Computer Organization PA2

Implement a single cycle MIPS CPU

Student ID: B11107051

Name: 李品翰

Area: 34021.4 **Slack:** 4.3645

1. Module Implementation

Since Part I and Part II are subsets of Part III, only Part III is explained in detail below.

1.1. RF.v

- `RdAddr != 5'b0` ensures that register $0 is read-only.

```
always @(negedge clk)
    if (RegWrite && RdAddr != 5'b0)
        R[RdAddr] <= RdData;
```

1.2. IM.v

- Provides instruction words in big-endian format.

```
assign Instr = {
    InstrMem[InstrAddr],
    InstrMem[InstrAddr + 1],
    InstrMem[InstrAddr + 2],
    InstrMem[InstrAddr + 3]
};
```

1.3. DM.v

- In my design, `MemRead` is not required, `MemReadData` can be assign directly with combinational logic.

```
assign MemReadData = {
    DataMem[MemAddr],
    DataMem[MemAddr + 1],
    DataMem[MemAddr + 2],
    DataMem[MemAddr + 3]
};
```

1.4. Control.v

- The control unit is implemented using a large `case` statement. Any unknown OpCode will be considerd to be a `nop`.

```
default: begin // nop
    Reg_dst     <= 1'bx;
    Branch      <= 1'b0;
    Reg_w       <= 1'b0;
    ALU_src     <= 1'bx;
    Mem_w       <= 1'b0;
    Mem_to_reg  <= 1'bx;
    Jump        <= 1'b0;
    ALU_op      <= 2'bxx;
end
```

1.5. SimpleCPU.v

1. All multiplexers are implemented using ternary operators. For example:

```
assign Output_Addr =
    Jump ? {NextPC[31:28], Instruction[25:0], 2'b00} :
    (Branch & Zero) ? NextPC + {imm_extend[29:0], 2'b00} : NextPC;
```

2. `DM.MemRead` is simply assigned with a logic high:

```
DM Data_Memory(..., .MemRead(1'b1), ...);
```

2. Module Testing and Execution Results

Assembler

A assembler script (`assembler.py`) written by ChatGPT was developed to generate the `IM.dat` file from a `main.a` file with MIPS assembly inside.

Part I

- **main.a**

```
addu $25, $8, $9
addu $26, $7, $1
subu $27, $0, $1
subu $28, $0, $7
sll $29, $2, 30
sll $30, $2, 31
or $31, $3, $4
```

- **RF.dat**

```
0000_0000   // R[0]
0000_0001   // R[1]
0000_0002   // R[2]
0F0F_0F0F   // R[3]
3333_3333   // R[4]
```

```
F7F7_F7F7   // R[5]
7FFF_FFFF   // R[6]
FFFF_FFFF   // R[7]
FFFF_0000   // R[8]
0000_FFFF   // R[9]
...
```

- **RF.out**

```
...
ffffffff   // R[25]
00000000   // R[26]
ffffffff   // R[27]
00000001   // R[28]
80000000   // R[29]
00000000   // R[30]
3f3f3f3f   // R[31]
```

Part II

- **main.a**

```
# the main idea is to get the number 0x00114514
addiu $1, $0, 1284
ori $2, $1, 17428
addiu $3, $0, 16
ori $4, $3, 1
sw $2, 2($0)
sw $4, 0($0)
lw $5, 2($0)
```

- **RF.dat, DM.dat**

  They can be arbitrary because all values are initialized by `addiu`.

- **RF.out**

```
00000000 // R[0]
00000504 // R[1]
00004514 // R[2]
00000010 // R[3]
00000011 // R[4]
00114514 // R[5]
...
```

- **DM.out**

```
00 // Addr = 0x00
00 // Addr = 0x01
00 // Addr = 0x02
11 // Addr = 0x03
45 // Addr = 0x04
14 // Addr = 0x05
...
```

Part III

- **main.a**

```
# let DM.out be 00 01 02 03 04 ...
addu $1, $0, $0
addiu $2, $0, 128
addiu $3, $0, 1
sll $3, $3, 24
addu $4, $0, $0
sw $4, 0($1)
addiu $1, $1, 1
addu $4, $4, $3
beq $1, $2, 1
j 5
```

- **RF.out**

```
00000000 // R[0]
00000080 // R[1]
00000080 // R[2]
01000000 // R[3]
80000000 // R[4]
...
```

- **DM.out**

```
00 // Addr = 0x00
01 // Addr = 0x01
02 // Addr = 0x02
03 // Addr = 0x03
04 // Addr = 0x04
05 // Addr = 0x05
...
7e // Addr = 0x7E
7f // Addr = 0x7F
```

3. Datapath Rethinking

In PA1, the multiplier and divider were implemented using shifts, additions, and subtractions over 32 clock cycles. This means that an instruction must be locked until the operation finishes. To support this, the datapath needs these changes:

- **Instruction Locking:** Add one more multiplexer in the instruction address path so that the `Input_Addr` can directly go to the `Output_Addr`, keeping the instruction fixed until the operation is done.
- **ALU Update:** Add `srl` (shift right logical) operation into the ALU.
- **New Registers:** Add 2 registers, HI and LO, to store the full 64-bit result of these operations.

4. Conclusion and Insights

In this assignment, I successfully built a single-cycle MIPS CPU with a basic instruction set. At first, looking at the architecture diagram, I thought the project would be very challenging. However, once I understood the role of each component, completing each part became much clearer.

This report is also my first experience writing in LaTeX. Compared to Microsoft Word, LaTeXsaved me a lot of time with formatting so that I could focus on writing the content. Unfortunately, since LaTeX's support for Traditional Chinese is limited, I had to write this report in English—but that also gave me a chance to practice my English skills.