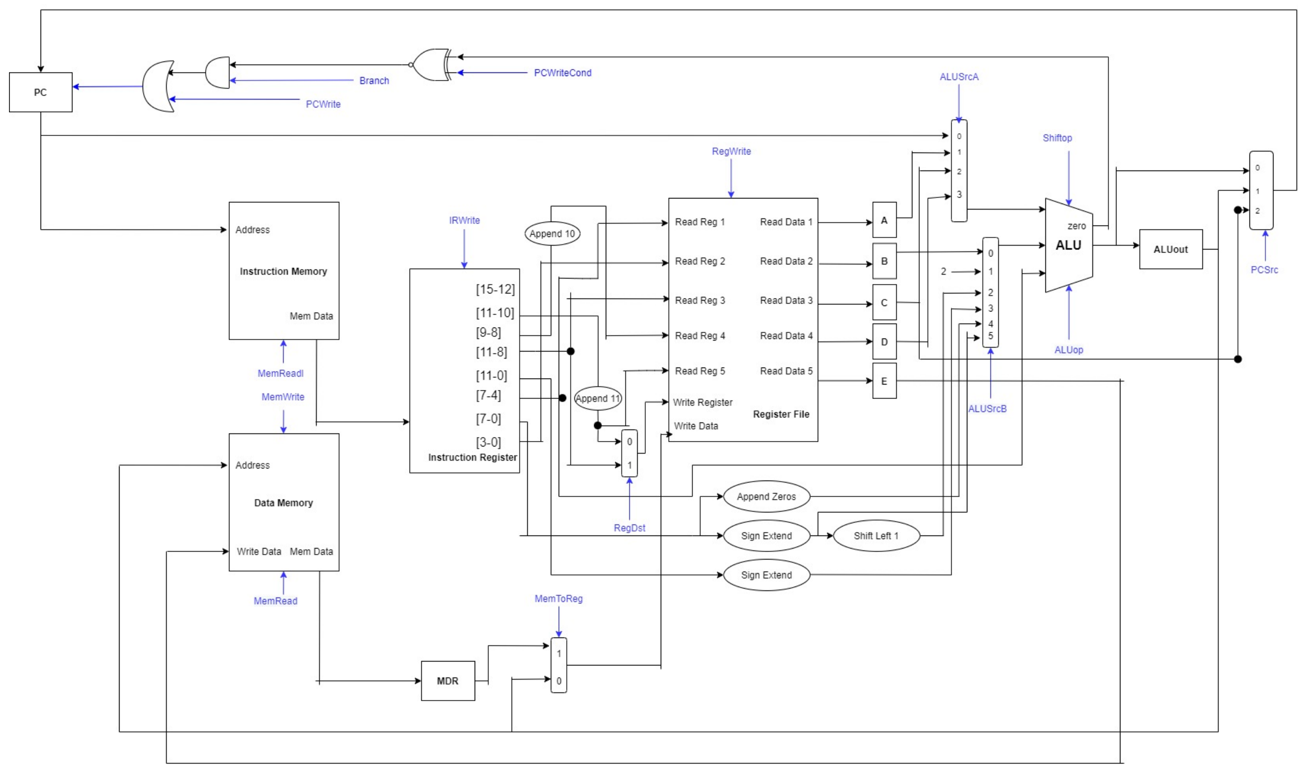
Computer Architecture Assignment

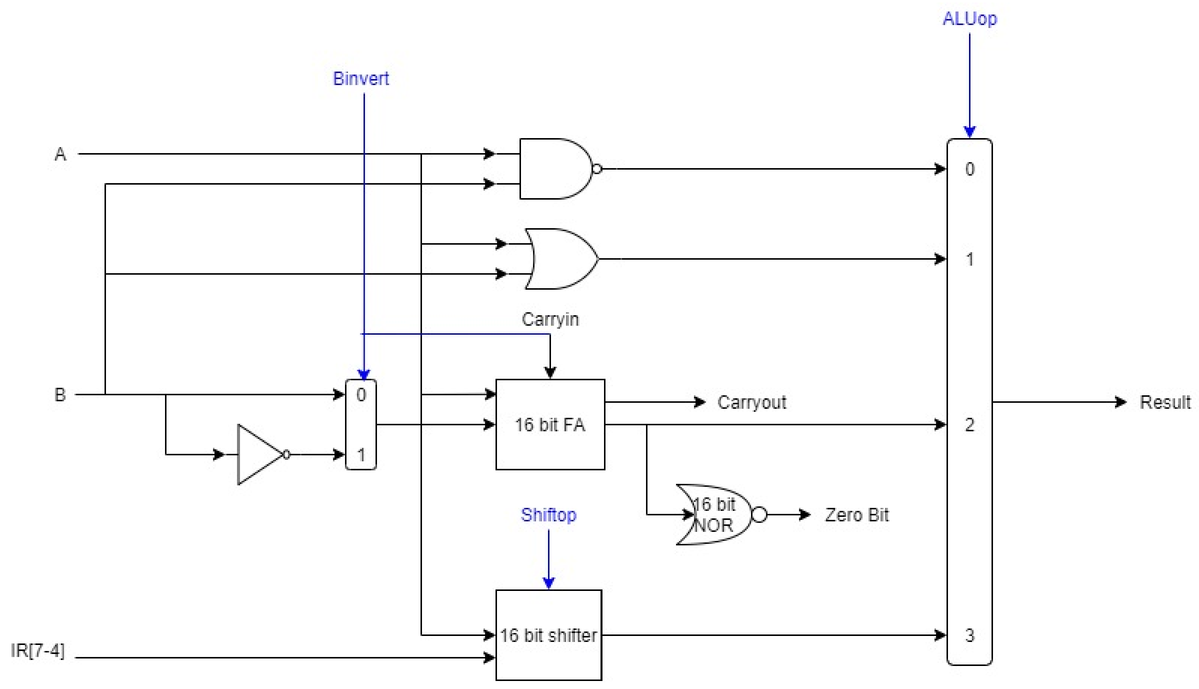
Aarshibh Singh	2018A3PS0437P
Karan Singh Mathur	2018A3PS0340P
N Harishchandra Prasad	2018A3PS0422P

20th April, 2021

Datapath



ALU Design



Sequence of Operations

```
1. IF:

IR = Memory[PC]
PC = PC+2

2. ID:

A = Reg[IR[7-4]]
B = Reg[IR[3-0]]
C = Reg[IR[11-8]]
D = Reg[IR[[2’b10,IR[9-8]]]
E = Reg[IR[2’b11,IR[11-10]]]

3. EX:

if branch:
        PC = C

if jump:
        PC = PC + sign_extend(IR[11:0])

if R-type:

        if Register Addressing
        ALUout = AopB

        If Immediate Addressing
        ALUout = CopImmediate_data (sign extend or append zeros)

if memory ref (LW/SW):
        ALUout = D + sign_extend(IR[7:0]<<1)

4. MEM:

if R-type:
        Reg[IR(11:8)] = ALUout

if SW:
        Memory[ALUout] = Reg[IR({2'b11,IR(11:10)})] = E

if LW:
        MDR = Memory[ALUout]

5. WB:
        Reg[{2'b11,IR(11:10)}] = MDR
```

Control Signals

Control Signal	**Value**	**Description**
PCWrite	0	PC register can not be written to
	1	PC register can be written to
Branch	0	Instruction is not Branch
	1	Instruction is Branch
PCWriteCond	0	Branch Not equal Instruction
	1	Branch Equal Instruction
MemReadI	0	Cannot read from instruction memory
