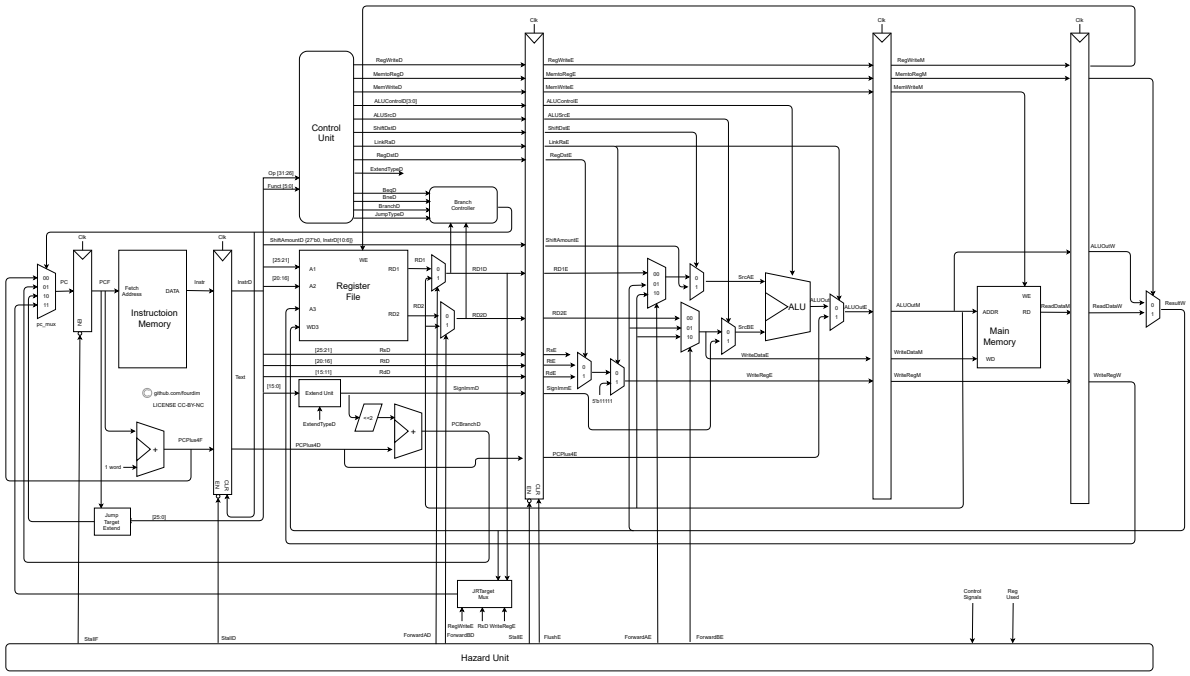
Project3 Report

1. Overview



© fourdim under CC-BY-NC license

The design is all showed in the picture above. This is an SVG picture, you can zoom in to view the details.

It is relatively easy for you to read my codes by using this diagram.

2. Idea

Use pipeline registers to divide the whole CPU into 5 stage, so that all the signal that are used can be stored in the pipeline registers. And the hazard unit will use forward and stalls to resolve conflictions between instructions.

We first implement all the submodules like Register File, Control Unit, Branch Controller, ALU, and so on. And then we include it in a top module called CPU, where we need to use wires to connect each sub modules.

And then connect the control signals and the registers' address into hazard unit. If hazards are detected, then use forward or stall to resolve the issues.

We will use the test bench to generate the clock pulses, and send it to the CPU. Also, the cycles can be calculated in the test bench by just uncomment the code:

```
$monitor("%h %d", cpu.InstrD, i);
```

If the CPU gets the instruction 32'hffffffff, give some another cycles to make all the data saved and then output the stop signal to let the test bench stop the clock.

Sample Execution of my CPU

```
lw $t1, 0($t2)
addi $t3, $t1, 1
```

First, we fetch the instruction, and enter the ID stage.

The control unit will decode the lw with: // comments are in the codes

```
RegWriteD <= 1;
MemtoRegD <= 1;
MemWriteD <= 0;
BranchD <= 0;
ALUControlD <= 2;
ALUSrcD <= 1;
RegDstD <= 0;
BeqD <= 0;
BneD <= 0;
LinkRaD <= 0;
JumpTypeD <= 0;
ShiftDstD <= 0;
ExtendTypeD <= 0;
```

The register file will give $t2's value and send it to the next stage. When the clock pulse, addi enter the ID, and lw enter the EX.

