Computer Organization and Architecture Laboratory

Assignment 7

Anish Datta (21CS30006)
Smarak Kanjilal(21CS30061)

Instruction Format Encoding

Number of supported registers = 2^5 = 32

R-type					
OPCODE	Source Register 1	Source Register 2	Destination Register	Shamt	Func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

First 6 bits will be reserved for the opcode.
Next 10 bits will be reserved for the destination register.
Next 5 bits will be reserved for two source registers.
Next 5 bits are used for shamt (don't care as it can only shift by 1 bit depending on operand).
Next 6 bits are used for the function name (don't care as determined by opcode).

Operations: ADD, SUB, AND, OR, XOR, NOT, SLA, SRA, SRL

I1-type			
OPCODE	Source Register	Destination Register	Immediate
6 bits	5 bits	5 bits	16 bits

First 6 bits will be reserved for the opcode.
Next 10 bits will be reserved for the destination and source registers.
Next 16 bits will be reserved for the immediate value.

Range of immediate value: -2^{16} to $2^{16}-1$

Operations: LD, ST, MOVE

I2-type		
OPCODE	Register	Immediate
6 bits	5 bits	21 bits

First 6 bits will be reserved for the opcode.
Next 5 bits will be reserved for the register.
Next 21 bits will be reserved for the immediate value.

Range of immediate value: -2^{21} to $2^{21}-1$

Operations: LDSP, STSP, ADDI, SUBI, ANDI, ORI, XORI, NOTI, SLAI, SRAI, SRLI, BR, BMI, BPL, BZ, PUSH, POP

J-type	
OPCODE	Target Address
6 bits	26 bits

First 6 bits will be reserved for the opcode.
Next 26 bits will be reserved for the target address.

Range of target address: -2^{21} to $2^{21}-1$

Operations: CALL, RET, HALT, NOP

OPCODE Description

Operation	OPCODE(in decimal)
ADD	0
ADDI	1
SUB	2
SUBI	3
AND	4
ANDI	5
OR	6
ORI	7
XOR	8
XORI	9
NOT	10
NOTI	11
SLA	12
SLAI	13
SRA	14
SRAI	15
SRL	16
SRLI	17
LD	18
ST	19
LDSP	20
STSP	21
BR	22
BMI	23
BPL	24
BZ	25
PUSH	26
POP	27
CALL	28
RET	29
MOVE	30
HALT	31
NOP	32

Control Unit

opcode	ALU_{op}	ALU_{source}	WriteReg	MemWrite	MemRW	Res	Branch	St
0	0000	00	01	0	0	10	000	000
1	0000	10	11	0	0	10	000	000
2	0001	00	01	0	0	10	000	000
3	0001	10	11	0	0	10	000	000
4	0010	00	01	0	0	10	000	000
5	0010	10	11	0	0	10	000	000
6	0011	00	01	0	0	10	000	000
7	0011	10	11	0	0	10	000	000
8	0100	00	01	0	0	10	000	000
9	0100	10	11	0	0	10	000	000
10	0101	00	01	0	0	10	000	000
11	0101	10	11	0	0	10	000	000
12	0110	00	01	0	0	10	000	000
13	0110	10	11	0	0	10	000	000
14	0111	00	01	0	0	10	000	000
15	0111	10	11	0	0	10	000	000
16	1000	00	01	0	0	10	000	000
17	1000	10	11	0	0	10	000	000
18	0000	01	10	0	1	11	000	000
19	0000	01	00	1	1	01	000	000
20	0000	10	10	0	1	11	000	000
21	0000	10	00	1	1	00	000	000
22	0000	10	00	0	0	10	001	000
23	0000	10	00	0	0	10	010	000
24	0000	10	00	0	0	10	011	000
25	0000	00	00	0	0	10	100	000
26	0000	01	10	0	0	01	000	001
27	0000	10	11	0	1	11	000	010
28	0000	11	00	1	1	00	000	011
29	0000	11	00	0	1	00	000	100
30	0000	00	10	0	0	01	000	000
31	0000	11	00	0	0	00	000	000
32	0000	11	00	0	0	00	0	000

