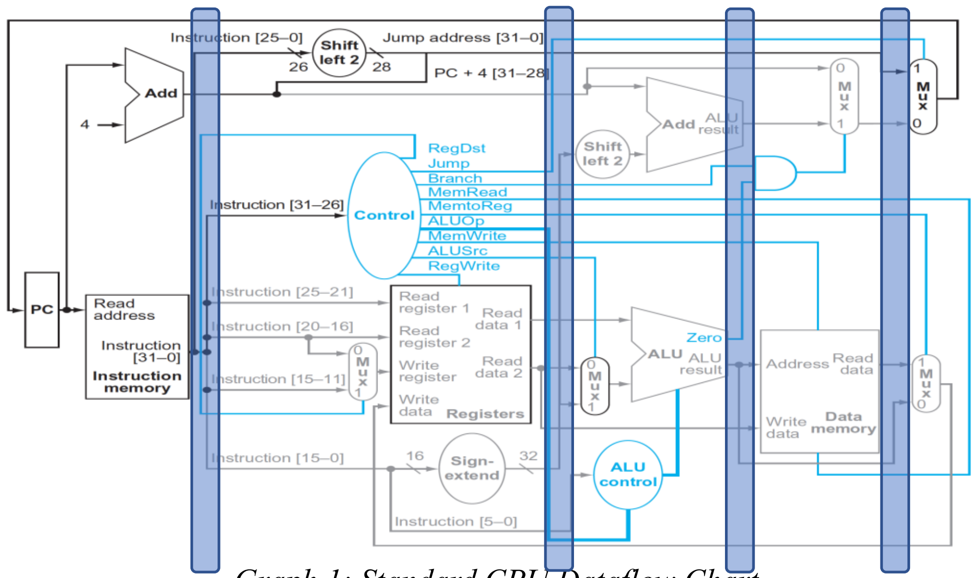
5-stage Pipelined CPU project report

1. Overview

In computer architecture, RISC processors have 5-stage instruction pipeline to execute all the instructions in the RISC instruction set. Following are the 5-stage of **MIPS CPU pipeline** with respective operations:

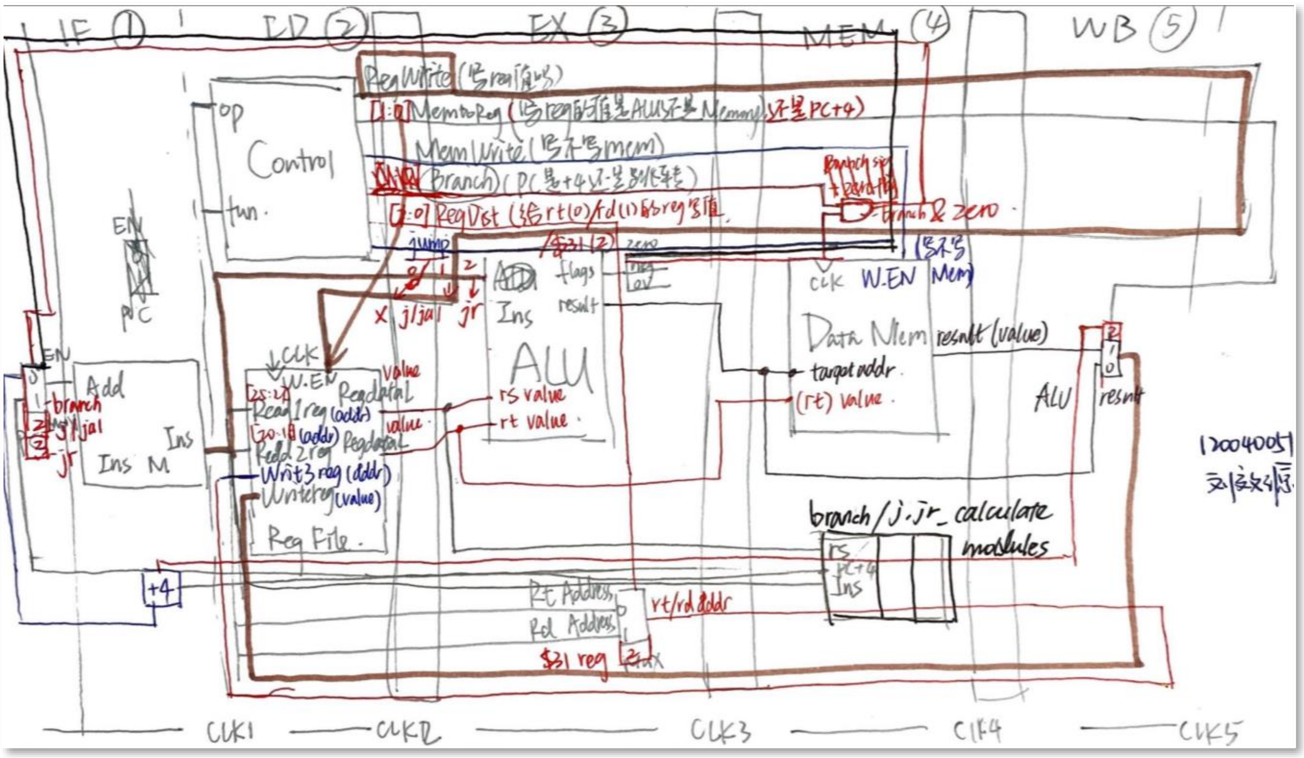
1. Stage 1 (Instruction Fetch: IF)
   In this stage the **PC** reads instructions from the address in the **Instruction memory** whose value is present in the program counter.
2. Stage 2 (Instruction Decode: ID)
   In this stage, instruction is decoded and the **Register File** is accessed to get the values from the registers used in the instruction. Signal
3. Stage 3 (Instruction Execute: EX)
   In this stage, **ALU** operations are performed. The operation code and function code in MIPS instruction are sent to the **Control Unit**, which recognizes the type of an instruction and generates the according control signals (See Appendix)
4. Stage 4 (Memory Access: MEM)
   In this stage, memory operands are read and written from/to the Main Memory according to the signal from the control unit.
5. Stage 5 (Write Back: WB)
   In this stage, computed/fetched value is written back to the register present in the operands according to the control signal.