	1	Can read from instruction memory
MemRead	0	Cannot read from data memory
	1	Can read from data memory
MemWrite	0	Cannot write to data memory
	1	Can write to data memory
IRWrite	0	Cannot write to instruction register
	1	Can write to instruction register
RegWrite	0	Cannot write to register file
	1	Can write to register file
RegDst	0	{2'b11,IR[11-10]} chosen as the write destination register. For instruction: LW
	1	IR[11-8] chosen as the write destination register. For instruction: for all except branch, jump, SW and LW. (Don't care for branch, jump and SW)
MemToReg	0	ALUout chosen as input to write data. For Instruction: For all except branch, jump, SW and LW. (Don't care for branch, jump and SW)
	1	MDR chosen as input to write data. For Instruction: LW
ALUSrcA	00	PC Chosen as one input to ALU. For Instruction: Jump, IF state.
	01	Reg A (Source 1, IR[7-4]) Chosen as one input to ALU. For Instruction: Add, Subtract, NAND, OR, Branch
	10	Reg C (Destination, IR[11-8]) Chosen as one input to ALU. For Instruction: Addi, Subi, Shift, NANDi, ORi
	11	Reg D (RP, {2'b10,IR[9-8]}) Chosen as one input to ALU. For Instruction: LW, SW (for target address).
ALUSrcB	000	Reg B (Source 2, IR[3-0]) Chosen as other input to ALU. For instruction: Add, Subtract, NAND, OR, Branch