- **ALU_{op}**: opcode to be fed into the ALU, 0 for addition, 1 for subtraction, …, and 8 for shift right logical.
- **ALU_{source}**: 0 if second source operand is register, 1 if immediate value is of I1 type, 2 if immediate value is of I2 type, and 3 for J type.
- **WriteReg**: 00 if there is no register we store output in (or some special register), 01 if output is stored in “rd” register, 10 if output is stored in “rt” register, and 11 for “rs” register.
- **MemWrite**: 1 for store operations

- **MemRW**: 1 for both load and store operations as it is enable for BRAM in load and store operations.
- **Res**: 00 if no such output to be saved anywhere (or some series of sub-operations), 01 if output to be saved is from a register, 10 if output to be saved is ALU_{out} and 11 if it is accessed from memory.
- **Branch**: 0 if there is no branch operation, 1 for BR, 2 for BMI, 3 for BPL, 4 for BZ operations respectively.
- **St**: 0 for no stack operations, 1 for PUSH, 2 for POP, 3 for CALL, 4 for RETURN operations respectively.

Control Signals for some Instructions

1. ADD R1, R2, R3:
   aluop = 0 (for addition)
   alusc = 0 (for no immediate value)
   wrreg =1 (for writing back to rd register)
   memwr = 0 (since there is no writing to memory)
   memen = 0 (no need to enable memory operations)
   res = 2 (result is ALUoutput)
   br = 0 (no branch operation)
   st = 0 (no stack operations)

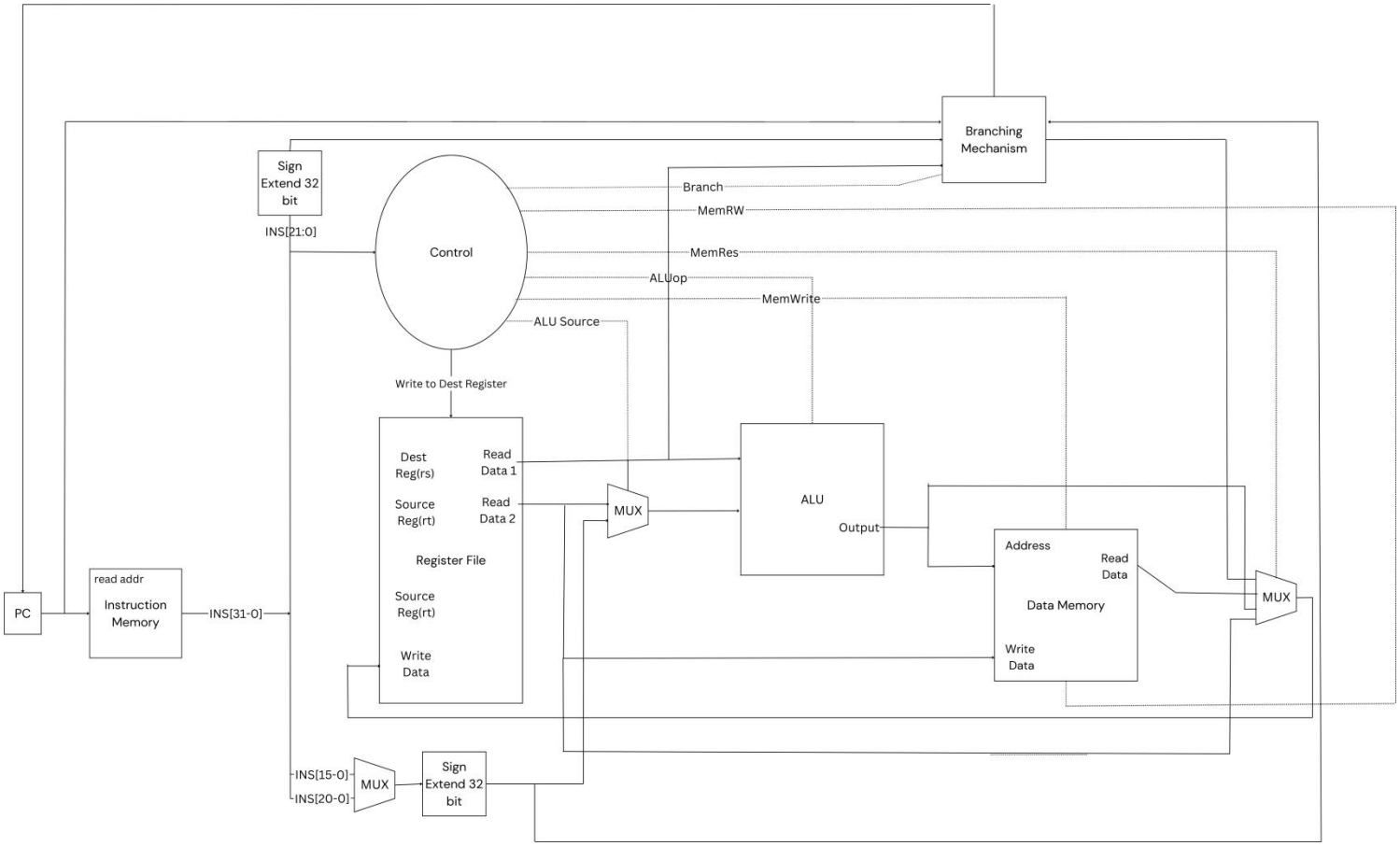
2. ADDI R1, 10
   aluop = 0 (for addition)
   alusc = 2 (for no immediate value of type I2)
   wrreg = 3 (for writing back to rs register)
   memwr = 0 (since there is no writing to memory)
   memen = 0 (no need to enable memory operations)
   res = 2 (result is ALUoutput)
   br = 0 (no branch operation)
   st = 0 (no stack operations)

