Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN2031 - Fundamentals of Computer Organization and Design

ISA and Microarchitecture Design Final Report

R.M.L.H. Ratnayake 210542B

A.D.T. Dabare 210089P

I.D. Madhushan 210349N

Part (a) i)

1. Arithmetic and Logic Instructions

Opcode	Operands		Meaning
	Source	Destination	
ADD	R_A, R_B	R_A	$R_A \leftarrow R_A + R_B$ Add data in Registers R_A and R_B and write the result to Register R_A
SUB	R_A, R_B	R_A	$R_A \leftarrow R_A - R_B$ Subtract data in Registers in R_A and R_B and write the result to Register R_A
AND	R_A, R_B	R_A	$R_A \leftarrow R_A\ AND\ R_B$ Perform bitwise AND operation with data in Registers R_A and R_B and write the result to R_A
OR	R_A, R_B	R_A	$R_A \leftarrow R_A\ OR\ R_B$ Perform bitwise OR operation with data in Registers R_A and R_B and write the result to R_A

2. Stack Operations

Opcode	Operands	Meaning
POP	R_A	$R7 \leftarrow R7 + 2,\ R_A \leftarrow Stack[R7]$ Increment Stack Pointer by 2 and then, move the data in the top of the stack and store it in the Register R_A
PUSH	R_A	$Stack[R7] = R_A,\ R7 \leftarrow R7 - 2$ Write the data in the Register R_A to the top of the stack, and decrement the Stack Pointer by 2

Stack is assumed to be filled from top to bottom.

3. Memory Operations

Opcode	Operands		Meaning
	Source	Destination	
LOAD	R_S	R_D	$R_D \leftarrow DM[R_S]$ Take the address in the register R_s and load the value in that memory address to the register R_D
LOADM	R_S	R_D	$R_D \leftarrow DM[R_S],\ R_S \leftarrow R_S + 2$ Take the address in the register R_s and load the value in that memory address to the register R_D, and then increment the address.
STORE	R_S, R_D		$DM[R_D] \leftarrow R_S$ Store value in Rs register to the memory address specified by R_D.

4. Data Transfer Instructions

Opcode	Operands	Meaning
MOV	R_A, R_B	$R_B \leftarrow R_A$ Move data in the register R_A to the register R_B

5. Branch Instructions

Opcode	Operands	Meaning
BRANCH	imm[7:0]	$PC \leftarrow PC + imm[7:0]$ Unconditionally add immediate value in the instruction to the PC.
BEQ	SREG	$SREG \leftarrow R_A - R_B$ $PC \leftarrow PC + imm[7:0]$ Branch if the selected two registers (R_A, R_B) are equal (using SREG).
BLE	SREG	$SREG \leftarrow R_A - R_B$ $PC \leftarrow PC + imm[7:0]$ Branch if R_A is less than R_B or equal (using SREG).
BGE	SREG	$SREG \leftarrow R_A - R_B$ $PC \leftarrow PC + imm[7:0]$ Branch if R_A greater than R_B or equal (using SREG).

SREG - This is the register where the output of a certain operation is saved immediately after execution.

6. Load Operations

Opcode	Operands	Meaning
LOADL	R_D, imm[7:0]	$R_D[7:0] \leftarrow imm[7:0]$ Load the first 8-bits of the immediate into a register.
LOADU	R_D, imm[7:0]	$R_D[15:8] \leftarrow imm[7:0]$ Load the last 8-bits of the immediate into a register.

Part (a) ii)

