HM4 Report

Tong Zhen 120090694

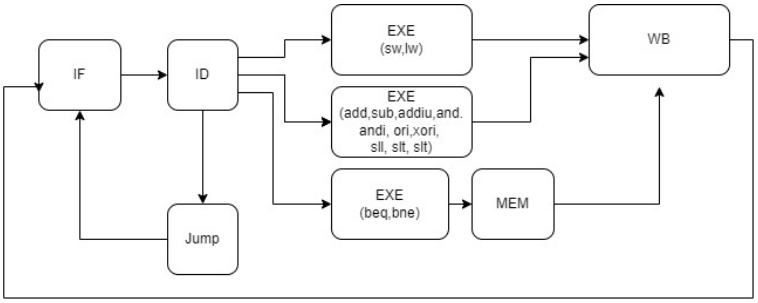
5-stage MIPS Pipeline CPU Description

Different from single-cycle CPUS, multi-cycle CPU divide the entire CPU execution process into 5stages. Each stage is completed with a clock, and then the next instruction is executed.

When the CPU processes instructions, it generally goes through the following stages:

1. **Fetch instruction (IF):** The Program Counter gets the instruction from an Instruction RAM. At the same time, PC automatically increases a word to generate an instruction address for the next instruction. But when encountering branch or jump instruction, the controller "transfer address" into the PC
2. **Instruction decoding (ID):** Decode the instructions obtained in the instruction.
3. **Instruction execution (EXE):** According to the operation control signal obtained by instruction decoding, the specific thing of the instruction is executed, and then transferred the result back.
4. **Memory access (MEM):** All operations that need to access memory will be performed in this step.
5. **Write back (WB):** The result of instruction execution or data obtained from accessing memory is written back to the corresponding destination register.

Different instructions can be executed in a flow chart



p1. An abstract data flow chart

In fact, the key for the CPU is the unit to control the flow, we call it the control unit. The signal needed are as follows.

Signal	0	1
ALUSrcA	Data1 output from the register file	From the shift number `shamt`, at the same time, apply (zeroextend)sa, i.e. `{{27{1'b0},sa}`
ALUSrcB	Data2 output from the register file	The immediate number from the `Sign` or `Zero` extension

RegDst	Register file write address is `Rt`	Register file write address is `Rd`
Jump	PC= jump	PC = PC+4 OR `beq` offset address
MemtoReg	Output from ALU operation results	Output from Memory
ALUOp	to be detailed —	—
MemWrite	Memory write invalid	Memory write is enabled
ExtSign	zero extension	sign extension
RegWrite	Invalid write to register group	Register group write is enabled

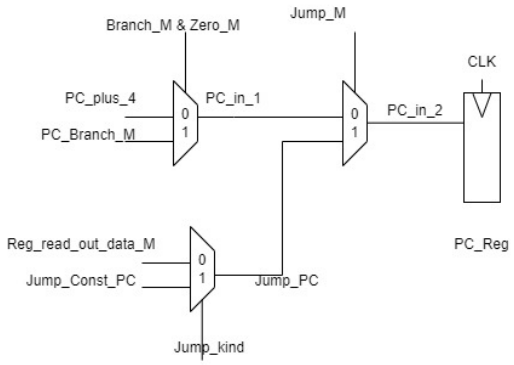
Problem break down

all the graph below use exact pin name in my program, better for you to understand

Description of related components and pins

signal with _D	wire in the ID stage
signal with _E	wire in the EXE stage
signal with _M	wire in the MEM stage
signal with _W	wire in the WB stage

PC source



p2. PC source by Multiplexer

As the Graph shows, the next PC can be of three kinds, normal PC-plus-4, branch, and the jump target. To be more specific, the Jump target can be rt (`jr`) and constant (`jal`, `j`)

ALU

ALU is the arithmetic logic operation unit with the following functions:

