32 Bit Processor using Verilog

Instruction Format, Data Path, and Control Unit Design

Group 3

Parth Patil - **22CS30041**

Rishit Garg- **22CS30045**

1 Instruction Encoding

1.1 R - Type Instructions

opcode	rs	rt	rd	dont care	shamt	func
6 bits	5 bits	5 bits	5 bits	6 bits	1 bit	4 bits

1.2 I - Type Instructions

opcode	rs	rt	Immediate
6 bits	5 bits	5 bits	16 bits

1.3 J - Type Instructions

opcode	Immediate
6 bits	26 bits

2 Usage and Format Details

Class	Instruction	Usage	Opcode	Function	Type
Arithmetic and Logical	ADD	add rd, rs, rt	000000	0001	R type
	SUB	sub rd, rs, rt	000000	0010	R type
	AND	and rd, rs, rt	000000	0100	R type
	OR	or rd, rs, rt	000000	0101	R type
	XOR	xor rd, rs, rt	000000	0110	R type
	NOR	nor rd, rs, rt	000000	0111	R type
	NOT	not rd, rs	000000	0000	R type
	SL	sl rd, rs, rt	000000	1001	R type
	SRL	srl rd, rs, rt	000000	1010	R type
	SRA	sra rd, rs, rt	000000	1011	R type
	SLT	slt rd, rs, rt	000000	1100	R type
	SGT	sgt rd, rs, rt	000000	1101	R type
	INC	inc rt, rs	001000	NA	R type
	DEC	dec rt, rs	000011	NA	R type
	ADDI	addi rt, rs, #imm	000001	NA	I type
	SUBI	subi rt, rs, #imm	000010	NA	I type
	ANDI	andi rt, rs, #imm	000100	NA	I type
	ORI	ori rt, rs, #imm	000101	NA	I type
	XORI	xori rt, rs, #imm	000110	NA	I type
	NORI	nori rt, rs, #imm	000111	NA	I type
	SLI	sli rt, rs, #imm	001001	NA	I type
	SRLI	srli rt, rs, #imm	001010	NA	I type
	SRAI	srai rt, rs, #imm	001011	NA	I type
	HAM	ham rt, rs	001111	NA	I type
	LUI	lui rt, #imm	001110	NA	I type
Memory	LD	ld rt, #imm(rs)	010010	NA	I type
	ST	st rt, #imm(rs)	010011	NA	I type
Branch	BR	br #imm	110100	NA	J type
	BMI	bmi rs, #imm	110001	NA	I type
	BPL	bpm rs, #imm	110010	NA	I type
	BZ	bz rs, #imm	110011	NA	I type
Move	MOVE	move rt, rs	010001	NA	I type
	CMOV	cmov rd, rs, rt	011100	NA	R type
Misc	HALT	halt	011110	NA	J type
	NOP	nop	011111	NA	J type

2.1 Examples

1. ADDI R5, R4, #-1 - *000001 00100 00101 1111111111111111*
2. ADD R1, R2, R3 - *000000 00010 00011 00001 000000 0 0001*
3. SL R5, R5, R7 - *000000 00101 00111 00101 0000000 1001*
4. LD R2, 10(R6) - *010010 00110 00010 0000000000001010*
5. ST R2, -2(R11) - *010011 01011 00010 1111111111111110*
6. BR #10 - *110100 0000000000001010*
7. BZ R8, #-75 - *110011 01000 00000 11111111001*

3 Register Encoding

Register	Function	Code
$R0	Hardwired to 0	00000
$R1 - $R15	General purpose registers	00001 to 01111
$RET	Store return value of function	10001
$PC	Program Counter	10000

4 Control Signals and Values

Opcode	Control Signals									
	brOp	**aluOp**	**BSel**	**wrRegSel**	**memRd**	**memWr**	**regWr**	**regISel**	**sgnExt**	**isMV**
000000	000	func	1	1	0	0	1	00	x	0
000010	000	0010	0	0	0	0	1	00	0	0
000001	000	0001	0	0	0	0	1	00	0	0
000100	000	0100	0	0	0	0	1	00	0	0
000101	000	0101	0	0	0	0	1	00	0	0
000110	000	0110	0	0	0	0	1	00	0	0
000111	000	0111	0	0	0	0	1	00	0	0
001001	000	1001	0	0	0	0	1	00	0	0
001010	000	1010	0	0	0	0	1	00	0	0
001011	000	1011	0	0	0	0	1	00	0	0
001111	000	1111	0	0	0	0	1	00	0	0
001110	000	1110	0	0	0	0	1	00	0	0
010010	000	xxxx	0	x	1	0	1	01	0	0
010011	000	xxxx	0	x	0	1	0	xx	0	0
110100	100	xxxx	0	x	0	0	0	xx	1	0
110001	001	xxxx	0	x	0	0	0	xx	1	0
110010	010	xxxx	0	x	0	0	0	xx	1	0
110011	011	xxxx	0	x	0	0	0	xx	1	0
010001	000	xxxx	0	0	0	0	1	11	0	0
011100	000	1100	0	1	0	0	1	1x	x	1
011110	000	xxxx	0	x	0	0	0	xx	1	0
011111	000	xxxx	0	x	0	0	0	xx	1	0