R Type - ADD, SUB, AND, OR

15 11	10 8	7 5	4 0
	R_B	R_A	OPCODE
5 bits	**3 bits**	**3 bits**	**5 bits**

S Type - LOAD, LOADM, STORE, MOV

15 11	10 8	7 5	4 0
	R_D	R_S	OPCODE
5 bits	**3 bits**	**3 bits**	**5 bits**

Special Case of S - PUSH, POP

15 11	10 8	7 5	4 0
		R_A	OPCODE
5 bits	**3 bits**	**3 bits**	**5 bits**

I Type - BRANCH, BEQ, BLE, BGE

15 8	7 5	4 0
imm [7:0]	R_A	OPCODE
8 bits	**3 bits**	**5 bits**

L Type - LOADL, LOADU

15 8	7 5	4 0
imm [7:0]	R_D	OPCODE
8 bits	**3 bits**	**5 bits**

Justification -

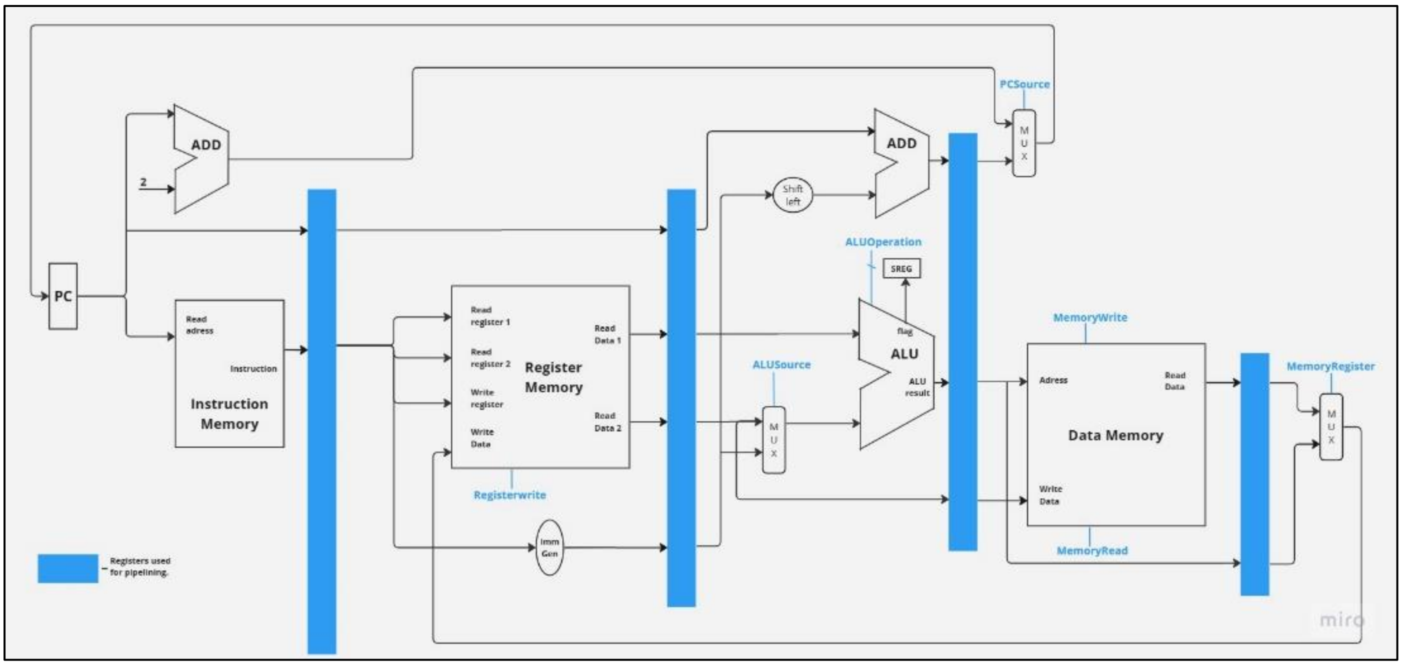
We have divided our instruction set under five types depending on the similarities between them. Main point we can highlight here is that we have designed the format in such a way that all the formats have similar fields and equal number of bits assigned (i.e., first we have the OPCODE, second we have the Operands and at last the immediate values). The reason behind this is that we can design the microarchitecture using hard wired configurations rather than using micro programming. Hence it will be much easier to do the decoding operations as well.

Part (b)

Type	Instruction	Instruction Structure (bits)															
		15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	ADD						R_B[2:0]			R_A[2:0]			0	0	0	0	0
	SUB						R_B[2:0]			R_A[2:0]			1	0	0	0	0
	AND						R_B[2:0]			R_A[2:0]			0	1	0	0	0
	OR						R_B[2:0]			R_A[2:0]			1	1	0	0	0
S	LOAD						R_D[2:0]			R_S[2:0]			0	0	0	0	1
	LOADM						R_D[2:0]			R_S[2:0]			1	0	0	0	1
	STORE						R_D[2:0]			R_S[2:0]			0	1	0	0	1
	MOV						R_D[2:0]			R_S[2:0]			1	1	0	0	1
S-Special	PUSH									R_A[2:0]			0	0	0	0	1
	POP									R_A[2:0]			1	0	0	0	1
I	BRANCH	imm[7:0]											0	0	0	1	0
	BEQ	imm[7:0]								SREG[2:0]			1	0	0	1	0
	BLE	imm[7:0]								SREG[2:0]			0	1	0	1	0
	BGE	imm[7:0]								SREG[2:0]			1	1	0	1	0
L	LOADL	imm[7:0]								R_D[2:0]			0	0	0	1	1
	LOADU	imm[7:0]								R_D[2:0]			1	0	0	1	1