	001	2 Chosen as other input to ALU. For IF stage, PC+2
	010	(Sign_extend(IR[7-0])<<1) Chosen as other input to ALU. For Instruction: SW, LW
	011	Sign_extend(IR[11-0]) Chosen as other input to ALU. For Instruction: Jump
	100	Append_Zeros(IR[7-0]) Chosen as other input to ALU. For instruction: Addi, Subi
	101	Sign_extend(IR[7-0]) Chosen as other input to ALU. For Instruction: Addi, Subi, NANDi, ORi
ALUop	000	NAND
	001	OR
	010	Add
	011	Shift
	110	Subtract
Shiftop	01	Shift Left Logical
	10	Shift Right Logical
	11	Shift Arithmetic Right
PCSrc	00	Output of ALU chosen as input to PC. For Instruction: IF stage, Jump
	01	ALUout chosen as input to PC. For Instruction: Probably not needed
	10	Reg C (Destination, IR[11-8]) chosen as input to PC. For Instruction: Branch

States

State	Control Signal Values	Description

State 0 (IF)	MemReadI = 1
	IRWrite = 1
	ALUSrcA = 00
	ALUSrcB = 001
	ALUop = 010
	PCSrc = 00
	PCWrite = 1
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	Branch = 0
	All other don’t care
		Instruction Fetch Stage.
		IR = Memory[PC]
		PC = PC + 2

State 1 (ID)	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Instruction Decode Stage.
		A = Reg[IR[7-4]]
		B = Reg[IR[3-0]]
		C = Reg[IR[11-8]]
		D = Reg[IR[[2’b10,IR[9-8]]]
		E = Reg[IR[2’b11,IR[11-10]]]

State 2	ALUSrcA = 01
	ALUSrcB = 000
	ALUop = 010
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Add instruction, Register Addressing.
		ALUout = RS1 + RS2
		ALUout = A + B
		Opcode: 1000

State 3	MemToReg = 0
	RegDst = 1
	RegWrite = 1
	MemReadI = 0
	PCWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		R-type Instruction
		RD = ALUout
		Reg[IR[11-8]] = ALUout
		
		ALUout goes into the write data input

State 4	ALUSrcA = 01
	ALUSrcB = 000
	ALUop = 110
		Subtract instruction, Register Addressing.
		ALUout = RS1 - RS2

	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		ALUout = A - B
		Opcode: 1100

State 5	ALUSrcA = 01
	ALUSrcB = 000
	ALUop = 000
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Logical NAND Register Addressing
		ALUout = RS1 nand RS2
		ALUout = A nand B
		Opcode: 1011

State 6	ALUSrcA = 01
	ALUSrcB = 000
	ALUop = 001
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Logical OR Register Addressing
		ALUout = RS1 or RS2
		ALUout = A or B
		Opcode: 1111

State 7	ALUSrcA = 10
	ALUSrcB = 101
	ALUop = 010
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Addition Immediate Addressing (Sign extended)
		ALUout = RD + Sign extended Immediate data
		ALUout = C + Sign extended Immediate data
		Opcode: 1001

State 8	ALUSrcA = 10
	ALUSrcB = 100
	ALUop = 010
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
		Addition Immediate Addressing (upper byte filled by zeros)
		ALUout = RD + append zeros
		ALUout = C + append zeros
		Opcode: 1010

	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care	

State 9	ALUSrcA = 10
	ALUSrcB = 101
	ALUop = 110
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Subtraction Immediate Addressing (Sign extended)
		ALUout = RD - Sign extended Immediate data
		ALUout = C - Sign extended Immediate data
		Opcode: 1101

State 10	ALUSrcA = 10
	ALUSrcB = 100
	ALUop = 110
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Subtraction Immediate Addressing (upper byte filled by zeros)
		ALUout = RD - append zeros
		ALUout = C - append zeros
		Opcode: 1110

State 11	ALUSrcA = 10
	ALUSrcB = 101
	ALUop = 000
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Logical NAND Immediate (sign extended)
		ALUout = RD nand Sign extended Immediate data
		ALUout = C nand Sign extended Immediate data
		Opcode: 0111

