Computer Architecture Assignment - 1

Name: Chinthakommadinne Vinay Kumar

Roll No: B210554CS

Batch: CS03

A multi-cycle processor is a processor that completes an instruction over multiple clock cycles. It breaks down each instruction into multiple steps, and the number of steps depends on the complexity of the instruction. This allows simpler instructions to finish faster than more complex ones.

The 16-bit NITC-RISC24 processor is an 8-register, 16-bit computer system. It uses registers R0 to R7 for general purposes. However, register R0 always stores the program counter. This architecture also uses a condition code register, which has two flags: the Carry flag (C) and the Zero flag (Z). Both instruction memory and data memory are word-addressable (1 address location is 16-bit data/instruction).

We have general machine code instructions as follows.

R-Type Instructions					
OPCODE	RA	RB	RC	Unused	Condition (CZ)
4-bit	3-bit	3-bit	3-bit	1-bit	2-bit

I-Type Instructions			
OPCODE	RA	RB	Immediate
4-bit	3-bit	3-bit	6-bit

J-Type Instructions		
OPCODE	RA	Immediate
4-bit	3-bit	9-bit

The instructions we have are following:

***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*

Multicycle processor allows each instruction to be executed in more clock cycles, by dividing the instruction into several phases.

That means each instruction goes through these 5 phases

- Instruction Fetch - IF
- Instruction Decode - ID
- Execution - EX
- Memory - MEM
- WriteBack - WB

Although Some instructions goes through less phases like store it requires only first 4 phases, and jump instructions etc.

Multicycle processor consist of mainly ALU, Control Unit, datapath, register file, Program file and memory file.

Control Unit:

It coordinates phases for each instruction.

In a multi-cycle processor, it ensures that each instruction progresses through several distinct stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). During the fetch stage, it enables the Program Counter (PC) to retrieve the next instruction from memory. In the decode stage, the instruction's opcode is interpreted to determine the operation, such as arithmetic, memory access, or branching. The control unit then configures the ALU in the execute stage to perform the appropriate

operation. For memory-related instructions like load (LW) and store (SW), it manages memory reads or writes during the memory access stage. In the write-back stage, it controls whether the result should be written back to the register file.

Datapath:

The datapath handles the flow of data through the processor, ensuring that instructions are executed correctly by performing the required operations (fetching data, processing it, and writing back results). It works in sync with the control unit, which generates the necessary control signals to guide the datapath at each stage of instruction execution.

ALU:

Alu performs arthematic operations in a processor.

In our assignment we are doing add and nand operations as below.

```
always @(ALUop) begin
    // Default values for Carry and Zero
    Carry = 0;
    Zero = 0;

    case (ALUop)
        3'b000: begin  // ADD
            {Carry, Result} = A + B;
            Zero = (Result == 16'd0) ? 1 : 0;
        end

        3'b010: begin  // NDU (NAND)
            Result = ~(A & B);
            Zero = (Result == 16'd0) ? 1 : 0;
        end


    endcase
    $display("Carry %b Zero: %b", Carry, Zero);
    $display("At time %0d,A = %d, B = %d,alu_op = %d, alu_result = %d",$time,A, B,ALUop,  Result);
end
```

Program Counter:

It stores the address of next instruction to be executed.

Register file:

It stores the general-purpose registers used for holding data during instruction execution.

Instruction Memory:

It stores and gives instructions for execution.

1. ADD instruction

0000 001 010 011 000

Initially all registers are initialized to 4 as below

```
Register values at time 20:
R0 =     4
R1 =     4
R2 =     4
R3 =     4
R4 =     4
R5 =     4
R6 =     4
R7 =     4
```

After writeback stage the register contents are as follows

```
Register values at time 100:
R0 =     4
R1 =     4
R2 =     4
R3 =     8
R4 =     4
R5 =     4
R6 =     4
R7 =     4
```

As we can see the contents of R1 and R2 are added and stored in R3.

2. ADC instruction

0000 001 010 000 010

With no carry generated

Initial register values are as follows

```
Register values at time 20:
R0 =     4
R1 =     4
R2 =     4
R3 =     4
R4 =     4
R5 =     4
R6 =     4
R7 =     4
```

After Writeback the register contents are

```
Register values at time 100:
R0 =     4
R1 =     4
R2 =     4
R3 =     4
R4 =     4
R5 =     4
R6 =     4
R7 =     4
```

If carry is generated

Initial registers are as follows

```
Register values at time 20:
R0 = 32768
R1 = 32768
R2 = 32768
R3 = 32768
R4 = 32768
R5 = 32768
R6 = 32768
R7 = 32768
```

After addition the registers are updated as

```
Register values at time 100:
R0 = 32768
R1 = 32768
R2 = 32768
R3 =     0
R4 = 32768
R5 = 32768
R6 = 32768
R7 = 32768
```

3. NDU instruction

0010 001 010 011 000

initial registers are as follows

```
Register values at time 20:
R0 =     4
R1 =     4
R2 =     4
R3 =     4
R4 =     4
R5 =     4
R6 =     4
R7 =     4
```

After executing instruction

```
Register values at time 100:
R0 =     4
R1 =     4
R2 =     4
R3 = 65531
R4 =     4
R5 =     4
R6 =     4
R7 =     4
```

4. NDZ instruction

0010 001 010 011 010

If zero flag is not set (result is 0 then we set zero flag)

Initial registers are

```
Register values at time 20:
R0 =     4
R1 =     4
R2 =     4
R3 =     4
R4 =     4
R5 =     4
R6 =     4
R7 =     4
```

After execution of instruction

```
Register values at time 100:
R0 =     4
R1 =     4
R2 =     4
R3 =     4
R4 =     4
R5 =     4
R6 =     4
R7 =     4
```

If zero flag is set

Initial registers are

```
Register values at time 20:
R0 = 65535
R1 = 65535
R2 = 65535
R3 = 65535
R4 = 65535
R5 = 65535
R6 = 65535
R7 = 65535
```

After executing the instruction

```
Register values at time 100:
R0 = 65535
R1 = 65535
R2 = 65535
R3 =     0
R4 = 65535
R5 = 65535
R6 = 65535
R7 = 65535
```

5. LW and 6. SW instructions

We are here verifiying these by first storing the instruction into memory and loading that memory content into a register as follows

Initial registers are

```
Register values at time 20:
R0 =     8
R1 =     5
R2 =     8
R3 =     8
R4 =     8
R5 =     8
R6 =     8
R7 =     8
```

After executing store instruction R1 content 5 is stored into memory location 8+2 = 10.

So technically for executing load instruction we are trying to get that data value in memory location to R7 register

So after executing both instructions final register values are

```
Register values at time 200:
R0 =     8
R1 =     5
R2 =     8
R3 =     8
R4 =     8
R5 =     8
R6 =     8
R7 =     5
```

We can see that R7 has 5.