3. LD R2, 10(R6)
   aluop = 0 (for addition)
   alusc = 1 (for no immediate value of type I1)
   wrreg = 2 (for writing back to rt register)
   memwr = 0 (since there is no writing to memory)
   memen = 1 (for read only)
   res = 3 (result is dataout from Memory)
   br = 0 (no branch operation)
   st = 0 (no stack operations)

4. BZ R8, -75
   aluop = 0 (for addition)
   alusc = 2 (for no immediate value of type I2)
   wrreg = 0 (no write back to any register)
   memwr = 0 (since there is no writing to memory)
   memen = 0 (no need to enable memory operations)
   res = 2 (result is ALUoutput)
   br = 4 (branch operation BZ)
   st = 0 (no stack operations)

5. MOVE R10, R5
   aluop = 0 (for addition)
   alusc = 0 (no immediate value involved)
   wrreg = 2 (for writing back to rt register)
   memwr = 0 (since there is no writing to memory)
   memen = 0 (no need to enable memory operations)
   res = 1 (result is read port of register bank)
   br = 0 (no branch operation)
   st = 0 (no stack operations)

Data Path Diagram



The dotted lines are the control lines, solid lines are the data lines.

RTL Micro-Operations

1. ADD R1, R2, R3

IF: ifsig = 1, npc = pc + 1

ID: ifsig = 0, rReg1 = ir[25:21], rReg2 = ir[20:16]

EX: A = rVal1(R2), B = rVal2(R3)

MEM: pc = npc

WB: wReg = ir[15:11], wVal = alu_out(writing alu_out to R1)

2. ADDI R1, 10

IF: ifsig = 1, npc = pc + 1

ID: rReg1 = ifsig = 0, rReg1 = ir[25:21], imm2 = ir[20:0]

EX: A = rVal1, B = signExt(imm2)

MEM: pc = npc

WB: wReg = ir[25:21], wVal = alu_out(writing alu_out to R1)

3. ST R2, 10(R6)

IF: ifsig = 1, npc = pc + 1

ID: ifsig = 0, rReg1 = ir[25:21], rReg2 = ir[20:16], imm1 = ir[15:0]

EX: A = rVal1, B = signExt(imm1)

MEM: pc = npc, mar = alu_out, datain = rVal2, memwr = memon = 1

WB: memwr = memon = 0

4. BZ R8, -75

IF: ifsig = 1, npc = pc + 1

ID: rReg1 = ir[25:21], imm2 = ir[20:0]

EX: A = pc, B = signExt(imm2), cond = (rVal1 == 0)

MEM: if(cond) pc = alu_out, else pc = npc

WB: -

5. MOVE R1, R2

IF: ifsig = 1, npc = pc + 1

ID: rReg1 = ir[25:21], rReg2 = ir[20:16]

EX: A = rVal1, B = rVal2

MEM: pc = npc

WB: wReg = ir[20:16], wVal = rVal1(copying R2 to R1)

Note: Instructions LDSP and STSP are similar to LD and ST instructions with special register SP as the register. POP and CALL can be implemented using multiple 32 bit instructions (2 and 3 respectively).