CSE 306
Computer Architecture Sessional

Assignment-3: 4-bit MIPS Design, Simulation, and Implementation

Lab Section - A1
Group - 06

11 February, 2024

Members of the Group:

i. 2005020 - Mostafa Rifat Tazwar

ii. 2005025 - Most. Sonia Khatun

iii. 2005027 - Swastika Pandit

iv. 2005029 - MD. Minhajul Islam Fuad

v. 2005030 - Fairuz Mubashwera

1 Introduction

MIPS (Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies. Instructions of MIPS are fixed, thus ensuring regularity.
Here is an example of sub instruction

Operation	Instruction	Action
Subtraction	sub $t0, $t1, $t2	$t0 = $t1 - $t2

Here $t1, $t2, $t3 are registers that hold values. To evaluate an expression p = a-b–c, we would do the following.
sub $t0, $t1, $t2 [p = a-b]
sub $t0, $t0, $t3 [p = p-c or p = a-b-c]

The prime objective of this assignment was to design an 4-bit processor which implements the MIPS instruction set.It took 1 clock cycle to execute each instruction. The length of the clock cycle was long enough so that the longest instruction on the MIPS instruction set was executed in a single clock cycle. Instruction memory, Data memory, Register file, ALU, Control unit were the main components of the processor. These components were designed and properly connected using some available multiplexers, adders and wires.

2 Instruction Set

Decimal (Opcode)	Instruction ID	Category	Type	Instruction
0(0000)	E	Logic	R	and
1(0001)	F	Logic	I	andi
2(0010)	N	Control-conditional	I	beq
3(0011)	P	Control-unconditional	J	j
4(0100)	O	Control-conditional	I	bneq
5(0101)	C	Arithmetic	R	sub
6(0110)	D	Arithmetic	I	subi
7(0111)	B	Arithmetic	I	addi
8(1000)	I	Logic	S	sll
9(1001)	G	Logic	R	or
10(1010)	A	Arithmetic	R	add
11(1011)	H	Logic	I	ori
12(1100)	M	Memory	I	lw
13(1101)	J	Logic	S	srl
14(1110)	K	Logic	R	nor
15(1111)	L	Memory	I	sw

Table 1: Instruction set

3 Description Of Modules

3.1 Instruction Memory with Program Count

Instruction Memory is the starting fundamental component of our MIPS. The instruction memory circuit takes an 8-bit program count (PC) and gives 16-bit instruction for the PC. Program Count is calculated using 8-bit binary adders. A ROM is used to store all possible 256 instructions and the circuit fetches the intended instruction for a particular program count. The instruction (output) is then split into 4-bit nibbles, for example, instruction bits [15:12] mean Opcode.There is a MUX in the circuit to select which 4-bit nibble will be used as the address for the destination register, and the MUX is controlled by RegDst control bit which comes from Control Unit. The circuit works in synchronization with clock signals.

3.2 Data Memory With Stack

Data Memory is the part of circuit that is used when we want to load word from memory element or store word in memory element. As input we give the 4 bit ALU output and 4 bit reg data which is to be written on a particular memory when we give store word command. A RAM is used as memory element as there will be frequent read and write operation on this memory. So there are also three control inputs which will come from the control unit. If memRead is 1, then the data from the given memory address will be given as output.If memWrite is 1, then the data will be written in given memory address. There is also a MUX to select whether the output of the ALU (for R-format) or the output of Data memory(for S format) will finally go as an output. MemToReg is the selection bit for this mux. If it is 1, output of the Data memory is selected and then the output goes to the register for write.The memory element also serves as the stack, where initially the end address of the memory signifies the top of the stack and the stack pointer handles the relevant operations.

3.3 ALU

An Arithmetic Logic Unit (ALU) is a fundamental component of a computer's central processing unit (CPU). It's responsible for performing arithmetic and bitwise logical operations on input data from registers or memory. In this assignment , the circuit takes two operands which are 4 bit number and an opcode which decides what type of operation is done by the ALU. After performing operation , the circuit produces a 4 bit output.