State 12	ALUSrcA = 10
	ALUSrcB = 101
	ALUop = 001
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
		Logical OR Immediate (sign extended)
		ALUout = RD or Sign extended Immediate data
		ALUout = C or Sign extended Immediate data
		Opcode: 0110

	Branch = 0 All other don't care	

State 13	ALUSrcA = 10
	ALUop = 011
	Shiftop = 01
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don't care
		Shift Left Logical
		ALUout = RD << immediate data
		ALUout = C << immediate data
		Opcode: 0000
		Func Field: 0001

State 14	ALUSrcA = 10
	ALUop = 011
	Shiftop = 10
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don't care
		Shift Right Logical
		ALUout = RD >> immediate data
		ALUout = C >> immediate data
		Opcode: 0000
		Func Field: 0010

State 15	ALUSrcA = 10
	ALUop = 011
	Shiftop = 11
	MemReadI = 0
	PCWrite = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don't care
		Shift Arithmatic Right
		ALUout = RD >>> immediate data
		ALUout = C >>> immediate data
		Opcode: 0000
		Func Field: 0011

State 16	ALUSrcA = 01
	ALUSrcB = 000
	ALUop = 110
	PCWrite = 0
	PCWriteCond = 1
	PCSrc = 10
	MemReadI = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 1
		Branch Equal
		If (A-B = 0) zero bit set
		PC = C
		Opcode: 0100

	All other don’t care	

State 17	ALUSrcA = 01
	ALUSrcB = 000
	ALUop = 110
	PCWrite = 0
	PCWriteCond = 0
	PCSrc = 10
	MemReadI = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 1
	All other don’t care
		Branch not Equal
		If (A-B != 0) zero bit not set
		PC = C
		Opcode: 0101

State 18	ALUSrcA = 00
	ALUSrcB = 011
	ALUop = 010
	PCSrc = 00
	PCWrite = 1
	MemReadI = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Jump Instruction.
		PC = PC +
		sign_extend(IR[11:0])
		Opcode: 0011

State 19	ALUSrcA = 11
	ALUSrcB = 010
	ALUop = 010
	PCWrite = 0
	MemReadI = 0
	RegWrite = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Target Address calculation
		for LW and SW.
		ALUout = Reg{2’b10,IR[9-8]}
		+ sign extend(IR[7:0]<<1)
		ALUout = D + sign
		extend(IR[7:0]<<1)
		Opcode: 0001,0010

State 20	MemWrite = 1
	PCWrite = 0
	MemReadI = 0
	RegWrite = 0
	MemRead = 0
	IRWrite = 0
	Branch = 0
	All other don’t care
		Store Word Instruction
		Memory[ALUout] =
		Reg[IR({2'b11,IR(11:10)})]
		Memory[ALUout] = E
		Opcode: 0010

State 21	MemRead = 1
	PCWrite = 0
		Load Word Instruction
		MDR = Memory[ALUout]

	MemReadI = 0
	RegWrite = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don't care
		Opcode: 0001

State 22	MemToReg = 1
	RegDst = 0
	RegWrite = 1
	PCWrite = 0
	MemReadI = 0
	MemRead = 0
	MemWrite = 0
	IRWrite = 0
	Branch = 0
	All other don't care
		Load Word Instruction
		Reg[{2'b11,IR(11:10)}] = MDR
		Opcode: 0001

