Project 3 Report

Outline:

1.Understanding of Project

2.Implementation of pipeline

3.Implement of Sub parts

4.Tools and other resources used.

1. Understanding of project.

This project requires us to write a five-stage pipeline mips CPU. To accomplish this, our project should have two parts: implement of pipeline, including the pipeline register and their behavior.The clock controls the action of pipeline. At the posedge of clock, the pipeline will pass its data to next sub module; at the negative edge of clock, it will receive the data from previous sub part and prepare data to be used in the next clock cycle. The other is five sub parts contained in a cpu, which takes previous pipeline register as input and output its result to next pipline regiester. Different pipeline register may have different length, and each position on the register have a fixed meaning, thus we can use bit location to identify different information. The detailed division will be talked in the comming sections.Besides, we also have two memory modules: InstrucionRAM, which stores all instructions and MainMemory, which stores data.Initially, when we start a cpu, it will read instructions stored in local file (e.g instructions.bin) and read all content into instruction memory. Then we start the cpu by fetching first instruction from offset=0. The we will increase the program counter by 4 each time unless branching. When fetched terminate instruction 0xffffffff, we will send a signal to tell the system all instruction finished. The system will wait till all stage finished, then finish the program.

2.Implememnntation of pipeline.

Within the top module, there's totally five pipeline registes: IFID, IDEX, EXMEM, MEMWB and WBIF, which is named by parts before and after it. Each register can be further divide into two parts: now and result.Each physical register have a bus to dirve. The "\*result" drives the output wire connected with the previous sub module; the "\*now" drives the input wire connected with the next sub module.

At every negative edge of clock, the data in "\*result" reg will be assigned to the corresponding "\*now" register. In this stage, there's no operation in sub modules. The top module provides all data needed in the comming rising edge. At the positive edge of clock, the data in "\*now" register will be read in.

Whenever the Top module receives the "finished" signal from last stage, it will wait one cycle until the finish of the final command. Then the program terminate.

3.Design and implement of Sub modules.

Our CPU have five distinct modules: IF, ID, EX, MEM and WB. All of them takes a pipeline register as input. Some of them may also take other input. All of them will use an output wire to connect to next pipeline register, some may also connect to other modules directly.

(1)Instuction fetch (IF)

The IF is to fetch the instruction of giving address.It keeps a program counter(pc) internally. It takes clock and a 32-bit pc as input. This input pc (denoted as pc2) is for branching. It also receives an branching signal to decide whether it will use the PC stored internal. The local pc is either replaced with the branching pc2 or pc+$, then it is connected to the InstructionRAM and pass the fetch address to it by the local PC. It receives the result from IRAM with an register inst. Then it outputs both current pc and the fetched instruction to the next pipeline.

Whenever the fetched address is 0xffffffff, the "finished" bit in the output is set to one. This means all instructions have been fetched. This signal will be passed through the pipeline and terminate each sub module sequentially. We will talk this in the latersection.

The layout of output is:

IF/ID[64:0]:

[31:0]:instruction;

[63:32]:pc+4.

[64]:finished

(2)Instruction decode(ID)

ID receives the data from IF and decodes the instruction. It also keeps the register file of the CPU, which have 32\*32-bit registers. The corresponding register can be accessed using regfile[pointer]. ID will generate the controal signals of each comming parts through function "control". The rule of control generation is same as in lecture slides.

There are overall 10 bit control signals. Their sequence and meaning as well as function modules are listed as follow:

///used in wb:ctr[4](jump),ctr[1](regwrite)and ctr[0](memtoeg)

///used in ex:ctr[9](regdst),ctr[8](alusrc),ctr[7:6](aluop)

///used in mem:ctr[5](branch),ctr[3](memread),ctr[2](memwrite)

Original control signal takes only op as input. However, there's limitation when dealing with andi/ori/xori and slt instructions. There fore we also take these special conditions into consideration.

Besides the control signal, we also get the address of RegA, RegB, possible target register address, immediate number, shift-amount, and put them into the output register.

Since ID keeps the register file, it will also take write\_enable, write\_target and write\_data as input, which will store data to the registers.

The layout of output is:

ID/EX:

[31:0]: pc+4

[63:32]:sign\_extend inst[15:0]

[68:64]:inst[15:11]

[73:69]:inst[20:16]

[105:74]:regB data(regA\_and\_regB[31:0])

[137:106]:regA data(regA\_and\_regB[63:32])

[147:138]:10bit control signal

[148] FINISH

[180:149]:inst

Although there's no necessity to pass instruction forward, we will do so because it helps us tracing the flow of instruction.

(3).Execution(EX)

EX keeps the ALU of the CPU. the main part of ALU is same as project 2, except that we replaces some local verialbes. We use alusrc(ctr[8]) to decide which input(regB or imm) should be the data source of rt. The rs is fixed to the regA. Initially we have tried to use aluop to construct our alu, but this alu can do only a limied amount of jobs. Thus we still use the high-level alu and disert the aluop(ctr[7:6]).

When execution completed, we will get a 35-bit output: one 32-bit normal alu result and 3 bits for flags. We will keep only the zerro bit. Also, EX will decide the address of register writing with a mux and regdist(ctr[9]). Ex will also shift the imm by 2 and add it with pc+4 of input pc in "pc+4", and give the result to the corresponding part.

The layout of output is:

EX/MEM:

[31:0]:pc+4+imm<<2(IDEX[31:0])

[36:32]:MUX(IDEX[4:0],IDEX[9:5]) write reg

[68:37]:IDEX[105:74](REGB DATA) write data

[100:69]:ALU result

[101]:zerobit

[107:102]:IDEX[143:138](CTR[5:0])

[108] finished

[140:109]:INST

(4).Memory operation(MEM)

MEM is connected to the MainMemory module. It takes the alu result as data write address. If the Memory write is enabled(ctr[2]), then it will write the target address and data to a bundle "edit\_series", give it to the memory, and let the Memory to decode the module. If the Memory read is enabled, it will fetch data at target address and give it to next stage. When receives the finished signal from pevious stage, it will print all data in memory to screen, because there cound't been more instruction to write to it.

The layout of output is:

MEM/WB:

[4:0]:EX/MEM[36:32]

[36:5]:READ DATA

[68:37]:ALU RST

[71:69]ctr[4,1,0]

[72] finish

[104:73] INST

(5).Write back (WB)

WB takes the memory data, reg dist as input. Whenever regwrite(ctr[1]) is enabled, it will pass the corresponding data and address to ID module, at where the data will be write to the memory.

It also takes the "pc" selected in EX stage as input, when branching(ctr[5]) is enabled, it will give this new pc to the IF module, at where this pc will replace the local pc.

4.Tools and referemces:

Icarus Verilog version 10.1

Icarus Verilog runtime version 10.1

GTKWave Analyzer 3.3.86

JetBrains IntelliJ IDEA with Verilog extension for coding

MIPS32® Architecture For Programme Volume II: The MIPS32® Instruction Set by MIPS Technologies ,Inc, to check format and code of instruction, meaning, implementation.

online mips converter to check translation result http://mipsconverter.com/instruction.html?optradio=on