CSE-306

Assignment on 8-bit MIPS Design and Simulation

Submitted by:

Group: 1

Section: B-2

Members:

Nahian Salsabil	- 1705091
Farhana Khan	- 1705100
Rittik Basak Utsha	- 1705105
Muhtasim Noor	- 1705108

Introduction

In this assignment, our goal was to design and implement an 8-bit processor that implements the MIPS instruction set. Each instruction takes 1 clock cycle to be executed. The length of the clock cycle is long enough to execute the longest instruction in the MIPS instruction set. The main components of the processor are as follows: instruction memory, data memory, register file, ALU, and a control unit.

Instruction Set

Sequence: PLIADGFBKJNCEMOH (B2-Group 1)

ALU OP:

000 – add // A+B

001 – sub // A-B

010 – or // A|B

011 – and // A & B

100 – nor // -(A|B)

101 – sll // B << A

110 – srl // A >> B

Control Unit:

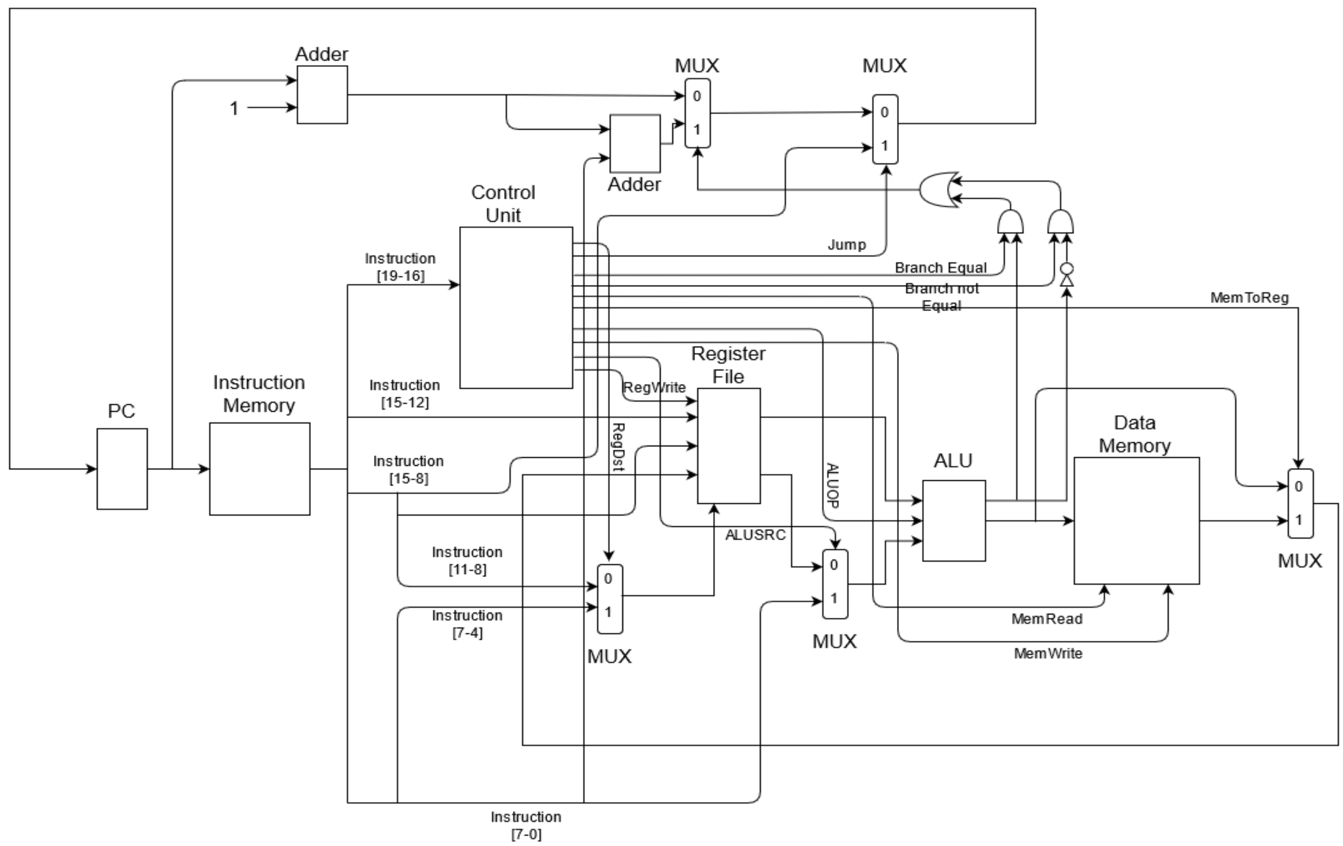
MSB…...LSB

Control Signal ->

RegDst, ALUSrc, MemToReg, RegWrite, MemRead, MemWrite, ALUop, Jump, BranchEqual, BranchNotEqual

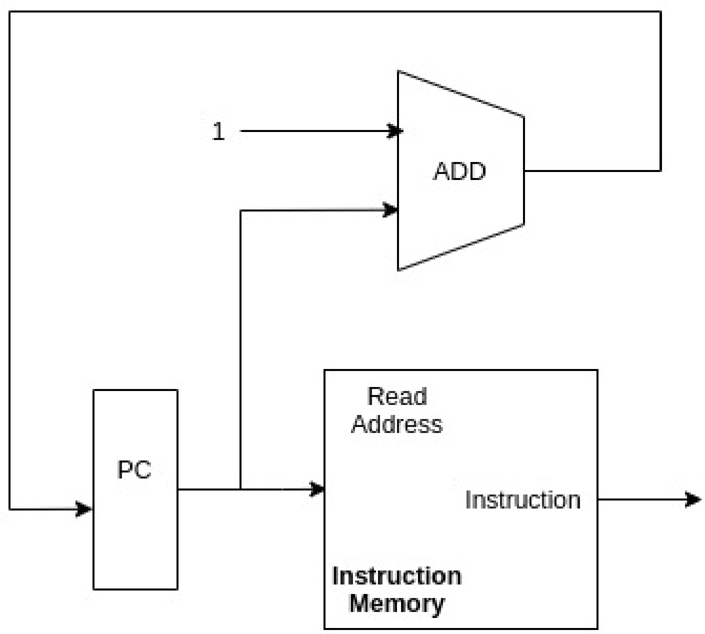
Instruction ID	Instruction Name	Instruction type	OP CODE	Control Sequence	Control sequence in hex
P	j	J	0000	000000 000 100	004
L	sw	I	0001	010001 000 000	440
I	sll	R	0010	110100 101 000	d28
A	add	R	0011	100100 000 000	900
D	subi	I	0100	010100 001 000	508
G	or	R	0101	100100 010 000	910
F	andi	I	0110	010100 011 000	518
B	addi	I	0111	010100 000 000	500
K	nor	R	1000	100100 100 000	920
J	srl	R	1001	110100 110 000	d30
N	beq	I	1010	000000 001 010	00a
C	sub	R	1011	100100 001 000	908
E	and	R	1100	100100 011 000	918
M	lw	I	1101	011110 000 000	780
O	bneq	I	1110	000000 001 001	009
H	ori	R	1111	010100 010 000	510

