CSE 306
Computer Architecture Sessional

8-bit MIPS Design and Simulation

Group: 02
Section: A2

Group Members:
1705036
1705037
1705038
1705039
1705044

1 Introduction

MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Computer Systems, now MIPS Technologies, based in the United States.

In this assignment, we have to design an 8-bit processor that implements the MIPS instruction set. Each instruction will take 1 clock cycle to be executed. The length of the clock cycle will be 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.

Our MIPS Instructions will be 20-bits long with the following three formats:

- **R-type**

Opcode	Src Reg 1	Src Reg 2	Dst Reg	Shft Amnt
4-bits	4-bits	4-bits	4-bits	4-bits

- **I-type**

Opcode	Src Reg	Dst Reg	Address / Immediate
4-bits	4-bits	4-bits	8-bits

- **J-type**

Opcode	Target Jump Address	0	0
4-bits	8-bits	4-bits	4-bits

In order to execute the instructions, the processor has to follow a series of steps. For every instruction (except unconditional jump), 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.

After these two steps, the actions required to complete the instruction depend on the instruction class. Fortunately, for each of the three instruction classes (memory-reference, arithmetic-logical, and branches), the actions are largely the same, independent of the exact instruction. The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.

For example, all instruction classes, except jump, use the arithmetic-logical unit (ALU) after reading the registers. The memory-reference instructions use the ALU for an address calculation, the arithmetic-logical instructions for the operation execution, and branches for comparison. After using the ALU, the actions required to complete various instruction classes differ. A memory-reference instruction will need to access the memory either to read data for a load or write data for a store. An arithmetic-logical or load instruction must write the data from the ALU or memory back into a register. Lastly, for a branch instruction, we may need to change the next instruction address based on the comparison; otherwise, the PC should be incremented by 1 to get the address of the next instruction. The jump instruction looks somewhat like a branch instruction but computes the target PC differently and is not conditional.

Finally, we have to implement push and pop operations in the MIPS design for using a stack.

