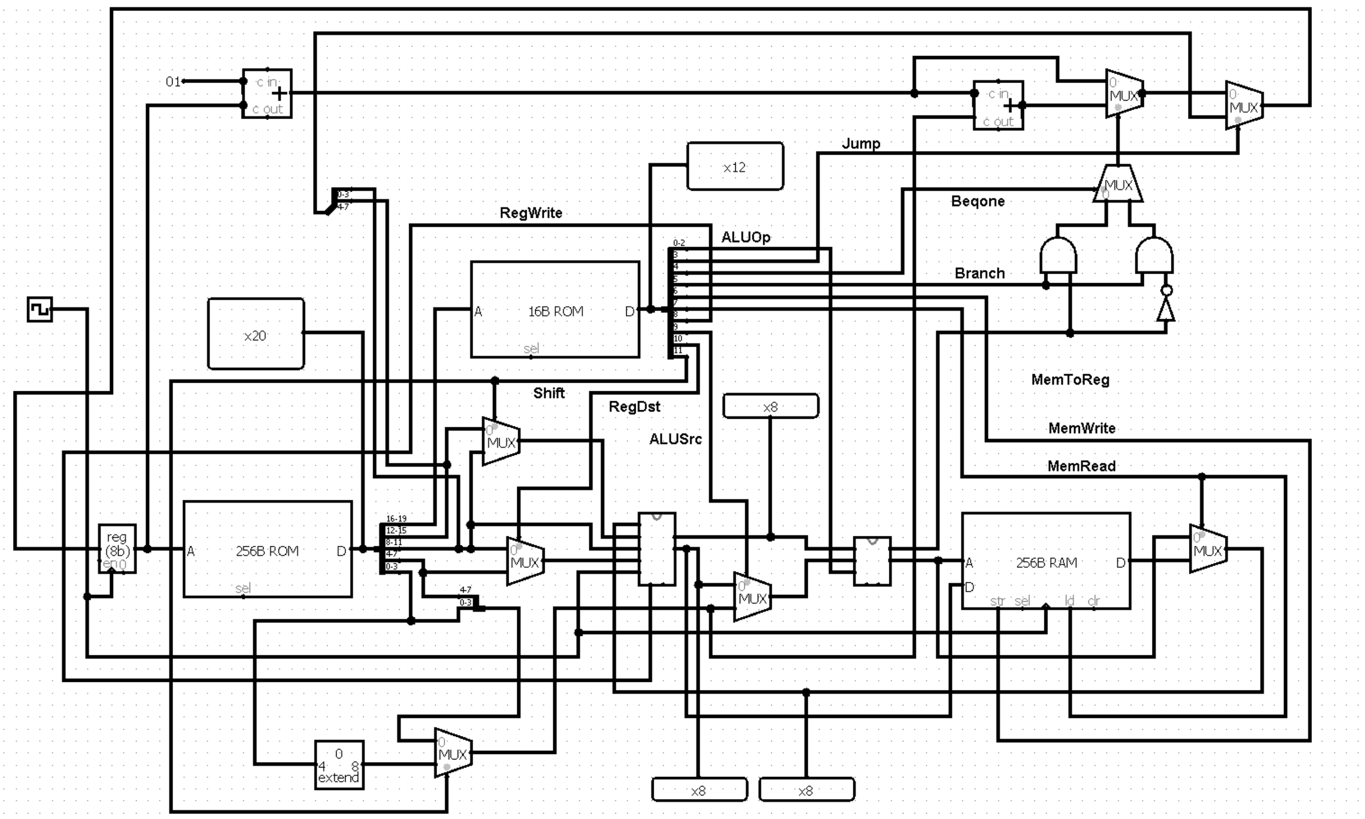


Figure 2: 8-bit MIPS circuit

6 Description of MIPS Processor

The main components in our 8bit MIPS processor are as follows:

1. Program counter register PC
2. Instruction Memory ROM
3. Control Unit ROM
4. Register file
5. 8bit ALU

In our processor, the instruction memory ROM holds the machine codes for MIPS instructions and the control unit ROM holds the necessary control bits as defined by the opcodes. The program counter register PC initially holds the address 00. When the program is initiated the first instruction is loaded into the processor which is then splitted into five parts. The first part, opcode goes to the control unit and acts as the address for ROM. Necessary control bits are generated and provided as necessary throughout the whole processor.

