CSE 306

COMPUTER ARCHITECTURE SEASIONAL

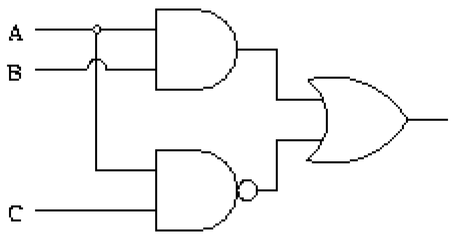
Assignment 3 : 8-bit MIPS Design and Simulation

Section : A2

Group No : 03

Student ID:
1.1605051
2.1705040
3.1705042
4.1705048
5.1705054

Submission Date: 20.06.2021



Introduction

MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA). MIPS only takes register values as operands for arithmetic operations. MIPS has thirty-two 32-bit general-purpose registers (GPR). Register $zero is hardwired to zero and writes to it are discarded. The program counter has 32 bits. Instructions are divided into three types: R, I and J. Every instruction starts with a 6-bit opcode. In addition to the opcode, R-type instructions specify three registers, a shift amount field, and a function field; I-type instructions specify two registers and a 16-bit immediate value; J-type instructions follow the opcode with a 26-bit jump target.

In this assignment, we designed an 8-bit processor that implements the MIPS instruction set.Here, 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 our 8-bit processor are as follows: instruction memory, data memory, register file, ALU, and a control unit.To simulate our design, we needed to use an assembly code. Before starting the simulation, we converted the given assembly code into MIPS machine code and loaded the machine code into the instruction memory.

Problem Specification

In this assignment, we were required to implement an 8-bit processor that implements add, addi, sub, subi, and, andi, or, ori, sll, srl, nor, sw, lw, beq, bneq, j with the “FDNKLIMJEOPBGCHA” sequence of opcodes. Also, we were required to design push and pop operations in our MIPS design using a stack. Our MIPS instructions were 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