2 Instruction Set

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

3 Complete Block diagram of an 8-bit MIPS processor

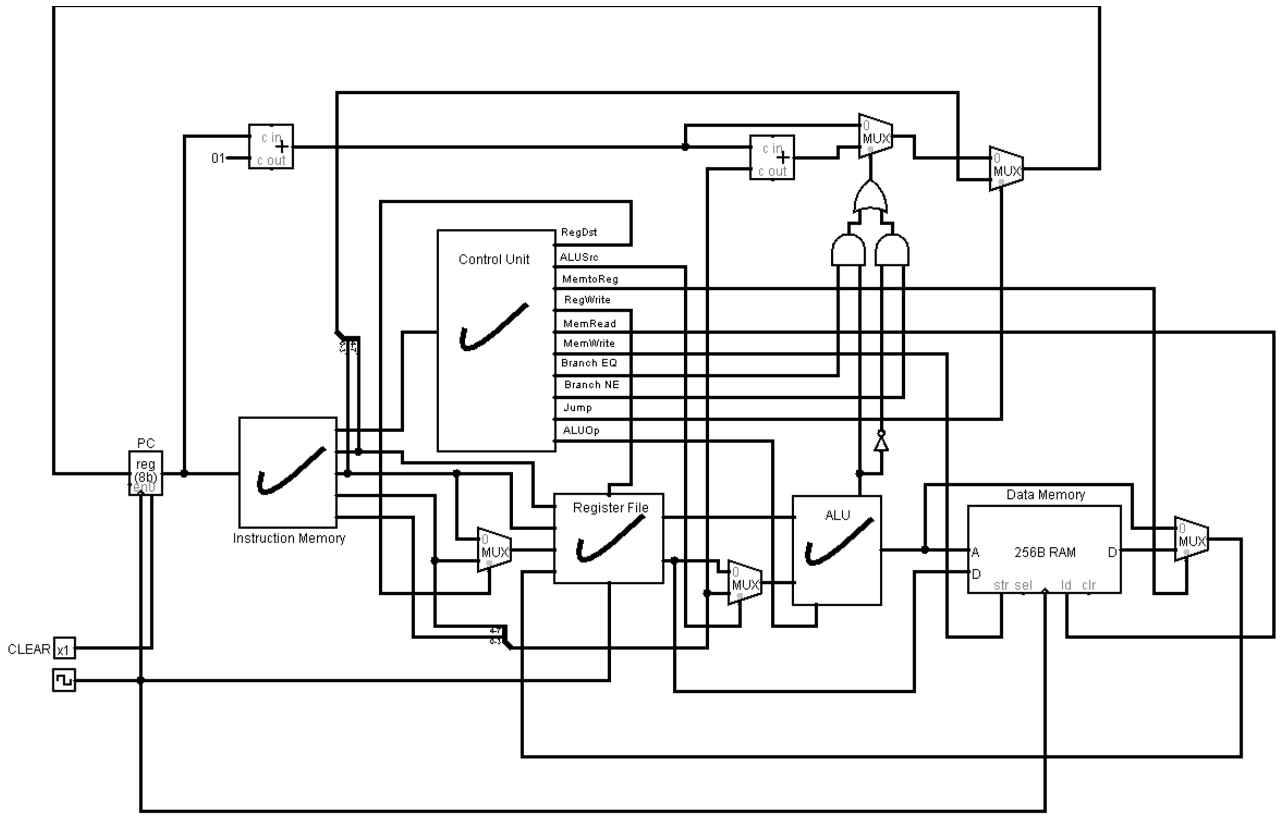


Figure 1: 8-bit MIPS processor

4 Block diagrams of the main components

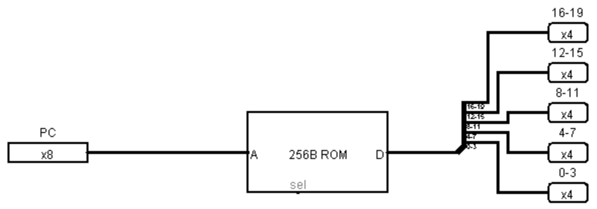


Figure 2: Instruction Memory

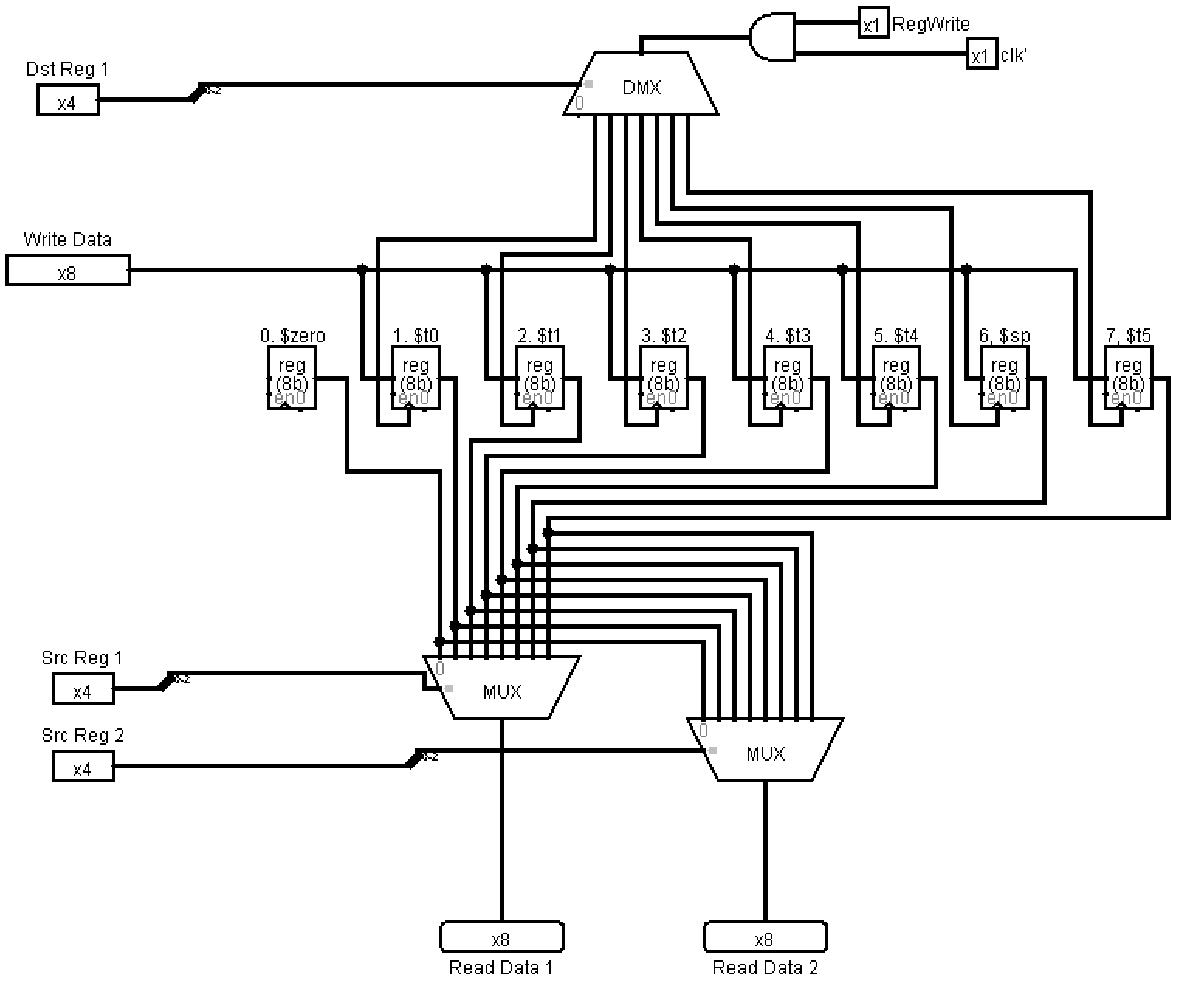


Figure 3: Register File

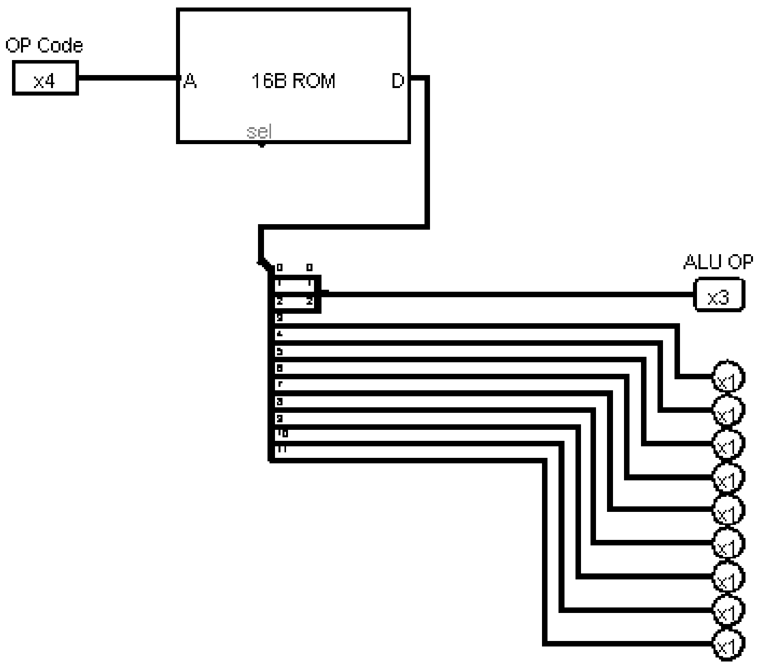


Figure 4: Control Unit

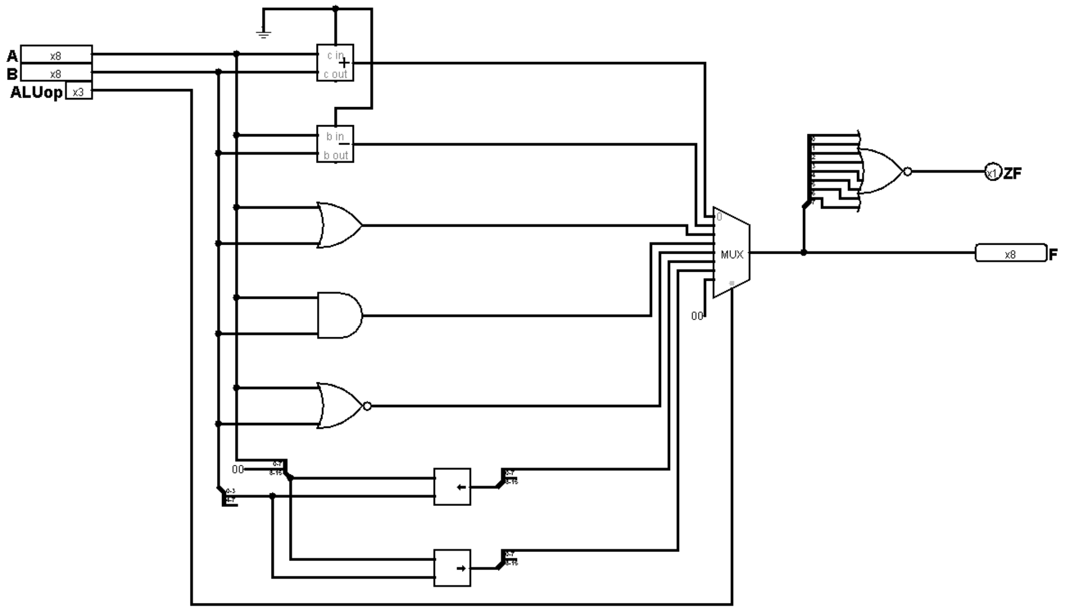
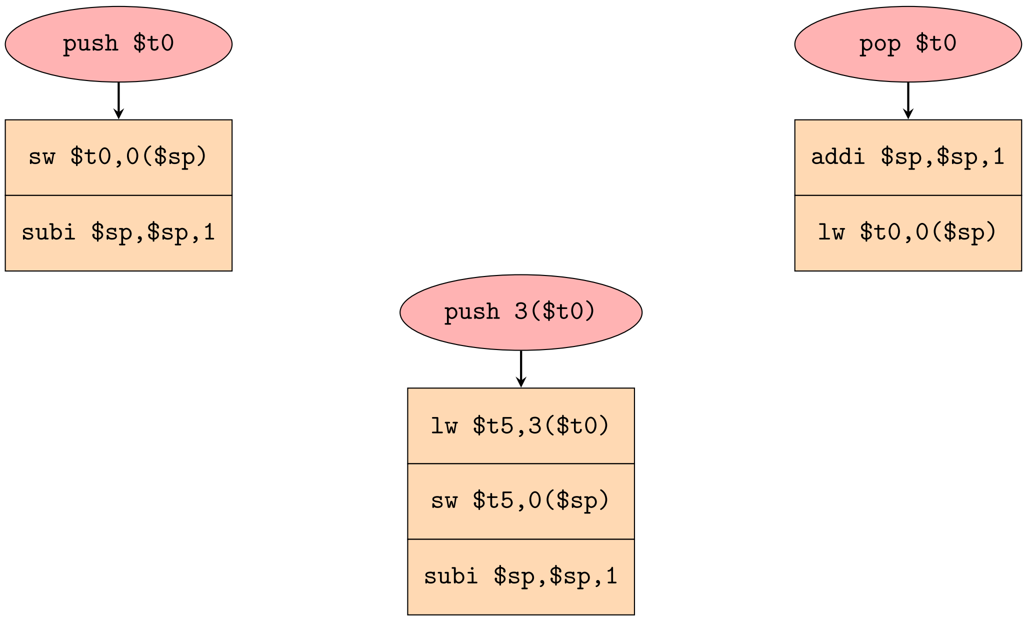


Figure 5: ALU

5 Approach to implement the push and pop instructions

To incorporate the operations of a stack into our assignment, we converted each push and pop instruction into appropriate load and store instructions. We also incremented value of stack pointer when a pop operation occurred and decremented the same for a push operation. This entire process was handled programmatically with the script to convert the assembly code