CSE306: Offline 3

4-bit MIPS Design and Simulation

Submission Date: August 17, 2022

Lab Section: B2
Group: 05

Group Members:

1705107
1805097
1805099
1805100
1805102

Introduction:

In this assignment, we are to design, simulate in software, and implement in hardware a modified and reduced version of the MIPS instruction set. Our MIPS processor is 4-bit as we are using a 4-bit ALU. There will also be a 4-bit data bus, and an 8-bit address bus. Our design will have 5 temporary registers ($t0, $t1, $t2, $t3, $t4), and two special purpose registers $zero (the zero register) and $sp (the stack pointer register).

Instruction Set:

Instruction	Category	Type	Opcode
or	Logic	R-Type	0000
subi	Arithmetic	I-Type	0001
ori	Logic	I-Type	0010
sub	Arithmetic	R-Type	0011
add	Arithmetic	R-Type	0100
lw	Memory	I-Type	0101
j	Control-Unconditional	J-Type	0110
sll	Logic	I-Type	0111
sw	Memory	I-Type	1000
addi	Arithmetic	I-Type	1001
srl	Logic	I-Type	1010
beq	Jump-Conditional	I-Type	1011
nor	Logic	R-Type	1100
and	Logic	R-Type	1101
andi	Logic	I-Type	1110
bneq	Jump-Conditional	I-Type	1111

Circuit Diagram:

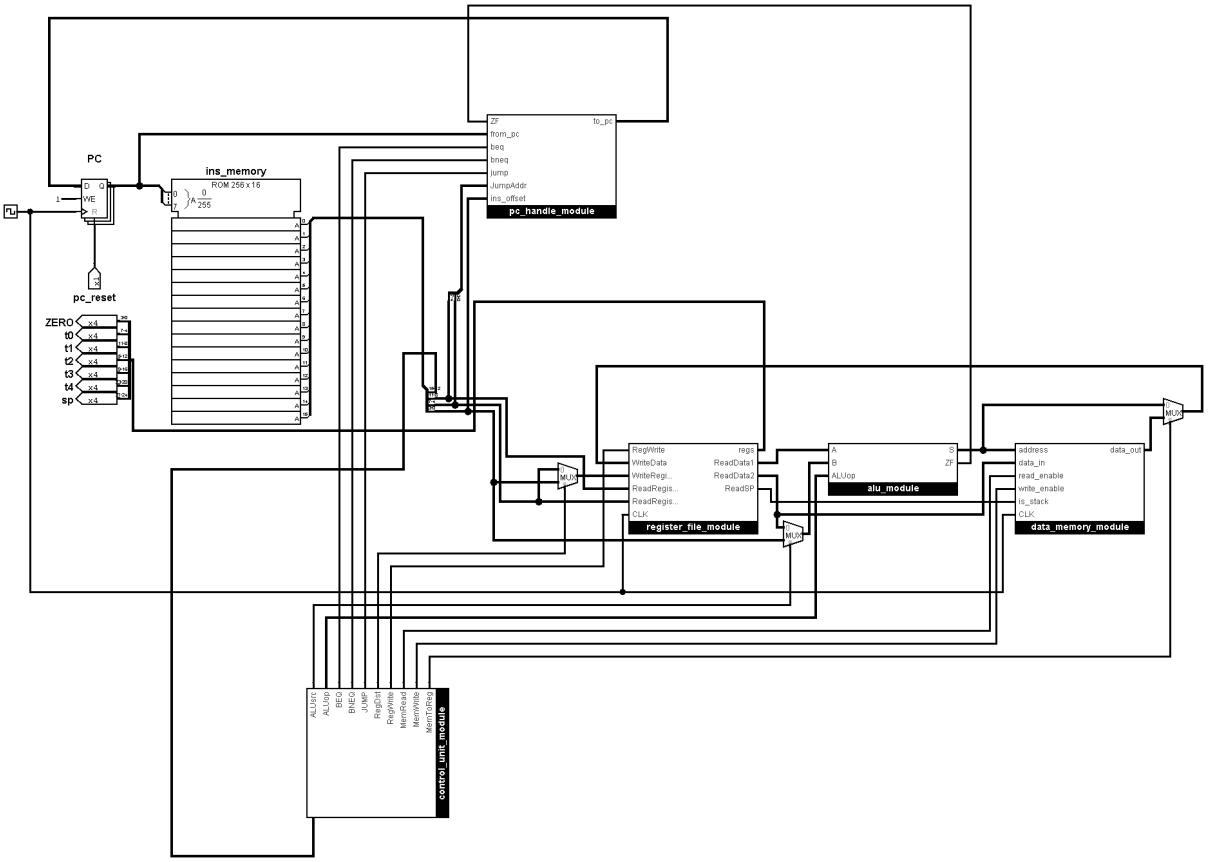

Figure: 4-bit MIPS Processor

How to Write and Execute a Program on this Processor:

1. First of all, we have to write MIPS instructions for the program in a text file.
2. Next, we produce the machine code for our program by feeding the MIPS source file to our custom assembler.
3. Finally the generated machine code is loaded into the instruction memory, and the processor is ready to execute the program.

Control Unit Signals:

Opcode	Reg Dst	ALU Src	Mem toReg	Reg Write	Me Read	Me Write	Branch Eq	Branch Neq	Jump	ALUop
0000	1	0	0	1	0	0	0	0	0	011
0001	0	1	0	1	0	0	0	0	0	001
0010	0	1	0	1	0	0	0	0	0	011
0011	1	0	0	1	0	0	0	0	0	001
0100	1	0	0	1	0	0	0	0	0	000
0101	0	1	1	1	1	0	0	0	0	000
0110	0	0	0	0	0	0	0	0	1	000
0111	0	1	0	1	0	0	0	0	0	101
1000	0	1	0	0	0	1	0	0	0	000
1001	0	1	0	1	0	0	0	0	0	000
1010	0	1	0	1	0	0	0	0	0	110
1011	0	0	0	0	0	0	1	0	0	001
1100	1	0	0	1	0	0	0	0	0	100
1101	1	0	0	1	0	0	0	0	0	010
1110	0	1	0	1	0	0	0	0	0	010
1111	0	0	0	0	0	0	0	1	0	001

ALU Operations:

ALU Opcode	Operation
000	A + B
001	A - B
010	A AND B
011	A OR B
100	A NOR B
101	A << B
110	A >> B
111	NO-OP

Special Feature:

We have additionally implemented the stack memory. We can interact with the stack memory via two special instructions: push and pop.

push $t0 instruction is replaced by the following instructions:

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

pop $t0 instruction is replaced by the following instructions:

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

ICs Used with their Count:

Component	Count
Adder	3
Subtractor	1
Logical Left Shifter	1
Logical Right Shifter	1
AND Gate	6
OR Gate	3
NOR Gate	3
2x1 MUX	6
8x1 MUX	3
16x12 ROM	1
3x8 DEMUX	1
4-bit Register	7
NOT Gate	3
Priority Encoder	1
4x1 MUX	1
16x4 RAM	2
8-bit Register	1

Discussion:

We have used a ROM in the control unit. Instead of using a combinational circuit for the control unit, we precalculated all the possible input-output combinations for the control unit and loaded the result in a ROM. As for the ALU, we have performed all the operations and selected the result based on the ALUOp. Designing and implementing a combinatorial equivalent version of the ALU and the control unit would have been a tedious task, so we opted for the easier implementation.

For implementing the stack memory, we have added an additional register called $sp which is used as the stack pointer.