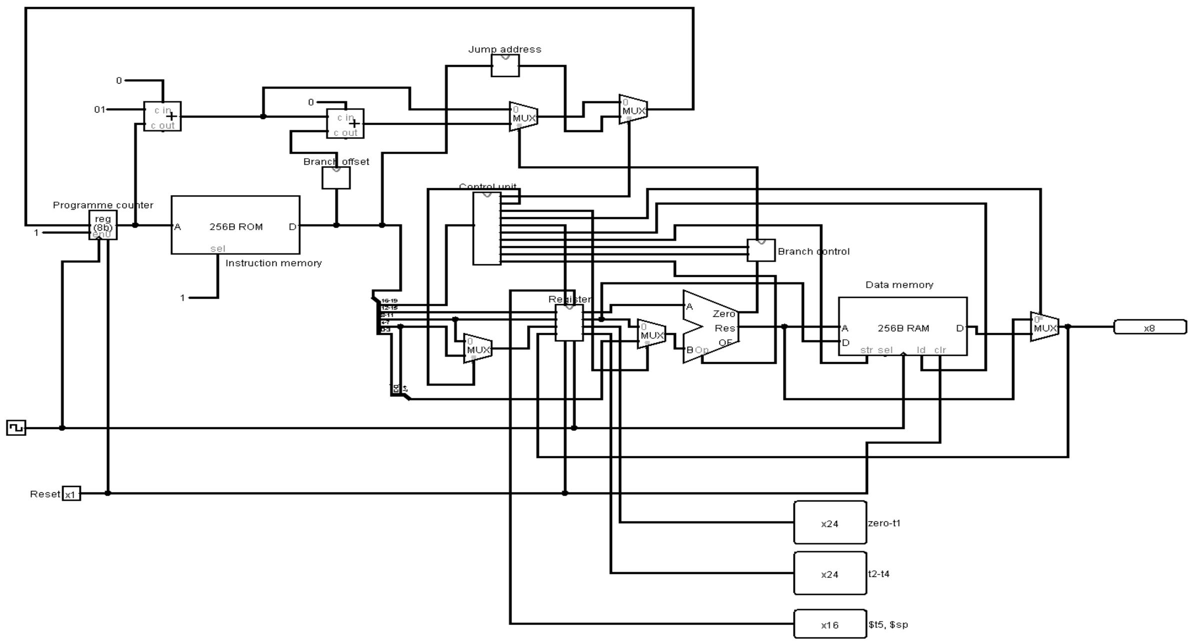
- J-type

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

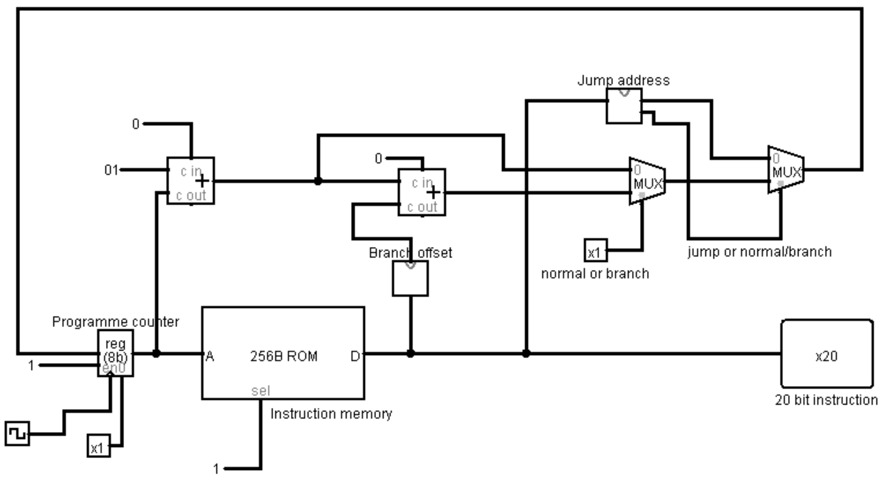
INSTRUCTION SET DESCRIPTION

Instruction ID	Category	Type	Instruction
A	Arithmetic	R	add
B	Arithmetic	I	addi
C	Arithmetic	R	sub
D	Arithmetic	I	subi
E	Logic	R	and
F	Logic	I	andi
G	Logic	R	or
H	Logic	I	ori
I	Logic	R	sll
J	Logic	R	srl
K	Logic	R	nor
L	Memory	I	sw
M	Memory	I	lw
N	Control-conditional	I	beq
O	Control-conditional	I	bneq
P	Control-unconditional	J	j

