Report of Pipelined CPU

Team GJMI: B04202008, B04901036, B04901165

December 11, 2017

1 Members and Team Works

1.1 B04202008

- Instruction Decoding Stage
  - Control Unit
  - Registers
- Hazard detector

1.2 B04901036

- Execution Stage
  - ALU Control Unit
  - ALU
- Forwording Unit

1.3 B04901165

- CPU Top Module:
  - Stage Interface Declaration
  - Stage Connection
- Instruction Fetching Stage
  - Instruction Memory
  - PC and next pc (+flush)
- Memory Stage: Data Memory

2 Implementation

The implementation of the pipelined CPU was divided into two phases. We first developped a single cycle CPU, then modified it to be a pipelined CPU.

In the first phase, 5 modules for IF, ID, EX, MEM, WB stages designed for the required instruction set were declared at the beginning so that each stage could be implemented independently. After the implementation of all the 5 stages, these stages were connected directly without register to form a single cycle CPU and checked by testing all the instructions could be normally executed.

In the second phase, the registers between these 5 pipeline stages were inserted into the CPU as well as the forwarding unit and the hazard detector. The original pipeline stages were also slightly modified to adapt the inserted units. Finally, after checking the forwarding unit and the hazard detector functioned correctly, the pipelined CPU implementation was complete.

3 Modules

We are going to explain the major modules implementation in our pipelined CPU .

3.1 IF Stage

The IF stage fetches the instruction from the instruction memory. It also decided the next PC value by the stall signal from Hazard Detector. If the stall signal is setted, both PC and does not update IF/ID register remains theirs value. Otherwise, the PC value depends on the branch and the jump signals from the Control Unit.

IF Stage receives the branch and the jump signals with those target addresses from the ID stage. If either branch or jump signal is set, the IF stage flushes out the instruction stored in IF/ID-register and updates the PC value to the target address. At last, if there is no any signal were set, the PC value simply pluses by 4.

3.2 ID Stage

Basically, we assign units and connect with wires just exactly the same as the the graph given in *project*1.*pdf* posted by TA.

In detail, there are some exceptions. We combine *Concatination* step to broaden 28 bits to 32 bits for *Jump* instruction into $Jump\_Shift\_Left\_2$ unit. Also, we combine $MUXB$ into *Control* unit, both are just for simplification.

3.3 Hazard Detector

In order to deal with the instruction *lw* while forwarding unit is working, the hazard detector has to detect the case that forwarding cannot work and

must stall. Thus, we input signal RS_addr , RT_addr in ID Stage, RT_addr in EX Stage, and $MemRead$ Signal in EX Stage. if one of RS_addr or RT_addr is RT_addr and $EX\_MemRead = 1$, which means EX Stage is to load word and the destination is one of the source in the ID Stage, then CPU must stall for a cycle.

Also, the stall signal must send to $Control$ unit(we combine the $MUXB$ unit into $Control$ unit for simplification) to send a bubble control signal. In addition, stall signal also need to send to PC unit and IF_Stage unit to avoid updating.

3.4 EX Stage

The critical part of the this stage is the ALU Control Unit, which decodes the Op code sent from Control Unit. If the Op code happens to mean R-type, ALU Control needs additional information from funct to output the corresponding control signal to ALU. Then ALU simply does whatever it is told to do.

Note that several multiplexers are needed. Three of them decides the operands of ALU, with two of them controlled by the Forwarding Unit. The other decides which register to write back to for WB stage.

3.5 Forwarding Unit

If RegWrite signal is set to 1 in MEM or WB stage, and the destination register happens to be one or both of the operands of ALU, we need forwarding to ensure that the correct data is read by ALU. Thus EX stage has additional data lines from MEM and WB stages, while the Forwarding Unit provides the control signal for multiplexers in EX stage to choose from data sent in from ID, MEM or WB stage.

Note that forwarding from MEM stage has higher priority than forwarding from WB stage.

3.6 MEM Stage

Different from the instruction memory, the data memory is a byte addressing memory and can be accessed a byte, a half of a word, or a word. Though only requiring word accessing, all these 3 access modes are implemented in both of memory reading and writing for completeness.

The data memory stores the word in little-endian format. So for writing a word, the less significant byte is stored in the cell on the input address and the following 3 bytes are stored in the following cells. Then, for reading a word, the cells from the input address to the next 3 address are read.

The mechanism for byte and half a word accessing are similar to word accessing. The only 2 different are, as data in the CPU are transferred by 4-byte bus, the unused bytes are blocked in writing and are extended by zero in reading.

Since the memory can be accessed in 3 different mode or be rest, it requires 2 bits to dicide what to do for reading and another 2 bits for writing. The 2-bits signals are:

- 00: No access.
- 01: Byte access.
- 10: Half a word access.
- 11: Word access.

4 Problems and Solutions

4.1 Branch instuction in the single cycle phase

Problems: At the first implementation phase, the implementation of single cycle CPU, the branch instuction could not be executed. It made the vvp simulation program freezed and not reponsed until we forced the simulation program terminate.

Solutions: We inserted the IF/ID register then the simulation could terminate. And we found the IF stage output wrong PC value to ID stage when the branch signal was set. After we removing this bug, the branch instuction functioned normally.