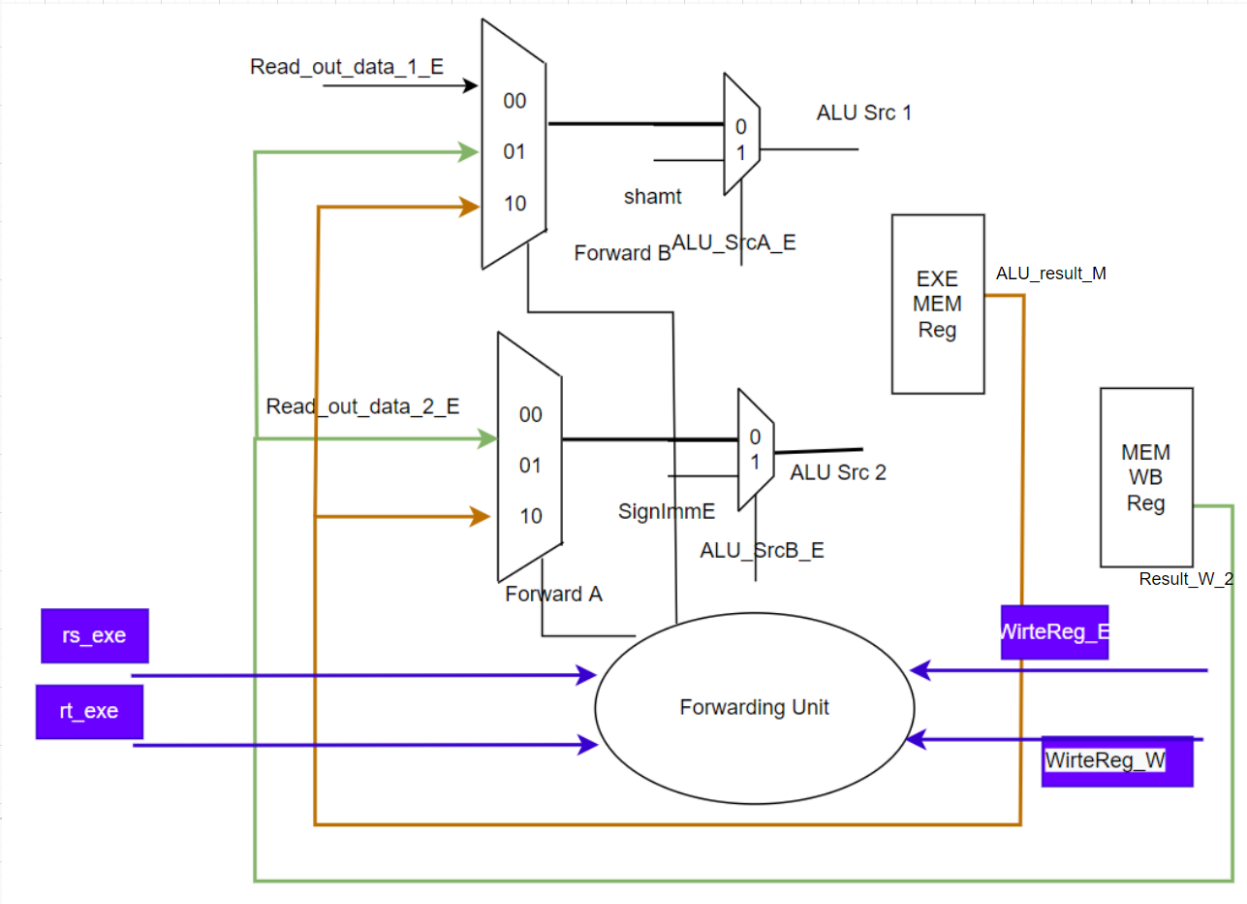
```
ALU_OP_ADD  // add the two input
ALU_OP_SUB  // subtract the two input
ALU_OP_SRL  // logic shift right
ALU_OP_SRA  // arithmetic shift right
ALU_OP_SLL  // logic shift left
ALU_OP_OR   // logic or
ALU_OP_AND  // logic and
ALU_OP_NOR  // logic nor
ALU_OP_SLT  // set less than
ALU_OP_XOR  // logic xor
```

It need two 32-bit input data, a 4-bit control signal `ALU_OP_` and can output a 32-bit data.

ALU op can be obtained according to the truth table of the instruction.

ALU source

In order to understand the ALU source, first let's ignore the three-way multiplexer, three-way multiplexer selects the 00 input. The first data input into ALU is the data 1 read out by register file, or the displacement of `SLL`, `SRL`, or `SRA` *shamt*. The second data entered into the ALU is the data 2 read out of the Register file, or the immediate number extended by the extend unit.



p3. ALU source with the Forwarding Unit

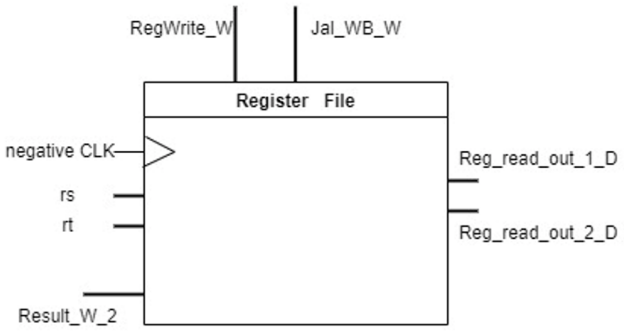
Extend Unit

p4. extand unit

The [15:0] of instruction can be an input of the ALU. Notice that all the instructions extend by the signed rule, except for the logic instructions (and, or...).