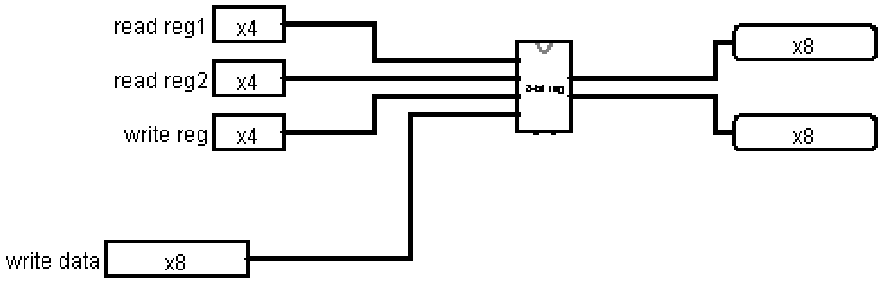
High Level Block Diagram



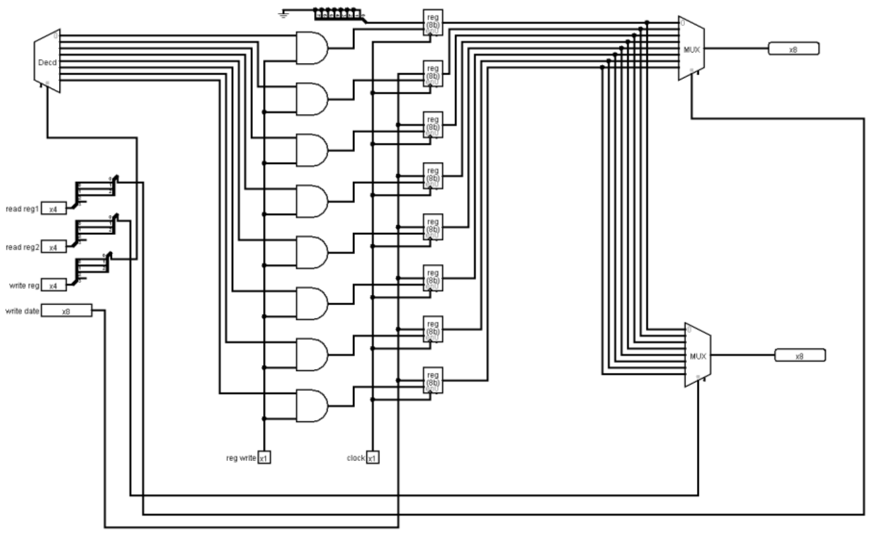
Instruction memory with PC



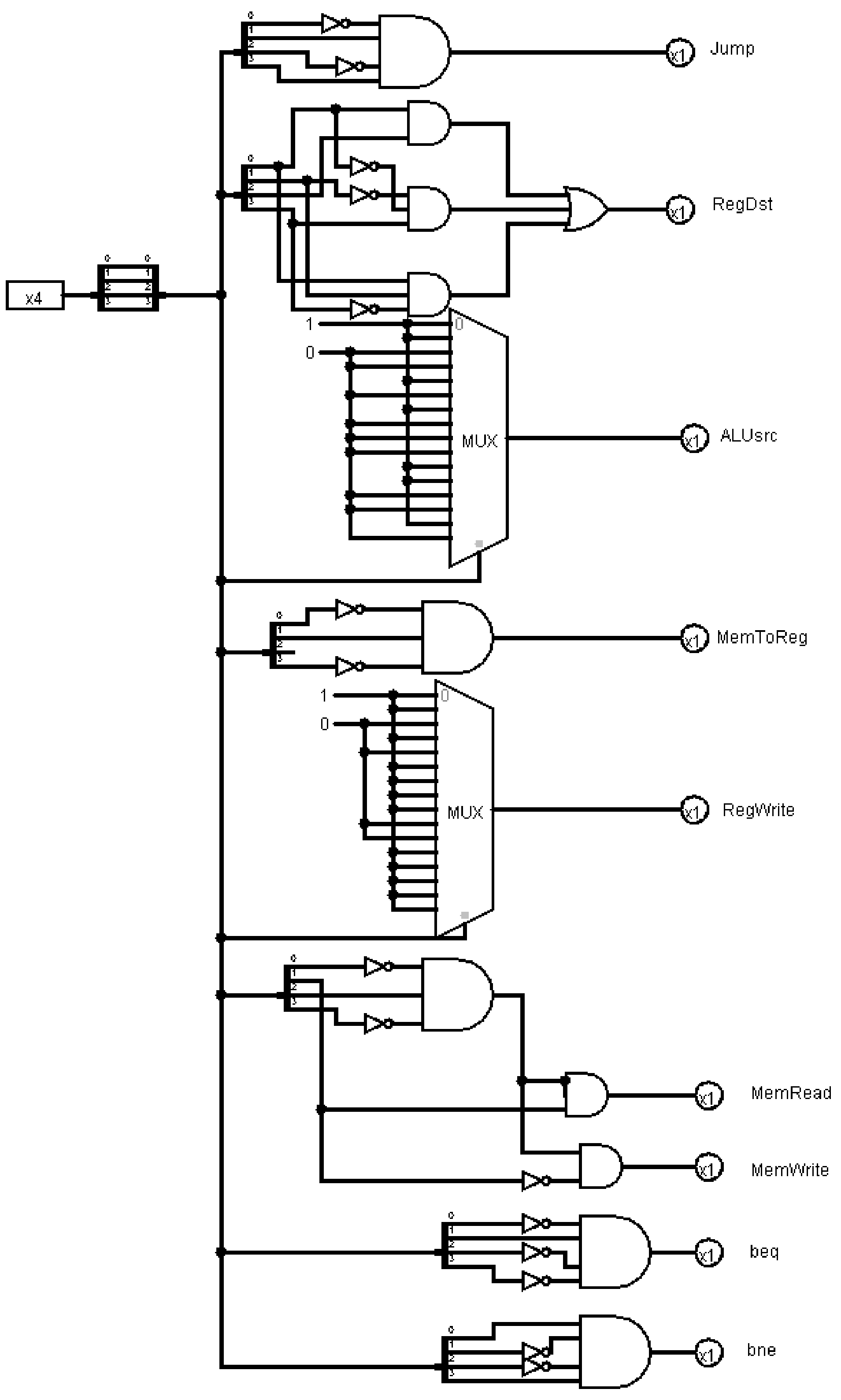
Register File



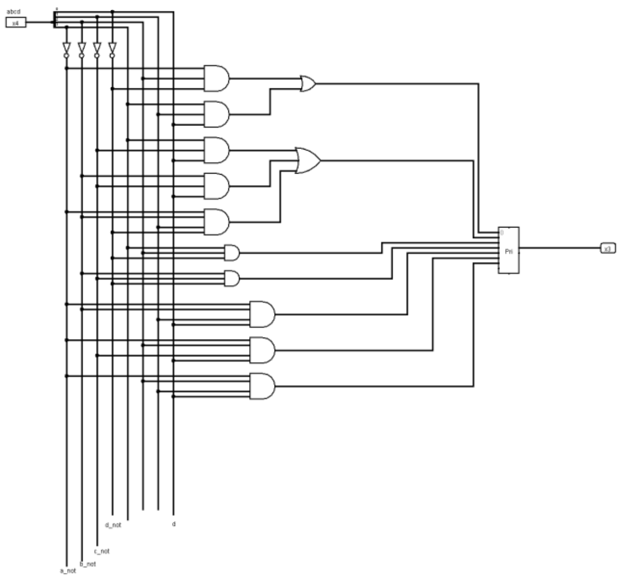
Register File Circuit



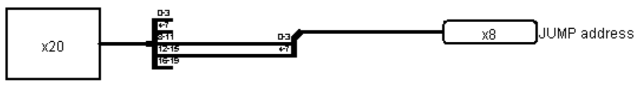
Control Unit



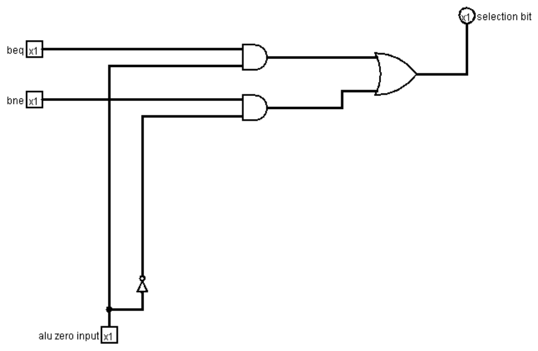
ALU Control Unit



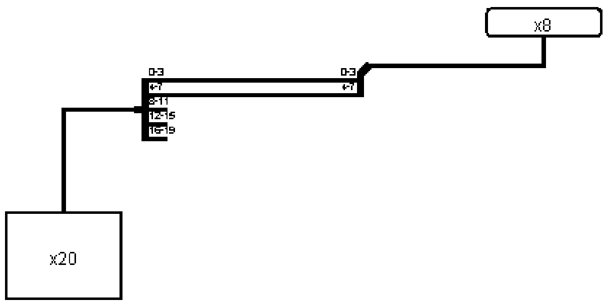
Jump Address



Branch Selector



Branch Offset



Approach To Implement Push Pop

For implementing push instruction , we considered two cases –
case -01: push 0($t0)

```
corresponding assembly instructions ,
lw $t5, 2($t0)
sw $t5, 0($sp)
subi $sp, $sp, 1
```

case -02: push $t0

```
corresponding assembly instructions ,
sw $t0, 0($sp)
subi $sp, $sp, 1
```

To implement push instruction , we performed load, store and subtract immediate operations respectively.
Accordingly, we used store and add immediate operations for implementing pop instruction.

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

Required ICs And Components

IC Number	IC Type	Count
7408	**quad 2 input AND gate**	**29**
7432	**quad 2 input OR gate**	**3**
7404	**hex NOT gate**	**19**
74148	**8*3 priority encoder**	**1**
74138	**3*8 decoder**	**1**
7480	**1 bit adder**	**8**
74151	**8*1 MUX**	**2**
74150	**16*1 MUX**	**2**
	8-bit register	**9**
	ROM	**1**
	RAM	**1**

Simulator Used

Logisim-generic: 2.7.1

Discussion

In this assignment, we designed an 8-bit processor that implements the MIPS instruction set using AND gate, OR gate, NOT gate, Adder, Priority Encoder, Multiplexer, Decoder, Register and ROM. We counted the IC number according to the ICs available in the real world. For initializing stack pointer $sp we have set a clock pulse. We used separate instruction and data memories as we can not access both the data and instruction memory at the same time. Memory is actually divided between instruction and data in a single-cycle machine as memory can only be accessed once per cycle.