COMPUTER ARCHITECTURE
Assignment-1
Multi-cycle Processor

Author: Dhwanish
Roll Number: B211266CS

INTRODUCTION

NITC-RISC24 is a 8-register 16-bit computer system. It supports 3 machine instruction formats(R, I and J Type) and a total of 8 instructions. Instruction encoding is as shown in the image below.

***Instruction Encoding*:**

ADD:	0000	RA	RB	RC	0	00
ADC:	0000	RA	RB	RC	0	10
NDU:	0010	RA	RB	RC	0	00
NDZ:	0010	RA	RB	RC	0	01
LW:	1010	RA	RB	6-bit Immediate		
SW:	1001	RA	RB	6-bit Immediate		
BEQ:	1011	RA	RB	6-bit Immediate		
JAL:	1101	RA	9-bit Immediate			

** RA: Register A, RB: Register B, RC: Register C*
** All immediate values are signed*

Multi-cycle implementation of the architecture includes 4 stages each encoded with *state* variable as show below.
1) IF & ID : Instruction Fetch & Decode (Combined) (state = 00)
2) ALU : ALU Operation (state = 01)
3) MEM: Memory Access (state = 10)
4) WB : Register Write Back (state = 11)

MODULES

1. Instruction Memory

```
module instr_memory(
        input clk, zero, carry,
        input [1:0] state,
        input [15:0] addr,
        output reg [15:0] rd
    );
```

This module reads instruction corresponding to *addr* from instruction memory. This module fetches the instruction only if the current state is 00(IF & ID).

2. Register File

```
module regfile(
        input [1:0] state,
        input clk, reset, we,
        input [2:0] ra1, ra2, wa,
        input [15:0] wd, pc,
        output reg [15:0] rd1, rd2
    );
```

This module has the register file with 8-registers each 16-bit. Register values are initialized to random values to make testing process easier. In the IF&ID stage the *regfile* fetches values corresponding to ra1, ra2 & stores it in rd1, rd2. In the WB stage pc values will be stored to R0 register and if the *we (write enable)* is on, value in wd(write data) is written to register corresponding to wa(write address).

3. State Logic

```
module state_logic(
        input clk, reset,
        input [15:0] instr,
        output reg [1:0] state
    );
```

State control is implemented within this module. If the reset is enabled state is reset to 00(IF&ID). Next state is determined based on the opcode of the current instruction. State values goes through these values for each instruction,

1) R-Type Instruction: 00 => 01 => 11 (3 cycles)
2) I-Type Instruction: 00 => 01 => 10 => 11 (4 cycles)
3) BEQ Instruction: 00 => 01 => 11 (3 cycles)
4) JAL Instruction: 00 => 11 (2 cycles)

4. Next PC Logic

```
module next_pc(
        input clk, reset, jal, pcsrc,
        input [1:0] state, input [15:0] pc, instr,
        output [15:0] pcplus1, signimm, pcnext
    );
```

Next PC value is determined in this module along with pcplus1 (for storing when jal instruction) and signimm (for giving input to ALU if alusrc is enabled). Sign extention from 6-bit to 16-bit (for BEQ) and 9-bit to 16-bit (for JAL) is performed.
jal signal decides which sign extended value should be used as pcbranch. And finally if the pcsrc signal decides if the pc value should be pcplus1 or pcbranch.

5. Data Memory

```
module data_memory(
        input [1:0] state, input clk, we,
        input [15:0] addr, wd,
        output [15:0] rd
    );      You, 1 second ago • Uncommitte
```

Data memory module is only active for I-Type instructions. Memory is read from memory address addr to rd. If the we is enabled, data in addr is written as wd.

6. Controller

```
module controller (
        input [1:0] state, input [3:0] op, input [1:0] cz,
        input zero,
        output memtoreg, memwrite, branch,
        output pcsrc, alusrc,
        output regdst, regwrite, jal,
        output [1:0] alucontrol,
        output adc, ndz
    );
```

Controller module decides all the control signals. It instantiates decoder and aludecoder module.

7. Decoder

```
module decoder (
        input [1:0] state, input [3:0] op, input [1:0] cz,
        output memtoreg, memwrite, memread,
        output branch, alusrc,
        output regdst, regwrite, jal,
        output reg adc, ndz
    );
```

Decoder module decides control signals based on opcode and cz (if R-Type instruction). Adc and ndz signal will be enabled for the corresponding instructions using cz value.