Graph 1: Standard CPU Dataflow Chart

2. Data flow chart

The chart below illustrates the implementation of the CPU pipeline in project 4. The detailed clarification will be discussed the **logistics section**.



Graph 2: Data flow chart for Project 4

3. Logistics

In this part, the interpretation of the data flow chart and the actually implementation in the code are going to be introduced.

The project is divided into different modules including the CPU.v which conducts the pipeline and a top module test_CPU.v which tests the CPU execution and generate the output.txt. Note that the makefile is also provided in the project.

3.1 Preparation Stage & IF stage

In the Preparation Stage, there is a multiplexer `PC_input_mux_4_to_1` receive the control signals branch and jump[1:0] to select the next PC address from the PC+4, branch_address, j/jal_address, and jr_address. Note that expect for PC+4 which is sent back from the ID stage, other addresses all come from the MEM stage.

In IF stage, PC reads the input and generates the output address (initially set to 0) to fetch the instruction from the `InstructionRAM`.

PC+4 and fetched address are fed into the IF_ID stage registers (latches) for the outputs.

3.2 ID stage

In this stage, `opcode[5:0]` and `funct[5:0]` are sent to the Control Unit to generate 7 signals for each instruction, namely:

```
.RegDst[1:0],     //Register to write: 0 for rt, 1 for rd, 2 for $ra(jal)
.MemWrite,        // whether to write memory
.MemRead,         // whether to read memory, set as constant 1
.Branch,          // whether to set PC to the branch address
.Jump[1:0],       // Jump signals: 0 for not jump, 1 for j/jal, 2 for jr
.MemToReg[1:0], // value to write the reg: 0 for ALU result, 1 for
MEM, 2 for rs register
.RegWrite,        // whether to write the register
```

Register File is fed with the rs, rt addresses, and `RegWrite`, `write_addr[4:0]`, and `value_to_write[31:0]` which are going to be sent back from the WB stage and output two register values, rt_value and rs_value.

PC is added with 4.

These two values, PC_plus4, instruction, together with the 7 control signals are fed to the ID_EX bank – stage registers for outputs.

3.3 EX stage

For simplicity, I reused my ALU module in the last project which has the instruction as the input to handle the operation and generate ALU result and flags (in this project, only zero flag is considered).

The multiplexer `MUX_writeReg_addr` is implemented here receiving the `RegDst[1:0]` signal to determine the address for the register to write.

ALU output, zero flag, the rest of the control signals, rt_value, PC_plus4, and the determined register address is fed the EX_MEM stage registers for outputs.

3.4 MEM stage

In this stage, the MainMemory Unit receives the signals `MemWrite` signal, ALU output and the rt_value to read/write the memory and **constantly** output the Memory result with regard to the ALU output (as the fetch address).

Three Branch_and_jump_calc modules (j/jal, beq, jr) are implemented here using the necessary inputs to calculate the target addresses and send back to the `PC_input_mux_4_to_1` at the Preparation Stage. Branch signal AND zero flag are assigned to the wire `branch_and_zero_sig to` indicate valid branch signal. `branch_and_zero_sig and jump[1:0]` are sent back to the `PC_input_mux_4_to_1` from here.

3.5 WB stage

The multiplexer `MUX_writeReg_value` is implemented here to select the value to write the register in the Register File module basing on the `MemToReg[1:0]` signal. The determined value is sent back to the Register File as `value_to_write[31:0],` together with the `write_addr[4:0]` signal generated at EX stage. `RegWrite` is also sent back to the Register File to enable/disable the register writing.

4. Implement details

1. ALU is from the last project with slight modifications, so there is no ALU Control Unit in the implementation.
2. MemRead is a useless control signal here.
3. branch/jump signals are sent back from the MEM stage.
4. PC = PC + 4 occurs at the IF stage.
5. To transfer address from byte space to word space, the address is always >> 2 before fetch.
6. A clock module is implemented separately to generate clock signal.
7. The program will print the MainMemory on the console and also generate output file named ***output.txt***
8. The instruction 32'hffffffff is regarded as the instruction the end the program.
9. The project should contain 15 .v files and 1 makefile and 1 pdf file. (Please contact me if any of the project files are missing)
10. **The program is expected to pass test 1, 5 (partial), and 6 (partial).**