Register file

This module has 32 registers for a 32 bit. The register file takes input from InstructionMemory and outputs data from the corresponding register to read the data in the register



p5. Register file

The negative clock is used to judge whether it is the correct time to store data. Using negative clock to avoid the read-write hazard. Specifically, the `Jal` demand us to store the next `PC-plus-4` into the Register File and defaultly store in `$ra` . Both the `RegWrite_W` and `Jal_WB__W` is a certificate to write in data.

Forwarding Unit

Now let's look at picture P3 again. Our bypass unit handles data risk in 2 cases

1. The data (EXE/MEM) after ALU calculation in EXE phase is connected to the register (ID/EXE) before ALU calculation in EXE phase as Rs or Rt input.
2. Rd before MEM memory is stored is used as RS input in EXE stage

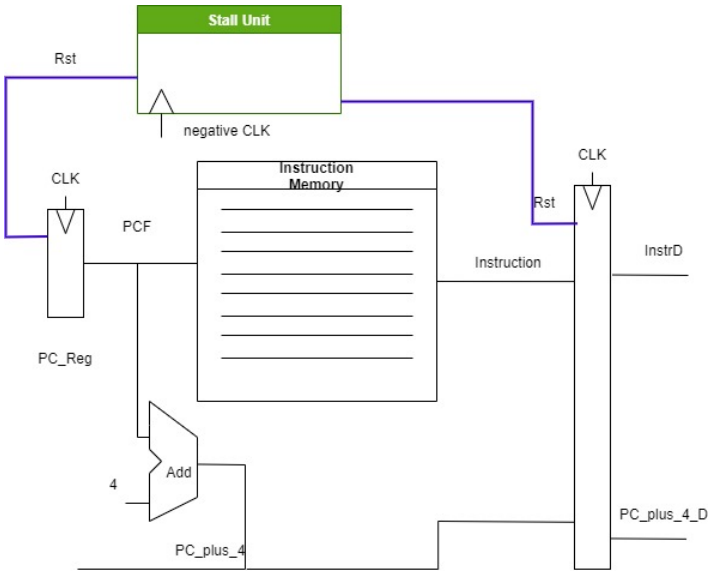
Because there are 2 data inputs for ALU and 2 situations above, there are 4 combination

EX/MEM.RegisterRd = ID/EX.RegisterRs
EX/MEM.RegisterRd = ID/EX.RegisterRt
MEM/WB.RegisterRd = ID/EX.RegisterRs
MEM/WB.RegisterRd = ID/EX.RegisterRt

Notice, if the `$zero` is involved in the data hazard as a rd, we can ignore the register file change, because the register `$zero` always 0.

Stall Unit

To deal with unexpected instructions, the pipeline CPU needs a stall unit. In my design, the stall unit is set in the IF stage. It will identify the `beq/bne` , `jal/jr/j` and `lw` .The PC_register module generates 4 bubbles and decreases the bubble by the clock. In the first bubble, the stall unit will remain the `IF_ID_Reg` to receive one instruction from IF stage. After that, it will give both the `PC_Reg` module and the `IF_ID_Reg` zero signal to stall them. And in the last bubble, the stlall unit will allow the `PC_Reg` to receive new PC.



The principle behind blocking the Control Unit is to prevent the control signal of branches and jumps from being reserved for subsequent normal commands

The principle of blocking `IF_ID_Reg` is to prevent the opcode and `funct` of the Control signal from inheriting the branch and jump instructions and affecting subsequent normal instructions.

Therefore, it is a negative edge clock trigger module in order to better control `IF_ID_Reg` and `PC_Reg`

CPU performance

The clock I used for pipeline CPU is as below.

test	1	2	3	4	5	6	7
clock size	61	16	22	18	244	70	55

We can observe that it dosn't seem to have less clock number than the simple single cycle CPU. However, with each clock interval become shorter (we only need to do a small thing each stage in pipeline), the total time amount can be as 4-5 times less as non-pipeline CPU.

Testing

Input insturction file name nistructions.bin and all 8 tests can pass, with a **RAM.txt** represent the Memory after the program run.