Complete Block Diagram

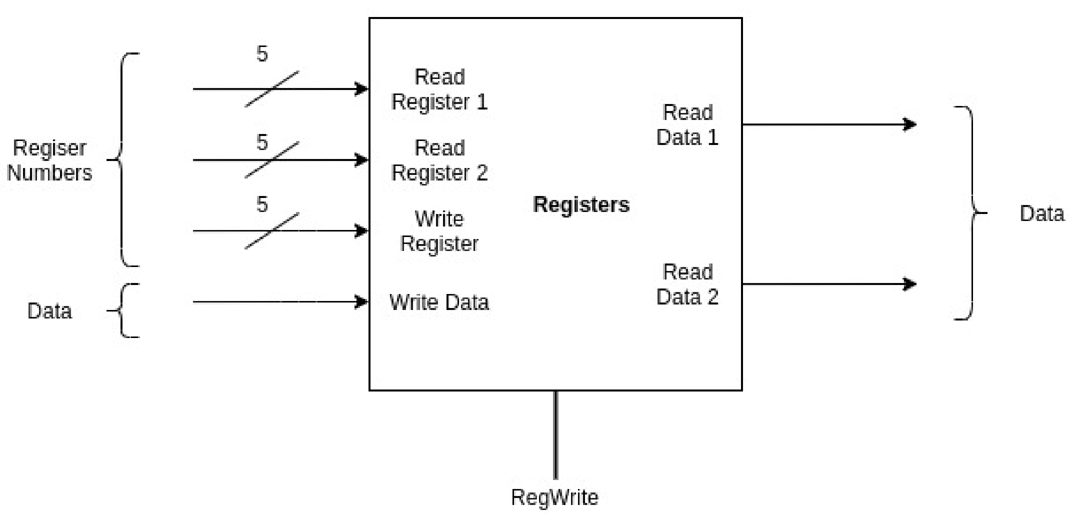


Block Diagrams of The Main Components

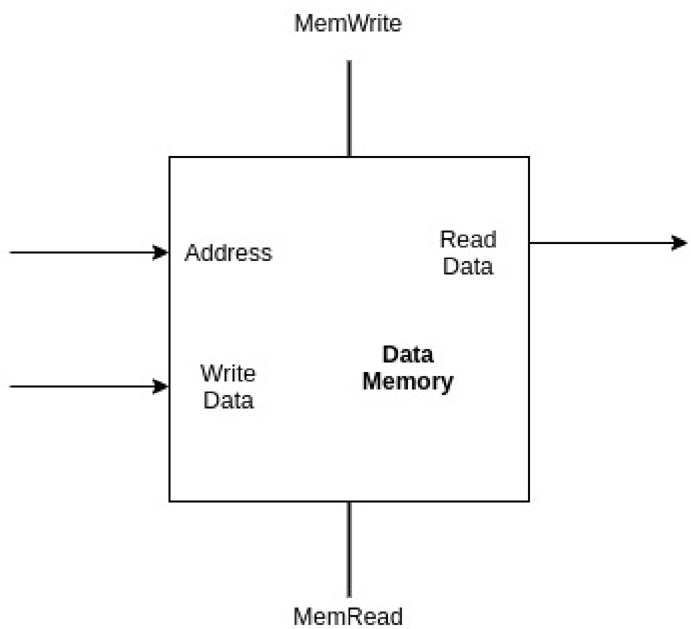
Instruction memory with PC:



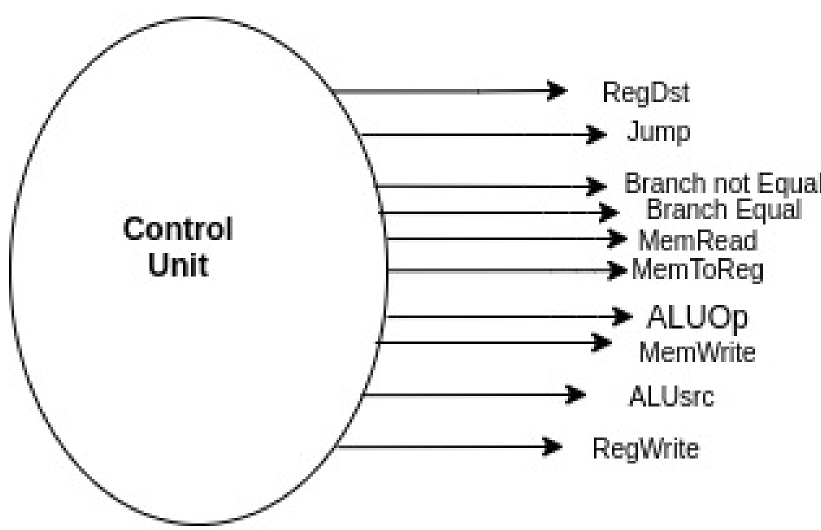
Register file:



