MIPS

OneCycle MIPS

The OneCycle MIPS in OneCycle.srcs supports the instructions including:

add，sub，and，or，slt，addi，andi，ori，slti，sw，lw，j，nop，beq，bne, jal, jr, sra, sll, srl, which sum up to 20.

So you can implore many operations like shift, stack, call and ret.

Hints:

If you want to simulate the top.v, then you'll find that all the results are "XX", because the clock divider I set is very large, i.e. q[26], q[23], so you won't see any change because the clock offered to CPU doesn't reach the positive edge. An easy way to fix this problem is to change the clock divider to smaller one, like q[0], and when you run the codes on the board, you should change it back.

MultiCycle MIPS

The MultiCycle MIPS still need to be improved, since it only pass the ad hoc testing in **Harris's Digital Design Computer Architecture**' s 7.4

2019.4.19

successfully pass most Oxer's interesting tests, including **quick_multiply.out** which needs srl, sll and **factorial.out** covering jal, jr. Next time, I'll try the Pipeline and I want to add the FPU instruction set.

Pipeline

2019.5.10

Today, I have just pass the ad hoc testing in **Harris's Digital Design Computer Architecture**' s 7.4, and it's the most difficult time to complete the basic version. Pipeline has so many signals to look into which have confused me greatly. Worse still, the mechanism behind is also different from OneCycle and MultiCycle since you have 5 instructions run at the same time and you have to deal with many hazards. Forwarding data branch prediction😳, WTF!!! It's just drive me crazy🤬

2019.5.16

Today, I've finished the Dynamic Prediction using Branch Prediction Buffer(only one prediction bit). And I've passed most Oxer's test code, and branch prediction indeed declines the nop inserted in the Pipeline. And there is a tiny bug in my Branch Prediction Buffer's update, but it won't bring out any problems in the tests I've passed, maybe I'll fix it next Monday.

And to be honest, it's a bit challenging for me since from OneCycle to Pipeline most of what I do is just copying the code from my teacher's PPT and adding some trivial things but this time I only referred to **Computer organization and architecture by Chun feng Yuan** and finish the code all by myself. And I think it's interesting and meaningful to do this, because it surely helps me understand the Pipeline clearlier. And I think I may eat my words that I'll add the FPU since the final is approaching, so it's difficult for me to deal with it.

2019.5.31-6.2

These three days, I'm devoted into constructing the cache and fixing the multidriver problems in BPB and cache, too many bugs😭. And finally, I've fixed these problems, so that my BPB can cooperate with the icache well.

I've implemented 2-way set associative cache, which consists of 4 sets, 2 lines per set and 8 values(4 bytes = 32bits) per line using the LRU strategy to replace the line. To simulate the delay when missing the targeted value, I have designed a FSM with four states : cachefetch, waiting_1, waiting_2, memfetch. When hitting, We just get the value from cache without any delay, while missing, we have to wait for 4 clocks to fetch a block which the targeted value resides from mem and then fetch again from cache. In icache, you'd better just stall Fetch and insert nop to Decode; but in dcache, stall all the pipeline is recommended, otherwise if you just stall stages before Memory(included), and insert nop to Writeback, you may be confronted with problems that before stalling, there are forwarding between Writeback and Execute, but after the stall, the forwarding is not there. It's difficult to keep these related information since forwarding is not sequential logic.

And I've passed Oxer's factorial, but this time I set the CPU to add up from 1 to 33, so it have to involve replacement, which works well.

2019.6.4

Today, I pass the test on the board and add the random data generator **data.py**. A total of **num** random integers module 10 can be generated in generate(num) function. And sum(stride, num) function can sum up the generated integers with different strides. To help you pass the tests suming up the data with different strides. All thr sample test is generated through Oxer's excellent assembler.