Opcode	Function
000	ADD
001	SUB
010	OR
011	AND
100	NOR
101	SLL
110	SRL

Table 2: ALU Opcode

3.4 Register File

This circuit takes as input two register addresses, ReadReg1 and ReadReg2 and outputs the associated data in those addresses of register file, namely ReadData1 and ReadData2.
For write operation it takes four inputs, WriteData (the data to be written), WriteReg (the address of the register in which the data will be written), RegWriteEnable (whether this data should be written), and clock. WriteReg is input to a 3X8 decoder. When register x is selected, $RegSelect_x$ is 1. When $RegSelect_x$ is 1 and RegWriteEnable is 1, the given WriteData is input to that register, otherwise, the data previously stored in that register, Q_n is input to that register.

$$Q_{x(n+1)} = WriteEnable.RegSelect_x.WriteData + \overline{WriteEnable.RegSelect_x}.Q_{x(n)}$$

3.5 Control Unit

The control unit takes 4 bit Opcode as input and sets the relevant control bits for *R, S, I and J-type* instructions according to the following table.

Ins	**Opcode**	**ALUOp**	**Branch**	**Jump**	**Bneq**	**Mem** to Reg	**Mem** Read	**Mem** Write	**Reg** Write	Reg Dst	ALU Src
PORTS	$D_3D_2D_1D_0$	$A_7A_6A_5$	A_4	A_3	A_2	A_1	A_0	D_7	D_6	D_5	D_4
and	0000	011	0	0	0	0	0	0	1	1	0
andi	0001	011	0	0	0	0	0	0	1	0	1
beq	0010	001	1	0	0	0	0	0	0	0	0
j	0011	111	0	1	0	0	0	0	0	0	0
bneq	0100	001	1	0	1	0	0	0	0	0	0
sub	0101	001	0	0	0	0	0	0	1	1	0
subi	0110	001	0	0	0	0	0	0	1	0	1
addi	0111	000	0	0	0	0	0	0	1	0	1
sll	1000	101	0	0	0	0	0	0	1	0	1
or	1001	010	0	0	0	0	0	0	1	1	0
add	1010	000	0	0	0	0	0	0	1	1	0
ori	1011	010	0	0	0	0	0	0	1	0	1
lw	1100	000	0	0	0	1	1	0	1	0	1
srl	1101	110	0	0	0	0	0	0	1	0	1
nor	1110	100	0	0	0	0	0	0	1	1	0
sw	1111	000	0	0	0	0	0	1	0	0	1

Table 3: Control Bits For Opcodes

Opcode (Binary)	**Control (Hex Codes)**
0000	0x606
0001	0x605
0010	0x300
0011	0xe80
0100	0x340
0101	0x206
0110	0x205
0111	0x005
1000	0xa05
1001	0x406
1010	0x006
1011	0x405
1100	0x035
1101	0xc05
1110	0x806
1111	0x009