In case of memory instructions, i.e. lw, sw, push and pop, the data memory RAM is used to either write the calculated data or read the data to be written into the destination register of the register file.

The register file consists of 8 registers ($zero, $t0, $t1, $t2, $t3, $t4, $t5, $sp) and some necessary decoders and multiplexors to read and write data into the desired register. The register file also takes as input a 8 bit data to be written in the destination register.

For all the arithmetic and logical operations, the 8bit ALU is used which takes as input either the 8bit data read from the register for R-type instructions or a constant in case of I-type instructions. It performs the necessary operations namely addition, subtraction, logical and, logical or, logical nor, right shift and left shift. The ALU opcode feed into the ALU from control unit determines what type of operation the ALU should perform. The output of the ALU consists of an 8bit data defining the result of ALU operations and an 1bit indicator if the result is zero or not.

Various multiplexors have been used throughout the processor to select data from two possible sources. 8 bit adders have been used to calculate the address of the next instruction the PC should hold (PC is incremented by 1 in general. In branch instructions, it is incremented by 1+offset and in J-type instructions the provided address in bits in 8-15 acts as the value of PC). The MIPS have been designed to execute each type of instruction in exactly one clock pulse. The processor is manually simulated by an external clock.

We also used an external assembler (cpp program) to convert the provided MIPS assembly code to machine code.