Opcode	regwrite	regdst	alusrc	branch	memwrite	memread	memtoreg	jal
0000	1	1	0	0	0	0	0	0
0010	1	1	0	0	0	0	0	0
1010	1	0	1	0	0	1	1	0
1001	0	0	1	0	1	0	0	0
1011	0	0	0	1	0	0	0	0
1101	1	0	1	1	0	0	0	1

8. ALU Decoder

```
module aludecoder (
        input [1:0] state, input [3:0] opcode,
        output reg [1:0] alucontrol
    );
```

This module decides alucontrol which is used to choose what operation is to be done in ALU.

Opcode	ALU control	Operation
1010	2’b00	Add
1001	2’b00	Add
0000	2’b00	Add
0010	2’b10	Nand
1011	2’b01	Subtract

9. ALU

```
module alu(
        input [1:0] state, input [15:0] i_data_A, i_data_B,
        input [1:0] i_alu_control, output reg [15:0] o_result,
        output reg o_zero_flag, o_carry_flag
    );
```

ALU operation is decided by *i_alu_control*. *o_result* is computed if state is 01. Zero and Carry is also generated based on the operation.

Testing Instructions

The following set of instruction is run to test the implementation,

0250
0292
0dea
2c68
2449
2fa1
9aca
a8ca
b743
b942
0848
d3ff

Register values are initialized as shown below to test the implementation,
R0 = 0, R1 = 3, R2 = 4, R3 = 9, R4 = 24, R5 = 0, R6 = 65535, R7 = 65535

ADD

```
Time: 0, PC: 0 => ADD R1, R1, R2 ZERO: x, CARRY: x
Time: 20, Register values: R0 = 0, R1 = 3, R2 = 4, R3 = 9, R4 = 24, R5 = 0, R6 = 65535, R7 = 65535
Time: 20, Data read rd1: 3, rd2: 3
Time: 40, ALU Operation: A: 3, B: 3, result: 6, Zero: 0, Carry: 0
Time: 60, Writing Back: R[2] = 6
Time: 60, Updating pc=1
------------------------------------------------------------
```

R2 = R1 + R1 = 3 + 3 = 6
PC updated from 0 to 1

ADC

In the first ADC instruction 3 + 6 = 9 does not generate carry hence R2 is not written back.
But the second ADC instruction generates carry, so the aluresult is written back to R5.
PC is updated after completion of each instruction.

Images of execution of instruction is in the next page.

```
Time: 70, PC: 1 => ADC R1, R2, R2 ZERO: 0, CARRY: 0
Time: 80, Register values: R0 = 1, R1 = 3, R2 = 6, R3 = 9, R4 = 24, R5 = 0, R6 = 65535, R7 = 65535
Time: 80, Data read rd1: 6, rd2: 3
Time: 100, ALU Operation: A: 6, B: 3, result: 9, Zero: 0, Carry: 0
Time: 120, Updating pc=2
------------------------------------------------------------
Time: 130, PC: 2 => ADC R6, R7, R5 ZERO: 0, CARRY: 0
Time: 140, Register values: R0 = 2, R1 = 3, R2 = 6, R3 = 9, R4 = 24, R5 = 0, R6 = 65535, R7 = 65535
Time: 140, Data read rd1: 65535, rd2: 65535
Time: 160, ALU Operation: A: 65535, B: 65535, result: 65534, Zero: 0, Carry: 1
Time: 180, Writing Back: R[5] = 65534
Time: 180, Updating pc=3
------------------------------------------------------------
```

NDU

```
Time: 190, PC: 3 => NDU R6, R1, R5 ZERO: 0, CARRY: 1
Time: 200, Register values: R0 = 3, R1 = 3, R2 = 6, R3 = 9, R4 = 24, R5 = 65534, R6 = 65535, R7 = 65535
Time: 200, Data read rd1: 3, rd2: 65535
Time: 220, ALU Operation: A: 3, B: 65535, result: 65532, Zero: 0, Carry: 1
Time: 240, Writing Back: R[5] = 65532
Time: 240, Updating pc=4
```

~(R1 & R6) = ~(3 & 65535) = 65532 is written back to R5 & PC is updated.

NDZ

```
Time: 250, PC: 4 => NDZ R2, R1, R1 ZERO: 0, CARRY: 1
Time: 260, Register values: R0 = 4, R1 = 3, R2 = 6, R3 = 9, R4 = 24, R5 = 65532, R6 = 65535, R7 = 65535
Time: 260, Data read rd1: 3, rd2: 6
Time: 280, ALU Operation: A: 3, B: 6, result: 65533, Zero: 0, Carry: 1
Time: 300, Updating pc=5
------------------------------------------------------------
Time: 310, PC: 5 => NDZ R7, R6, R4 ZERO: 0, CARRY: 1
Time: 320, Register values: R0 = 5, R1 = 3, R2 = 6, R3 = 9, R4 = 24, R5 = 65532, R6 = 65535, R7 = 65535
Time: 320, Data read rd1: 65535, rd2: 65535
Time: 340, ALU Operation: A: 65535, B: 65535, result: 0, Zero: 1, Carry: 1
Time: 360, Writing Back: R[4] = 0
Time: 360, Updating pc=6
------------------------------------------------------------
```

First NDZ instruction does not write back since zero flag is not set by nand of R1 and R2. But the second instruction sets the zero flag hence R4 is written back the aluresult which is equal to 0.