Table 4: Hex Code of Control bits

4 Circuit

4.1 Main Circuit:

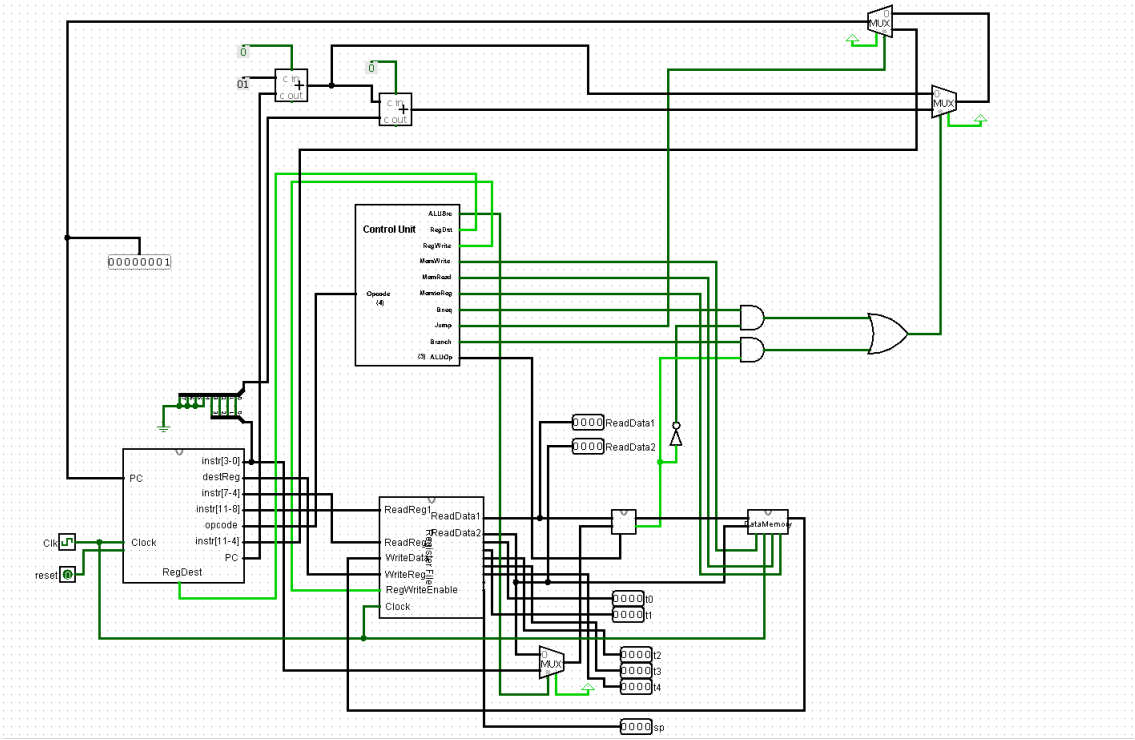

Figure 1: **4 Bit MIPS Processor**

4.2 Detailed Circuit:

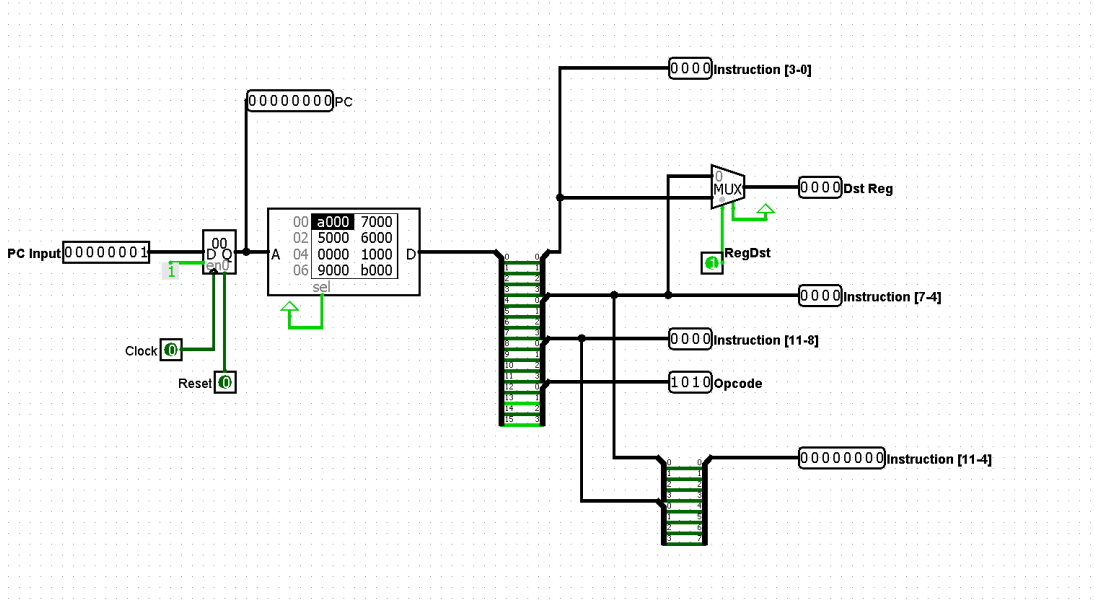

Figure 2: **Instruction Memory**

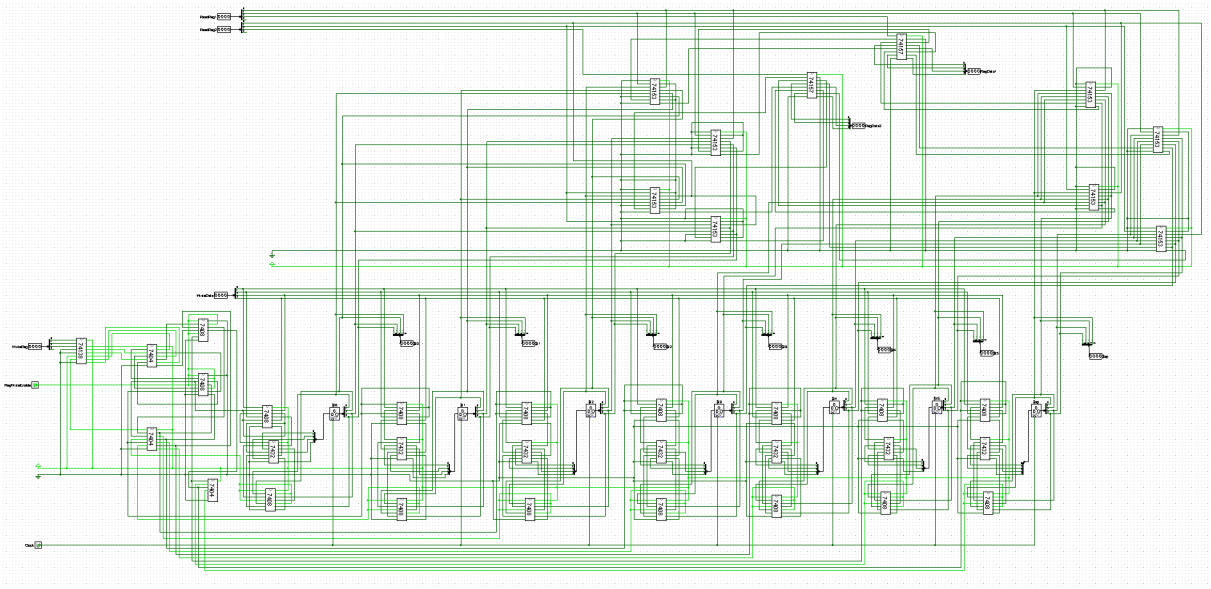

Figure 3: **Register File**

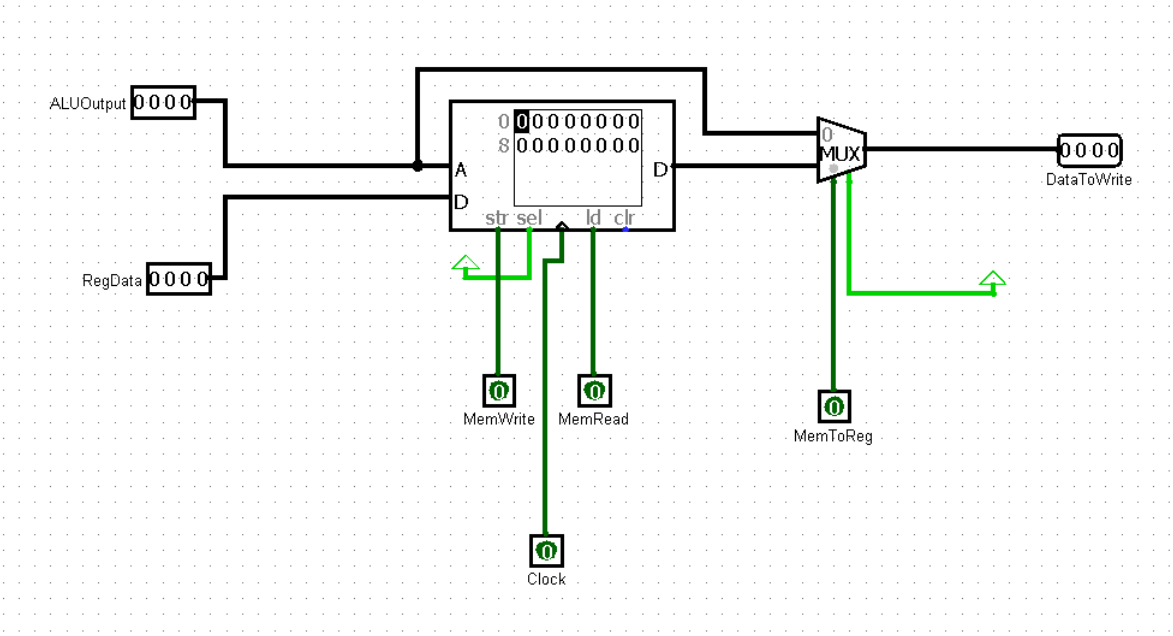

Figure 4: **Data Memory**

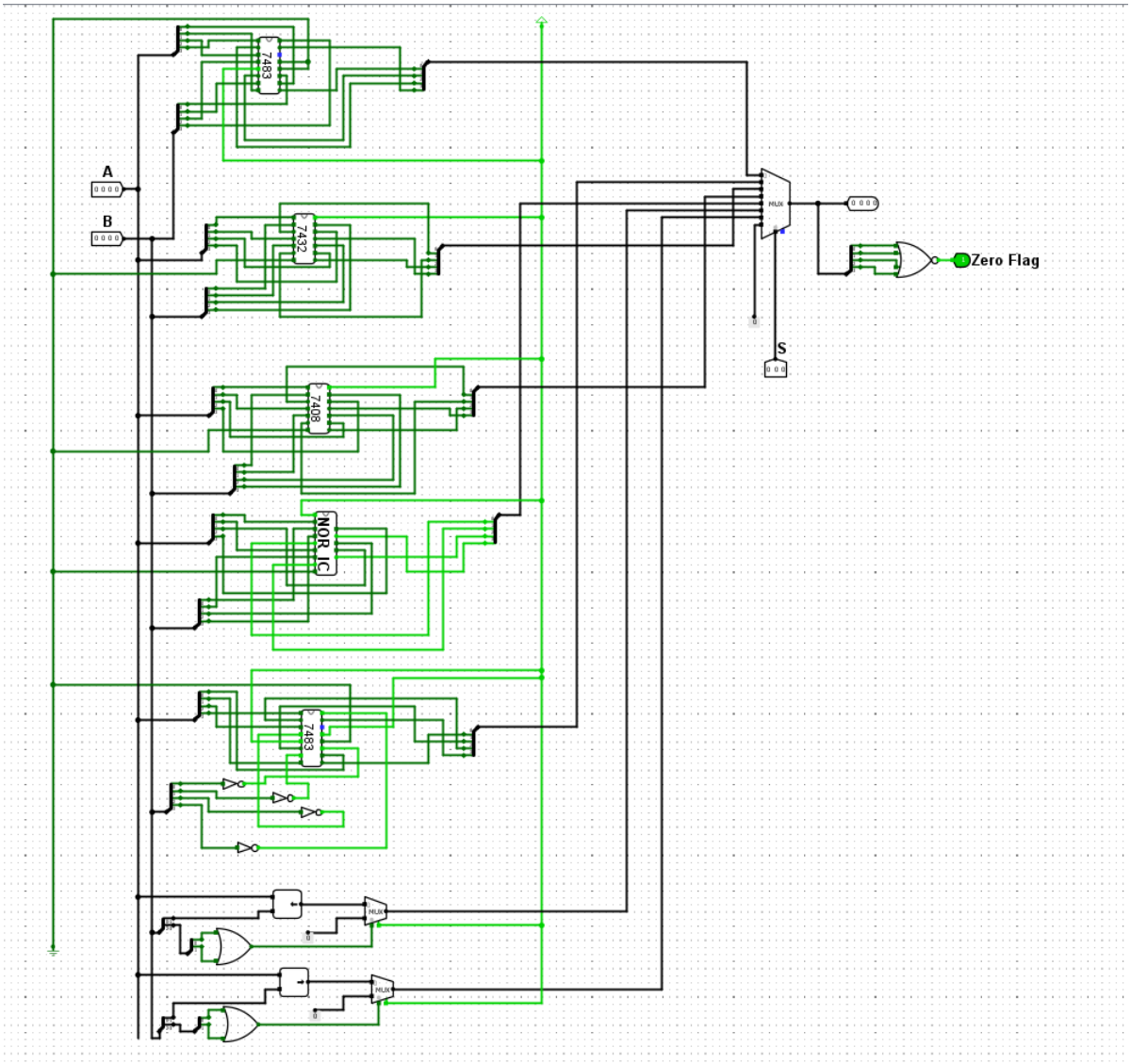

Figure 5: **ALU**

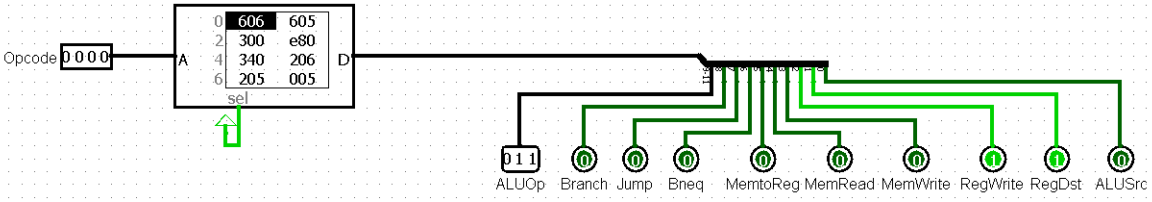

Figure 6: **Control Unit**

5 Approach to implement the push and pop instructions

We maintained a stack memory in Data Memory Unit and a stack pointer in the register file. MIPS follows the RISC architecture. So, there were no direct instructions for push and pop. However, we used pseudo instructions to implement push, push offset, and pop instructions as follows:

Instructions	Pseudo Instructions
push $t1	sw $t1, 0($sp) subi $sp, $sp, 1

push 3($t1)	lw $t5, 3($t1)
	sw $t5, 0($sp)
	subi $sp, $sp, 1

pop $t1	addi $sp, $sp, 1 lw $t1, 0($sp)

Table 5: Pseudo Instructions For Push-Pop