7 Block diagrams of the main components

7.1 Instruction memory with PC

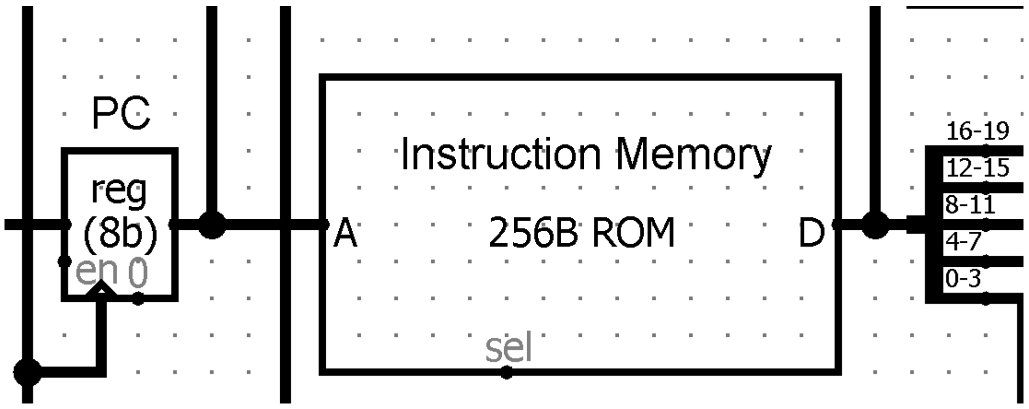


Figure 3: Instruction memory with PC

7.2 Register File

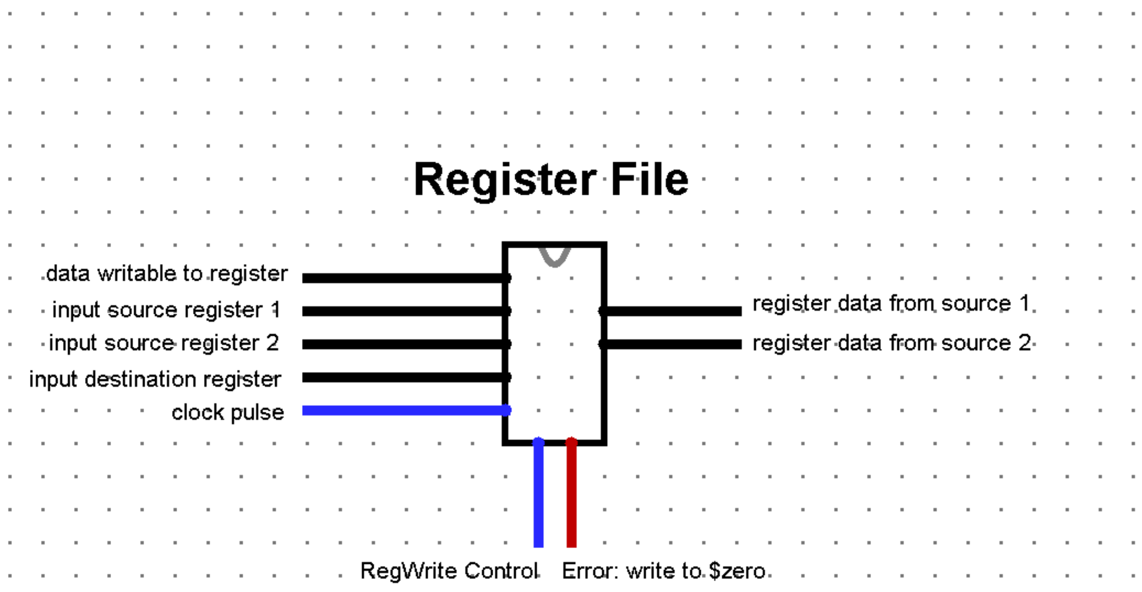


Figure 4: Register File

7.3 Data memory with the Stack

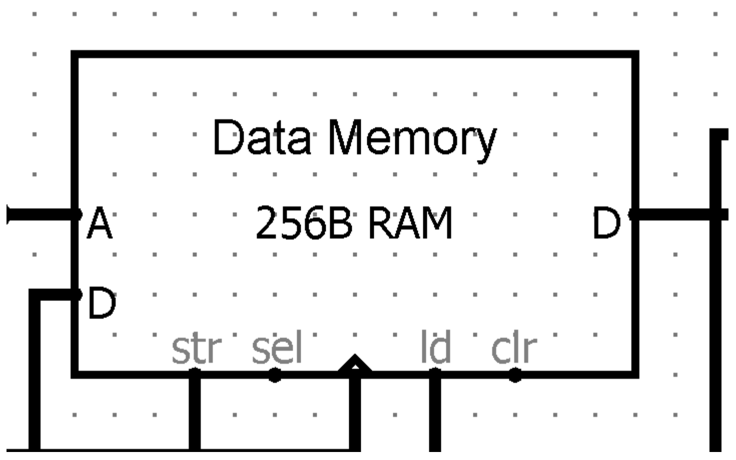


Figure 5: Data memory

7.4 Control Unit

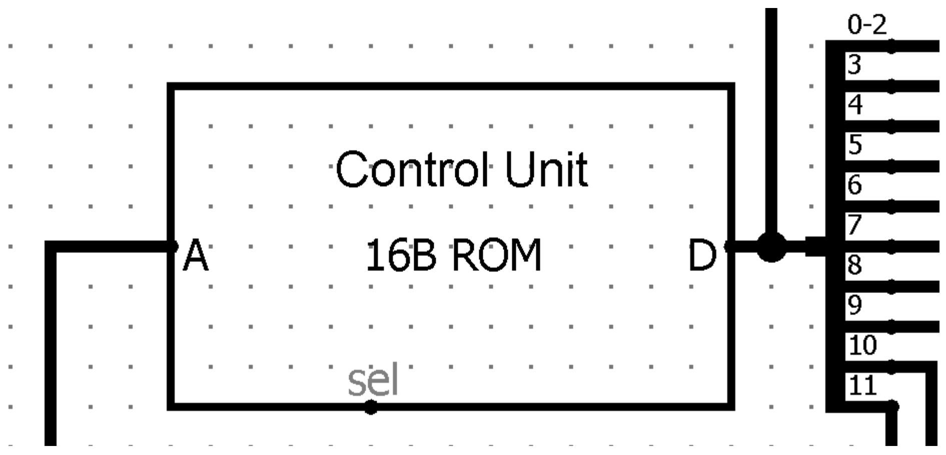
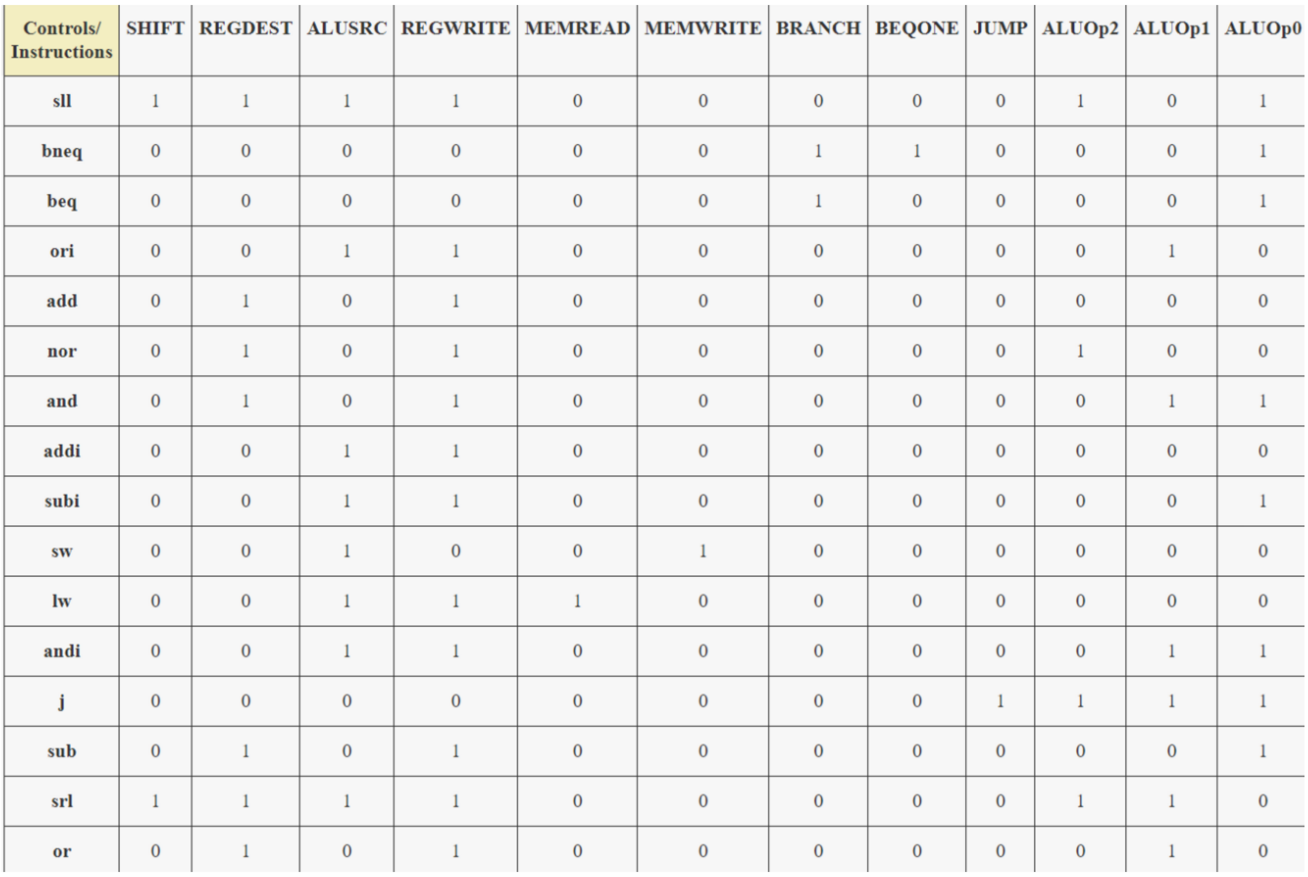


Figure 6: Control Unit

7.5 Control Unit Table



Controls/ Instructions	SHIFT	REGDEST	ALUSRC	REGWRITE	MEMREAD	MEMWRITE	BRANCH	BEQONE	JUMP	ALUOp2	ALUOp1	ALUOp0
sll	1	1	1	1	0	0	0	0	0	1	0	1
bneq	0	0	0	0	0	0	1	1	0	0	0	1
beq	0	0	0	0	0	0	1	0	0	0	0	1
ori	0	0	1	1	0	0	0	0	0	0	1	0
add	0	1	0	1	0	0	0	0	0	0	0	0
nor	0	1	0	1	0	0	0	0	0	1	0	0
and	0	1	0	1	0	0	0	0	0	0	1	1
addi	0	0	1	1	0	0	0	0	0	0	0	0
subi	0	0	1	1	0	0	0	0	0	0	0	1
sw	0	0	1	0	0	1	0	0	0	0	0	0
lw	0	0	1	1	1	0	0	0	0	0	0	0
andi	0	0	1	1	0	0	0	0	0	0	1	1
j	0	0	0	0	0	0	0	0	1	1	1	1
sub	0	1	0	1	0	0	0	0	0	0	0	1
srl	1	1	1	1	0	0	0	0	0	1	1	0
or	0	1	0	1	0	0	0	0	0	0	1	0

Figure 7: Control Unit Table

8 Description of Control Unit

Control Unit:

We have used 12 selection bits to control the entire operation of MIPS. the bits are described as following:

1. Bit 0,1,2 (ALUOP):

These 3 bits are dedicated to select how the ALU unit will perform such as: add, subtract, and, or, nor, left shift, right shift.