State Diagram

1) Add instruction Register Addressing opcode 1000

   State 0 (IF) → State 1 (ID) → State 2 → State 3

2) Subtract instruction Register Addressing opcode 1100

   State 0 (IF) → State 1 (ID) → State 4 → State 3

3) NAND instruction Register Addressing opcode 1011

   State 0 (IF) → State 1 (ID) → State 5 → State 3

4) OR instruction Register Addressing opcode 1111

   State 0 (IF) → State 1 (ID) → State 6 → State 3

5) Addition Immediate Addressing (Sign extended) opcode 1001

   State 0 (IF) → State 1 (ID) → State 7 → State 3

6) Addition Immediate Addressing (upper byte filled by zeros) opcode 1010

   State 0 (IF) → State 1 (ID) → State 8 → State 3

7) Subtraction Immediate Addressing (Sign extended) opcode 1101

   State 0 (IF) → State 1 (ID) → State 9 → State 3

8) Subtraction Immediate Addressing (upper byte filled by zeros) opcode 1110

   State 0 (IF) → State 1 (ID) → State 10 → State 3

9) Logical NAND Immediate Addressing (Sign extended) opcode 0111

   State 0 (IF) → State 1 (ID) → State 11 → State 3

10) Logical OR Immediate Addressing (Sign extended) opcode 0110

    State 0 (IF) → State 1 (ID) → State 12 → State 3

11) Shift Left Logical opcode 0000 Func Field 0001

    State 0 (IF) → State 1 (ID) → State 13 → State 3

12) Shift Right Logical opcode 0000 Func Field 0010

    State 0 (IF) → State 1 (ID) → State 14 → State 3

13) Shift Arithmatic Right opcode 0000 Func Field 0011

    State 0 (IF) → State 1 (ID) → State 15 → State 3

14) Branch Equal opcode 0100

    State 0 (IF) → State 1 (ID) → State 16

15) Branch Not Equal opcode 0101

    State 0 (IF) → State 1 (ID) → State 17

16) Jump opcode 0011

    State 0 (IF) → State 1 (ID) → State 18

17) Store Word opcode 0010

    State 0 (IF) → State 1 (ID) → State 19 → State 20

18) Load Word opcode 0001

    State 0 (IF) → State 1 (ID) → State 19 → State 21 → State 22

Note: After the last state it starts again from state zero

Test instructions being executed –

1000_1101_1000_0111

// add reg7, reg8 store in reg13

1001_0001_1000_0000

//imm. add sign extended 1000_0000 to reg1, store in reg1

1010_0010_0000_0100

// imm. add 4 to reg2 and store in reg2

1100_1110_1000_0111

// subtracting reg7 from reg8 store in reg14

1101_0101_1000_0000

// imm. sub sign extended 1000_0000 from reg5 and store in reg5

1110_0100_0000_0011

// imm. sub 3 from reg4 and store in reg4

1111_1111_1000_0111

// OR reg7, reg8 store in reg 15

0000_1000_0010_0011

// shift reg8 arithmetic right by 2

0001_0001_0000_0000

// load word to reg 12 from address specified by reg9

1010_0010_0000_0010

// add reg2 to 2 store in reg2

0011_0000_0000_0110

// jump 6 Bytes

1010_0010_0000_0010

// garbage

1010_0010_0000_0010

// garbage

1010_0010_0000_0010

// garbage

0010_01_10_0000_0000

//store reg13 in dat_mem, at location specified by reg10

0000_1000_0100_0001

// reg8 logical left shift by 4

0000_1000_0001_0010

// r8 log right by 1

1011_1111_0001_0010

// nand reg1 and reg2 store in reg15

0111_1111_0000_0000

// imm. NAND of reg15 with 0000_0000

0110_1111_0000_0000

// imm. OR of reg15 with 0000_0000

0011_0000_0000_0100

//jump 4 Bytes

0100_0011_1000_0111

// branch to addresss in reg3, if reg7, reg8 equal

0101_0011_1000_0111

// branch to addresss in reg3, if reg7, reg8 not equal

1000_1101_1000_0111

// add reg7, reg8 store in reg13

1001_0001_1000_0000

//imm. add sign extended 1000_0000 to reg1, store in reg1

1010_0010_0000_0100

// imm. add 4 to reg2 and store in reg2

1100_1110_1000_0111

// subtracting reg7 from reg8 store in reg14

1101_0101_1000_0000

// imm. sub sign extended 1000_0000 from reg5 and store in reg5

1110_0100_0000_0011

// imm. sub 3 from reg4 and store in reg4

1111_1111_1000_0111

// OR reg7, reg8 store in reg 15

0000_1000_0010_0011

// shift reg8 arithmetic right by 2

0001_0001_0000_0000

// load word to reg 12 from address specified by reg9

1010_0010_0000_0010

// add reg2 to 2 store in reg2

0011_0000_0000_0110

```
// jump 6 Bytes
1010_0010_0000_0010
// garbage
1010_0010_0000_0010
// garbage
1010_0010_0000_0010
// garbage
0010_01_10_0000_0000
//store reg13 in dat_mem, at location specified by reg10
0000_1000_0100_0001
// reg8 logical left shift by 4
0000_1000_0001_0010
// r8 log right by 1
1011_1111_0001_0010
// nand reg1 and reg2 store in reg15
0111_1111_0000_0000
// imm. NAND of reg15 with 0000_0000
0110_1111_0000_0000
// imm. OR of reg15 with 0000_0000
0100_0011_1000_0111
// branch to addresss in reg3, if reg7, reg8 equal
0101_0011_1000_0111
// branch to addresss in reg3, if reg7, reg8 not equal
```