Electrical Engineering
& Computer Science
University of Missouri

ECE 4270/7270: Computer Organization, Fall 2018
LAB 4: Pipelined MIPS Simulator
Supporting Data Hazard Detection and Data Forwarding

Scope

In this lab assignment, you will extend the pipelined MIPS simulator that you have developed in Lab 3. So far, for the pipelined simulator that you developed, we assumed there would be no data hazard, so there was no mechanism to handle possible data hazards. Now, we want the simulator to handle data hazards. In the lectures, we discussed two ways to handle data hazards: i) by introducing pipeline stalls; ii) by data forwarding.

In this lab, you need to extend the pipelined MIPS simulator that you developed in Lab 3, such that it should detect data hazards and handle them accordingly. You need to implement both pipeline stalls and data forwarding.

1. Data Hazard Detection and Introducing Pipeline Stalls:

Notice that the pipeline registers keep track of source and destination registers (i.e., registers to be read/written). So, to detect a data hazard, we need to compare the source and destination registers of instructions that we consider for data dependence. We know the source and destination registers of an instruction in ID stage, so we have to test the following conditions:

If you recall from our discussion in class, we had conditions to detect data hazards:

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and
(EX/MEM.RegisterRd = ID/EX.RegisterRs))

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and
(EX/MEM.RegisterRd = ID/EX.RegisterRt))

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and
(MEM/WB.RegisterRd = ID/EX.RegisterRs))

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and
(MEM/WB.RegisterRd = ID/EX.RegisterRt))

where RegisterRd, RegisterRs and RegisterRt represent the name of destination and source registers, respectively.

As you may recall, we can detect a data hazard for an instruction that is in ID stage, if there is an

instruction in EX or MEM stage where the destination register is one of the source registers of the instruction in ID stage.

If the instruction in the ID stage is stalled, then the instruction in the IF stage must also be stalled; otherwise, we lose the fetched instruction. Preventing these two instructions from making progress through the pipeline is accomplished simply by preventing the PC and the IF/ID pipeline register from changing. Provided these registers are preserved, the instruction in the IF stage will continue to be read using the same PC, and the registers in the ID stage will continue to be read using the same instruction fields in the IF/ID pipeline register.

On the other hand, we can insert pipeline bubbles (i.e., stalls) by setting all control signals to 0 in ID/EX pipeline register. During the consecutive cycles, these signals will be forwarded to EX, MEM, and WB stages through pipeline registers and will create a *nop* instruction (basically there will be no operation performed, but only control signals will be forwarded).

Once the producer (i.e., source) instruction finishes WB stage, the stalled instructions (i.e., the ones in ID and IF stages) can resume and make progress through the pipeline. Notice that the instruction in ID stage needs to read the source register whose value has just updated by the former instruction in WB stage. The control signals should be repopulated accordingly and should be forwarded to the next pipeline stage. The instruction in IF stage, on the other hand, should move into the ID stage, and the PC should be updated accordingly, such that we can fetch new instruction in the next cycle.

In this section of the lab, you are asked to implement data hazard detection and introducing pipeline stalls to avoid these hazards.

2. Data Forwarding to Reduce the Number of Pipeline Stalls

So far, we have not considered any forwarding paths in the pipeline. Thus, the data hazards would not be resolved until the producer (i.e., source) instruction reaches WB stage. However, as we have discussed in the classroom, the value that causes the dependence may become available earlier in the pipeline (either at the end of EX stage, or at the end of MEM stage), so the value can be forwarded to the EX stage for an instruction that is waiting for that particular value. Forwarding a value before the WB stage effectively reduces the stalls needed to avoid data hazards.

In this section of the lab, you are asked to implement data forwarding on top data hazard detection. Remember that we had the following forwarding conditions:

i) To forward from EX stage:

```
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and
                                                (EX/MEM.RegisterRd = ID/EX.RegisterRs)) {
        ForwardA = 10
}

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and
                                                (EX/MEM.RegisterRd = ID/EX.RegisterRt)){
        ForwardB = 10
}
```

ii) To forward from MEM stage:

```
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not
                                        (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)  and
                                        (EX/MEM.RegisterRd = ID/EX.RegisterRs)) and
                                        (MEM/WB.RegisterRd = ID/EX.RegisterRs)){
        ForwardA = 01
}

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not
                                        (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and
                                        (EX/MEM.RegisterRd = ID/EX.RegisterRt)) and
                                        (MEM/WB.RegisterRd = ID/EX.RegisterRt)){
        ForwardB = 01
}
```

Notice that, above forwarding conditions already takes care of double data hazards.

2. Details

You can exclude branch and jump instructions from the scope of this lab assignment.

Since you will implement two versions of the pipeline, namely:
i) Pipeline that detects data hazards, but cannot forward data
ii)) Pipeline that detects data hazards and can forward data

you need to define a global variable called ENABLE_FORWARDING and it can be set to 0 or 1 through a simulator command called "forwarding". You can add a case statement in the handle_command() function, similar to the following:

```
case 'f':
        if (scanf("%d", &ENABLE_FORWARDING) != 1) {
                                        break;
        }
        ENABLE_FORWARDING == 0 ? printf("Forwarding OFF\n") : printf("Forwarding ON\n");
        break;
```

Above case statement would allow you to set/reset ENABLE_FORWARDING variable that indicates whether you need to run the simulator with data forwarding capability or not. Default value of ENABLE_FORWARDING should be 0 (i.e., FALSE). In this case, simulator should simulate a pipeline that detects data hazards and introduce stalls to avoid them, but should not perform any data forwarding. On the other hand, when ENABLE_FORWARDING is set to 1 (i.e., TRUE) through running "forwarding 1" command in the simulator, simulator should simulate a pipeline that detects data hazards and uses data forwarding to avoid the data hazards.

ENABLE_FORWARDING = FALSE; //default value, no data forwarding
ENABLE_FORWARDING = TRUE; // will be set to TRUE when "forwarding 1" command is used, data forwarding should be in place when data hazard is detected.

You need to add the control signals required and maintain them through the pipeline stages via pipeline registers.

You are given a test input file in Canvas that contains a set of instructions. You should use it to test your implementation.

Grading Rubric

Code: Pipelined MIPS simulator that handles data hazards (75)
Report: 25 points

Pipelined MIPS simulator that handles data hazards (75 points):

In order to get a full credit for the code, your pipelined MIPS simulator that handles data hazards should work correctly (i.e., it should introduce pipeline stalls to avoid data hazards if there are data dependencies, and forwarding should work correctly when it is enabled).

Lab report (25 points):
Your report should give details about the work distribution within the group (who did what), milestones in your work and your implementation decisions (why did you choose the way you did it, and/or how did you do that). Explain any difficulty you observed, any optimizations you have made.

Submission

You should submit the lab report along with the code you developed (provide makefile, as well). Please, generate a pdf file of your report and name it lab4_report_groupX.pdf, then place it into the folder called lab4_groupX (where X is your group number). The folder lab4_groupX should also contain the src/ folder that contains the simulator code (i.e., mu-mips.h, mu-mips.c and Makefile). Then, please compress the lab4_groupX folder as lab4_groupX.tar.gz and submit it through the Canvas.

Due Date

Your lab is due on:
10/30 for Section A (before the lab)
11/01 for Section B (before the lab)