into MIPS machine code. A temporary register (t5) was also added in the register file for dealing with the push operation that transfers data from memory to stack.



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

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

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

6 ICs used with their count

IC Name	**Count**
IC 7404	1
IC 7408	1
IC 7432	1
IC 7402	1
IC 7483	8
IC 7486	2
IC 74151	3
IC 74157	2
IC 74238	1
IC 74HC4078	1

Component Name	**Count**
Register	9
ROM	2
Shifter	2
RAM	1

7 Discussion

In this assignment, we had to design and simulate an 8-bit processor of the MIPS architecture with a single cycle instruction set. The required design consisted of many components, all of which we had to implement and integrate to complete the processor.

Each instruction had a specific format with a corresponding opcode value. The parsing of each assembly instruction to specific binary encoding and the calculation of the control signals were done very carefully. The proper procedure of implementing each instruction of three different formats was meticulously followed. We also applied PC-relative addressing for branching instructions and Pseudo-direct addressing for jump instructions.

As our MIPS design was of a 20-bit instruction format whereas regular MIPS instructions are 32-bits, a few differences in the calculations were noticed while designing. A clock pulse was provided to each component for writing and moving to the next instruction. Since this was a single cycle implementation, the pulse width of the clock might be large. We also had to simulate the operations of a stack, but this could not be done in a single clock pulse. A maximum of three clock pulses was needed for the push operation of the stack. Finally, we tested and simulated the circuit with various test cases. The design was made as efficient as possible.

8 Software Version

Logisim-Win-2.7.1