Data memory with the Stack:



Control unit:



Approach to implement the push and pop instructions

Instruction	Description
push $t0	mem[**$sp**] = **$t0**
push 3($t0)	mem[**$sp**] = mem[**$t0**+3]
pop $t0	**$t0** =mem[**$sp**]

We processed the assembly code and replaced each push and pop statements with MIPS supported instructions, which implements the same functionality.

Modification of push $t0:

Here the value stored in the register following the push keyword is stored on the top of the stack. So, this instruction was replaced with a stack pointer decrement instruction followed by a store instruction.

e.g.

```
push $t0    =>    subi $sp, $sp, 1   //stack pointer $sp decremented by 1,
                                       since data is 8bit
                  sw $t0, 0($sp)     //value of $t0 is stored in the new
                                       location pointed by the stack pointer
```

Modification of push 3($t0):

Here the value stored in the memory location n (specified in the statement) bytes away from the memory location stored in the provided register is stored on the stack top. We came up with two strategies to handle this case.

1. to use a new register (e.g., $t5) and replace the statement by 3 statements as following:

push 3($t0) =>	subi $sp, $sp, 1	//stack pointer $sp is decremented by 1 since data is 8bit
	lw $t5, 3($t0)	// load the required value in temporary register
	sw $t5, 0($sp)	//value in the temporary register is stored in the new location pointed by the stack pointer

In this case an extra hardware is used.

2. to **not** use a separate register and instead replace the statement by 6 statements as following:

push 3($t0) =>	subi $sp, $sp, 2	// stack pointer $sp is decremented by 2, to make space for two data values
	lw $t0, 0($sp)	// save $t0 in stack

```
lw $t0, 3($t0)      // load the required value in $t0
sw $t0, 1($sp)      // value in the $t0 is stored in stack
lw $t0, 0($sp)      // restore the value of $t0 from stack
addi $sp, $sp, 1    // this increment of stack pointer will
                    remove the temporary data from stack
                    and 0($sp) will point to the location
                    where pushed data is stored
```

In this case no extra hardware is used. However, it takes 3 extra clock cycles to get the job done.

Considering this statement is not a common case, we opted for the second strategy. As a result, hardware cost and number of control bits are minimized.

Modification of pop $t0:

Here the value stored on the top of the stack is stored in the register following the push keyword. So, this instruction was replaced with a load instruction followed by a stack pointer increment instruction.

e.g.,

```
pop $t0 => lw $t0, 0($sp)       // value stored in the location pointed by
                                the stack pointer is loaded in $t0
                                register
           addi $sp, $sp, 1     //stack pointer is incremented by 1, data
                                removed from stack
```

IC Count

IC	Name	Count
7432	OR Gate	1
7408	AND Gate	1
7404	NOT Gate	1
7483	4-bit Full Adder	2
74157	2x1 MUX	2
74151	8x1 MUX	2
74299	8 bit Register	8
74138	Decoder	1
	8 bit ALU	1
	ROM	2
	RAM	1

Simulator Used

We have used "Logisim 2.7.1" to implement and simulate our design of 8 bit MIPS processor.

Discussion

The following were taken into consideration:

- A program has been written in C++ language to convert the given assembly code into MIPS machine code. This can be loaded automatically into the circuit simulator.
- The control unit has been micro-programmed. Control signals associated with the operations is stored in a separate ROM as ControlWords.

- All clocks required in the circuit is provided from a single clock source. Each instruction has been fetched and executed in a single clock cycle.
- Two separate memory units are used to implement the instruction memory (ROM) and data memory (RAM). Stack also uses the same memory component as the data memory but in reverse order and starts from the maximum address 0xFF as per specification.
- While designing, minimizing IC count and following the MIPS design principles were prioritized.