LW & SW

```
Time: 370, PC: 6 => SW R5, R3, Imm: 001010  ZERO: 1, CARRY: 1
Time: 380, Register values: R0 = 6, R1 = 3, R2 = 6, R3 = 9, R4 = 0, R5 = 65532, R6 = 65535, R7 = 65535
Time: 380, Data read rd1: 9, rd2: 65532
Time: 400, ALU Operation: A: 9, B: 10, result: 19, Zero: 0, Carry: 0
Time: 420, MEM access: writing RAM[19]=65532
Time: 440, Updating pc=7
------------------------------------------------------------
Time: 450, PC: 7 => LW R4, R3, Imm: 001010  ZERO: 0, CARRY: 0
Time: 460, Register values: R0 = 7, R1 = 3, R2 = 6, R3 = 9, R4 = 0, R5 = 65532, R6 = 65535, R7 = 65535
Time: 460, Data read rd1: 9, rd2: 0
Time: 480, ALU Operation: A: 9, B: 10, result: 19, Zero: 0, Carry: 0
Time: 500, MEM Access: read addr=19, value read=65532
Time: 520, Writing Back: R[4] = 65532
Time: 520, Updating pc=8
------------------------------------------------------------
```

First Instruction is store instruction, which stores the value of R5(=65532) to address 19 of data memory. Load instruction is accesses the data memory address 19(9 + 10 = 19) and writes its content(65532) to R4.

BEQ

```
Time: 530, PC: 8 => BEQ R3, R5, Imm: 000011  ZERO: 0, CARRY: 0
Time: 540, Register values: R0 = 8, R1 = 3, R2 = 6, R3 = 9, R4 = 65532, R5 = 65532, R6 = 65535, R7 = 65535
Time: 540, Data read rd1: 65532, rd2: 9
Time: 560, ALU Operation: A: 65532, B: 9, result: 65523, Zero: 0, Carry: 0
Time: 580, Updating pc=9
-----------------------------------------------------------
Time: 590, PC: 9 => BEQ R4, R5, Imm: 000010  ZERO: 0, CARRY: 0
Time: 600, Register values: R0 = 9, R1 = 3, R2 = 6, R3 = 9, R4 = 65532, R5 = 65532, R6 = 65535, R7 = 65535
Time: 600, Data read rd1: 65532, rd2: 65532
Time: 620, ALU Operation: A: 65532, B: 65532, result: 0, Zero: 1, Carry: 0
Time: 640, Updating pc=11
-----------------------------------------------------------
Time: 650, PC: 11 => JAL R1, Imm: 111111111 ZERO: 1, CARRY: 0
```

First BEQ instruction does not alter PC since R3 and R5 have different values. Second BEQ instruction leads to branch taken since R3 and R5 both have value 65532. PC is updated to PC + Immediate(2)(9 + 2 = 11).

JAL

```
Time: 650, PC: 11 => JAL R1, Imm: 111111111 ZERO: 1, CARRY: 0
Time: 660, Register values: R0 = 11, R1 = 3, R2 = 6, R3 = 9, R4 = 65532, R5 = 65532, R6 = 65535, R7 = 65535
Time: 660, Data read rd1: 65535, rd2: 3
Time: 680, Writing Back: R[1] = 12
Time: 680, Updating pc=10
-----------------------------------------------------------
Time: 690, PC: 10 => ADD R4, R1, R1 ZERO: 1, CARRY: 0
Time: 700, Register values: R0 = 10, R1 = 12, R2 = 6, R3 = 9, R4 = 65532, R5 = 65532, R6 = 65535, R7 = 65535
Time: 700, Data read rd1: 12, rd2: 65532
Time: 720, ALU Operation: A: 12, B: 65532, result: 8, Zero: 0, Carry: 1
Time: 740, Writing Back: R[1] = 8
Time: 740, Updating pc=11
```

JAL instuction updates PC to PC + Immediate value and writes PC + 1 to R1
PC = 11 + -1 (Immediate value is 2’s complement) = 10