Part (c)

Datapath with control signals for data path elements



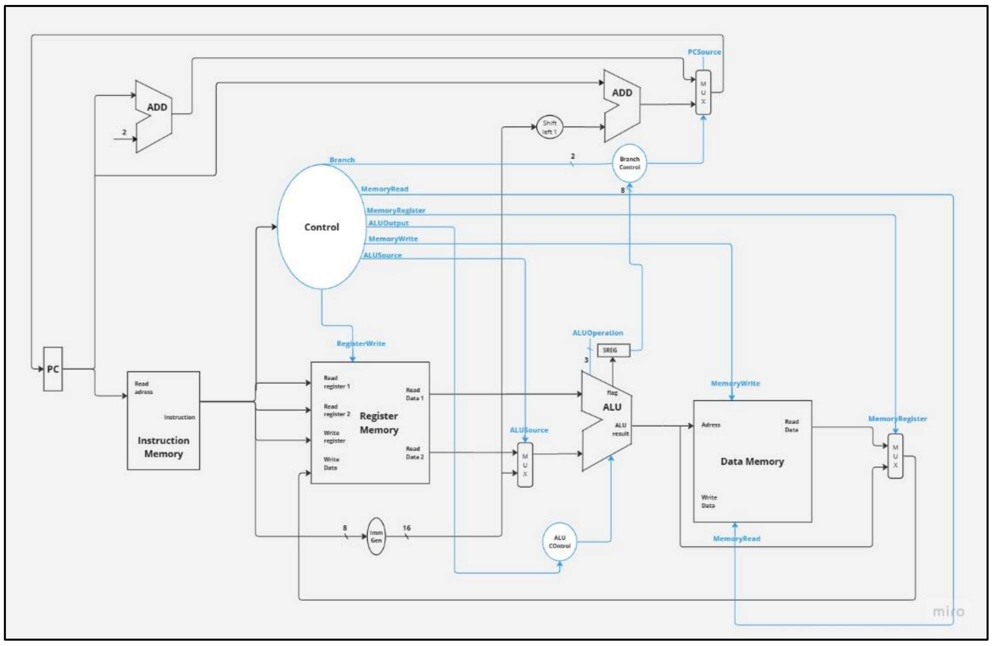
Part (d) i)

We have designed using the hard wired approach over microprogramming for the following reason.

Considering the fewer number of instructions, and their level of simplicity, it is not needed to use the micro programming approach where instructions are divided into smaller micro instructions. Also, these instructions can be executed within a single cycle meaning pipelining can be implemented easily in the hard wired approach.

Other than the instructions listed above, the CPU is not anticipated to add any more instructions. Control stores do not require microprogramming because the ISA is not explicitly anticipated to be modular for new instructions.

Part (d) ii)



Instruction	RegisterWrite	ALUSource	ALUOperation	PCSource	MemoryWrite	MemoryRead	MemoryReg	Branch
ADD	0	0	000	0	0	0	0	00
SUB	0	0	001	0	0	0	0	00
AND	0	0	010	0	0	0	0	00
OR	0	0	011	0	0	0	0	00
LOAD	1	0	000	0	0	1	1	00
LOADM	1	0	000	0	0	1	1	00
STORE	0	0	000	0	1	0	0	00
MOV	1	0	100	0	0	0	0	00
PUSH	0	0	100	0	1	0	0	00
POP	1	0	100	0	0	1	1	00
BRANCH	0	1	000	1	0	0	0	00
BEQ	0	1	000	1	0	0	0	01
BLE	0	1	000	1	0	0	0	10
BGE	0	1	000	1	0	0	0	11
LOADL	1	1	000	0	0	1	1	00
LOADU	1	1	000	0	0	1	1	00

RegisterWrite	0	No writing to the Register File
	1	Writing to the Register File
ALUSource	0	Register
	1	Immediate
ALUOperation	000	Add
	001	Subtract
	010	AND
	011	OR
	100	No operation (Pass value)
PCSource	0	From the adder which adds 2 to the PC
	1	From the adder which adds the immediate to the PC
MemoryWrite	0	No writing to the Data Memory
	1	Writing to the Data Memory
MemoryRead	0	No reading from the Data Memory
	1	Reading from the Data Memory
MemoryReg	0	Output from the ALU
	1	Output from the Data Memory