2. Bit 3 (Jump):

This bit indicates whether the instruction is J type or not.

3. Bit 4 (BEQONE):

If the 'Branch' bit is 1, then the instruction is a branch operation and then this bit will indicate whether it is beq or bneq instruction.

4. Bit 5 (Branch):

This bit indicates whether the instruction is Branch operation or not.

5. Bit 6 (MemWrite):

This bit enables the write selection bit of the memory for a write.

6. Bit 7 (MemRead):

This bit enables the read selection bit of the memory for a read.

7. Bit 8 (RegWrite):

This bit enables the write selection bit of the register for a write.

8. Bit 9 (ALUSrc):

This bit decides whether the 2nd input of the ALU unit is coming from the register or not. For I type instruction, this bit is 1 which indicates the 2nd input of the ALU unit is a constant.

9. Bit 10 (RegDst):

This bit decides whether SrcReg2 or DstReg will be the write register. If this bit is 1, then DstReg will be the write register.

10. Bit 11(Shift):
This bit indicates whether the instruction is Shift operation or not.

9 Push and Pop Instruction

To implement push and pop, combinations of several MIPS instructions were used. Here each syntax is described with details of the combinations of instructions.

1.**push $t0**

```
sw $t0, 0($sp)
subi $sp, $sp, 1
```

$sp holds the stack pointer. So, the value in register $to was first stored in top of the stack using address in $sp. Then, $sp was decremented by 1 to let it point to top of the stack.

2.**push 3($t0)**

```
lw $t5, 3($t0)
sw $t5, 0($sp)
subi $sp, $sp, 1
```

Here an extra register $t5 was used to load the data from the data memory that points to the address in $t0 incremented by the offset mentioned(in this case 3). Then the value in $t5 was stored in top of the stack using stack pointer in $sp. Then, stack pointer was decremented by 1 to let it point to the current top of the stack.

3.**pop $t0**

```
addi $sp, $sp, 1
lw $t0, 0($sp)
sw $zero, 0($sp)
```

Firstly, the stack pointer was decreased to let it point to the top of stack that holds the topmost data. That data was stored in the desired register $t0. Finally, 0 was written in data memory in place of the data that was just accessed.

10 IC used with Count

- IC count(MIPS)

IC	Name	Count
74157	Quad 2-line to 1-line Multiplexer	13
7483	4-bit Binary Full Adder	4
7408	Quad 2-AND gate	1
7404	Hex Inverter	1
-	Bit extender(4-bit to 8-bit)	1
-	16*12 ROM	1
-	256*20 ROM	1
-	256*8 RAM	1
74LS323	8bit Shift/Storage Register	1
Total:	-	**24**

- IC count(Register File)

IC	Name	Count
74154	4-to-16 Line Decoder	1
74150	16-to-1 Line MUX	2
7408	Quad 2-AND gate	2
74LS323	8bit Shift/Storage Register	8
Total:	-	**13**

- IC count(ALU)

IC	Name	Count
7483	4-bit Binary Full Adder	4
7408	Shifter	2
74451	8-to-1 MUX (8-bit Container)	1
74x2G08	2-input AND-gate(8-bit container)	1
74x2G32	2-input OR-gate(8-bit container)	1
74x2G02	2-input NOR-gate(8-bit container)	1
Total:	-	**10**

11 Simulator

Logisim 2.7.1

12 Discussion

In this assignment, we have implemented a 8-bit MIPS which can execute 16 different instructions. Each instruction is executed in a clock cycle.

We implemented three different types of instructions('I'-type, 'R'-type and 'J'-type) and each instruction type follows a specific instruction format which led the way of our circuit design.

While designing MIPS, we have used minimum number of ICs. We have used a 8-bit Arithmetic Logic Unit(ALU), a Register file, Ram, Rom and some logic gate. Our control unit is so robust that it can detect any particular instruction. We have tried our level best to design the data path independent of the exact instruction. Above all, we have tried to make our design as simple as possible.