*the instruction number is decoded from Opcode

1. **brOp**: Determines if the instruction is a branch operation (*implemented using MUX*)

$$brOp = Opcode[5] \,?\, Opcode[2:0] : 000$$

implemented using MUX

2. **aluOp**: Specifies the operation for the ALU (Arithmetic Logic Unit) (*implemented using MUX*)

$$aluOp = I_0 \, ? \, func : Opcode[3:0]$$

3. **BSel**: Selects the second input for the ALU from a register [Register / Immediate].

$$BSel = I_0$$

4. **wrRegSel**: Selects the destination register for the result [Rd / Rt]

$$wrRegSel = I_0 \vee I_{28}$$

5. **memRd**: Indicates if data should be read from memory

$$memRd = I_{18}$$

6. **memWr**: Indicates if data should be written to memory

$$memWr = I_{19}$$

7. **regWr**: Specifies whether to write data to a register

$$regWr = (\neg Opcode[4]) \vee I_{18} \vee I_{17} \vee I_{28}$$

8. **regISel**: Selects the data source to be written to a register [Data Memory / ALU out]

$$regISel[0] = I_{18} \vee I_{17} \; ; \; regISel[1] = I_{17} \vee I_{28}$$

9. **sgnExt**: Controls which sign extended Immediate Data to choose [Imm / Jump]

$$sgnExt = Opcode[5] \vee (Opcode[4] \wedge Opcode[3])$$

10. **isMV**: Controls LSB of regISel in accordance with ALU output

$$isMV = I_{28}$$

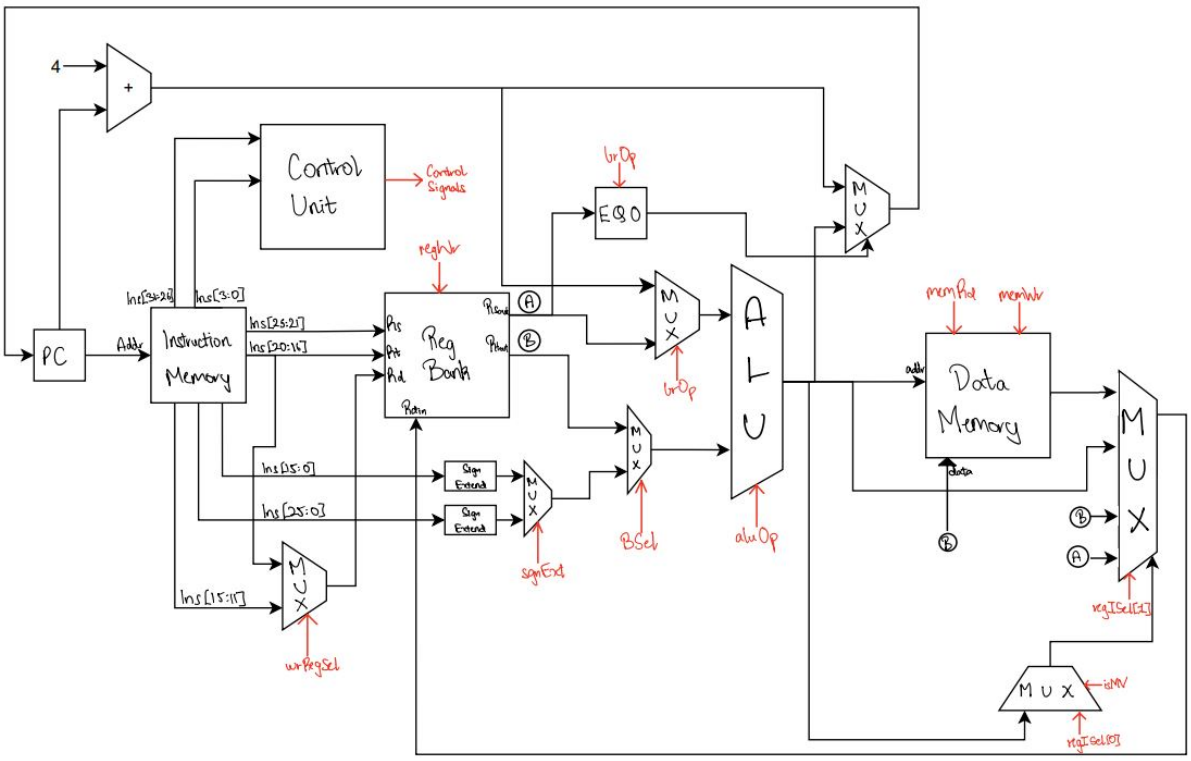

Figure 1: Visual Representation of the Datapath. Red text denotes control signals