The control unit will decode the addi with:

```
RegWriteD <= 1;
MemtoRegD <= 0;
MemWriteD <= 0;
BranchD <= 0;
ALUControlD <= 2;
ALUSrcD <= 1;
RegDstD <= 0;
BeqD <= 0;
BneD <= 0;
LinkRaD <= 0;
JumpTypeD <= 0;
ShiftDstD <= 0;
ExtendTypeD <= 0;
```

The hazard controller will detect the MemtoRegE and $t1 should be used after lw and then ready to stall. Due to the stalling, addi will remain in ID stage, whereas lw will enter the MEM stage. In the next stage, the hazard unit will forward the result gets from the memory to the addi in the EX stage.

```
lw $t1, 0($t2)  IF  ID  EX  MEM WB
nop                 IF  NOP NOP NOP NOP
addi $t3, $t1, 1        IF  ID  EX  MEM WB
```

*Forward from lw MEM to addi EX.

3. Implementation

3.1 Hazard Unit

As you can see in the hazard unit, we can use forward and stall to resolve hazards.

For data hazards, we need to check the registers used between stages to forward data we need.

For control hazards, I move the branch controller to ID stage, it will save some cycles when branch. Although, we still need to judge where to branch when EX, we can use stall to resolve it. Compared with implementing branch controller in EX or MEM stage, it saves much more cycles.

For structural hazards, as we already have forwarding unit, what we only do is to stall one cycle but not two cycles.

And when the target register is $zero, there is no need for us to forward as it cannot be assigned.

3.2 Control Unit

Both i-type and r-type instructions will have the same ALU Control signal. It will reduce the complexity of ALU.

3.3 Branch Controller

Branch controller will decide whether to branch. As we have four sources, the branch controller will decide which source will be chose by analyzing the inputs.

3.4 Register File

Do not assign value when $zero.

3.5 ALU

Eight types of ALU Control. It will not have the zero flag, as the branch is resolved by branch controller in order to save cycles. And the ALU implementation is already specified in the last report.

3.6 Pipeline Registers

When positive edge CLK, give the value to the next stage.

If not EN, do not give the value and maintain the status.

If CLEAR, use the status of nop, and remove all the current status.

4. Build, Run, and Test

```
$ ls -l
total 100
-rwxrwxrwx 1 fourdim fourdim   360 May 14 20:49 Adder.v
-rwxrwxrwx 1 fourdim fourdim  1629 May 13 22:18 ALU.v
-rwxrwxrwx 1 fourdim fourdim   884 May 15 15:12 BranchController.v
-rwxrwxrwx 1 fourdim fourdim 17451 May 14 23:22 ControlUnit.v
drwxrwxrwx 1 fourdim fourdim  4096 May 14 21:24 cpu_test
-rwxrwxrwx 1 fourdim fourdim  5473 May 15 13:47 CPU.v
-rwxrwxrwx 1 fourdim fourdim   476 May 14 21:01 Extend.v
-rwxrwxrwx 1 fourdim fourdim  4339 May 14 21:19 HazardUnit.v
-rwxrwxrwx 1 fourdim fourdim  1519 May 12 11:16 InstructionRAM.v
-rwxrwxrwx 1 fourdim fourdim 20669 May 11 21:58 MainMemory.v
```

```
-rwxrwxrwx 1 fourdim fourdim  3178 May 15 10:56 Makefile
-rwxrwxrwx 1 fourdim fourdim  1877 May 15 13:31 Mux.v
-rwxrwxrwx 1 fourdim fourdim  5336 May 15 13:43 PipelineReg.v
-rwxrwxrwx 1 fourdim fourdim  1378 May 11 22:00 RegisterFile.v
-rwxrwxrwx 1 fourdim fourdim   728 May 15 13:45 test_CPU.v
```

4.1 Build

```
$ make
[Build] Building the project...
[Build] The generated file cpu is available at /mnt/d/workspace/mips-cpu/cpu
```

4.2 Run

```
$ make run
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
```

4.3 Test

If you want to use `make test`, there is no need to run `make`.

If you want to use other files to test, it may not works. You may need to write your own scripts.

It uses diff to check whether the content in `output.txt` and `cpu_test/DATA_RAM?.txt` are the same.

```
$ make test
[INFO] The provided tests are all passed on my computer, if there is something
wrong when you are running this test, check whether the output.txt is in the
CRLF mode.
[Build] Building the project...
[Build] The generated file cpu is available at /mnt/d/workspace/mips-cpu/cpu
[Test] Test1
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
[Test] Test2
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
[Test] Test3
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
[Test] Test4
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
[Test] Test5
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
[Test] Test6
```

```
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
[Test] Test7
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
[Test] Test8
[Run] Running...
WARNING: ./InstructionRAM.v:38: $readmemb(instructions.bin): Not enough words in
the file for the requested range [0:511].
[Test] Tests complete with no error.
```

4.4 Clean

```
$ make clean
[Clean] Cleaning...
[Clean] Done
```