Although these instructions require 2, 3, and 2 clock cycles, respectively, there was no need for handling them separately.

6 Approach to implement branch operation

We had to implement beq and bneq operations in our design. For this, we used two control bits branch and bneq from the control unit. Whether branch is enabled or not is decided using the zero flag of the ALU unit.

Branch	Bneq	Zero Flag	Branch Enable
0	x	x	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 6: Branch Enable

$BranchEnable = Branch \wedge (Bneq \oplus Zero)$

7 IC Count

IC	**Count**
Atmega32	6
IC-7483	4
IC-74157	7

Table 7: IC Count for MIPS

8 The Simulator Used along with the Version Number

Logisim - 2.7.1

9 Discussion

In this project, we implemented a 4 bit Mips supporting the instruction set as specified here 1.
We initially implemented the software simulation using **Logisim**. For simulating atmega32 we used Ram and Rom. In line with our hardware design, we also implemented a proteus design. For converting mips assembly to hex for instruction memory, we designed a compiler. It also generates the hex codes needed for the logisim design. We also implemented a manual clock and reset in our design.
The register file has 8 registers in total with 6 temporary registers ($t0, $t1,$t2,$t3,$t4,$t5), 1 zero register $zero , and 1 stack pointer register($sp) which is an extra temporary register was used to implement the push-pop instructions associated with memory.
From the ALU, we demonstrated output in each step of operation. The value of PC was also shown for each clock cycle. All of our ATmega were tested before integrating it to our final circuit.Wires with different sizes were used to make the whole circuit organized and easily understandable for the user.The outputs were checked multiple times for a large number of input test cases to ensure that our circuit satisfies the expected output values. Using minimal number of ICs in order to reduce complexity was the priority while designing the circuit. Outputs of some of the ICs were reused to minimize the circuit even more. The connections were made carefully and the components were placed at fair distances. Messiness was avoided as much as possible to ensure higher readability. Suitable labels were used at different parts of the circuit to make the functions of individual parts more understandable. Finally, the circuit was tested several times to make sure it did not have any sort of error.

10 Contribution

2005020 - Mostafa Rifat Tazwar

- Simulation
  - Control Unit & Bug fix
- Report
  - Control Unit
  - Approach to implement the push,pop and branch instructions
  - IC Count

2005025 - Most. Sonia Khatun

- Simulation
  - ALU Unit
- Report
  - Introduction & Instruction Set
  - Discussion & ALU

2005027 - Swastika Pandit

- Simulation
  - Register File
  - Connecting the components to form complete circuit
- Report
  - Register File
- Hardware Implementation

2005029 - MD. Minhajul Islam Fuad

- Simulation
  - Instruction Memory with PC
- Report
  - Circuit Diagram
  - Instruction Memory with PC

2005030 - Fairuz Mubashwera

- Simulation
  - Data Memory
- Report
  - Data Memory With Stack
- Hardware Implementation