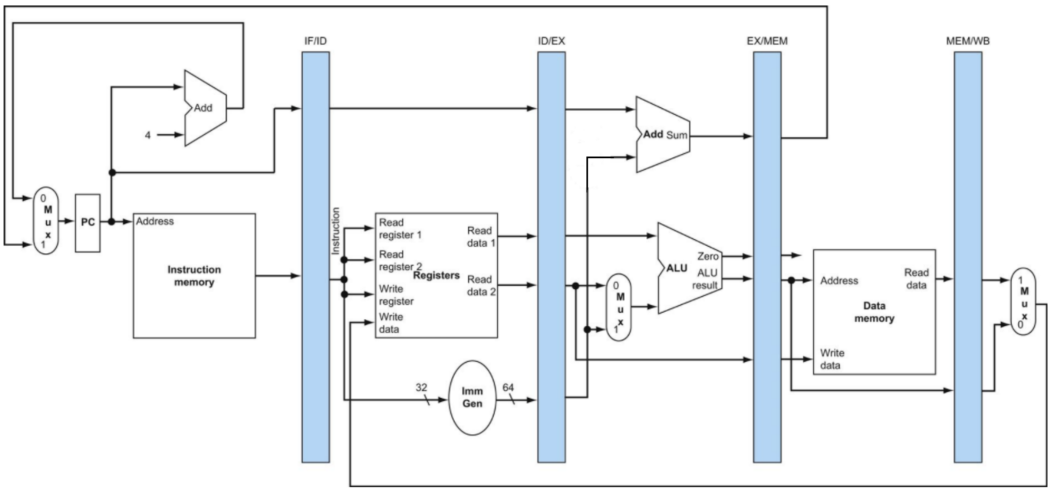
Organization of Digital Computers Lab
EECS 112L
Lab 3: Pipelining

University of California, Irvine

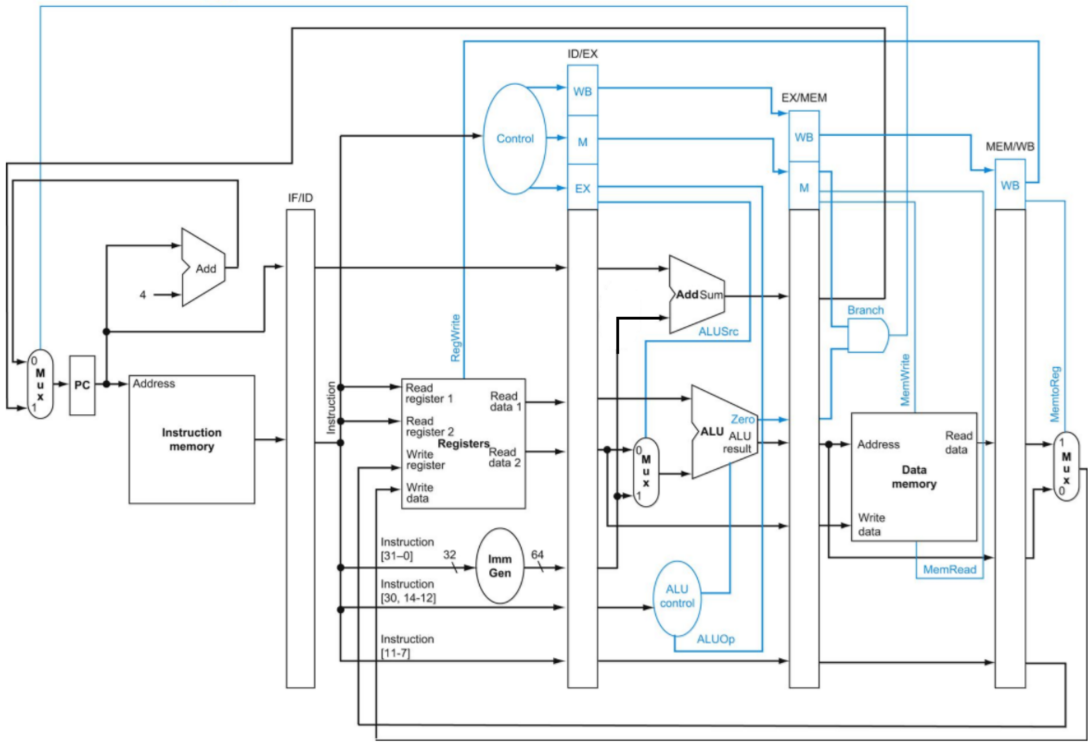
Due date: May 19th, 2019

1 Single Cycle versus Pipeline

In a single-cycle design, every instruction takes exactly one cycle. In other words, the single-cycle design must allow the slowest instruction. While in pipelining, the datapath is broken into separate stages independant from each other, named IF (Instruction Fetch) , ID (Instruction Decode), EXE (Execution), MEM (Memory) and WB (Write Back) in RISC-V. All the pipeline stages take a single clock cycle. As a result, every instruction in RISC-V datapath takes exactly five cycles to be executed. According to the following figure, pipeline registers are located between every two stage. All the figures in this document extracted from [1].

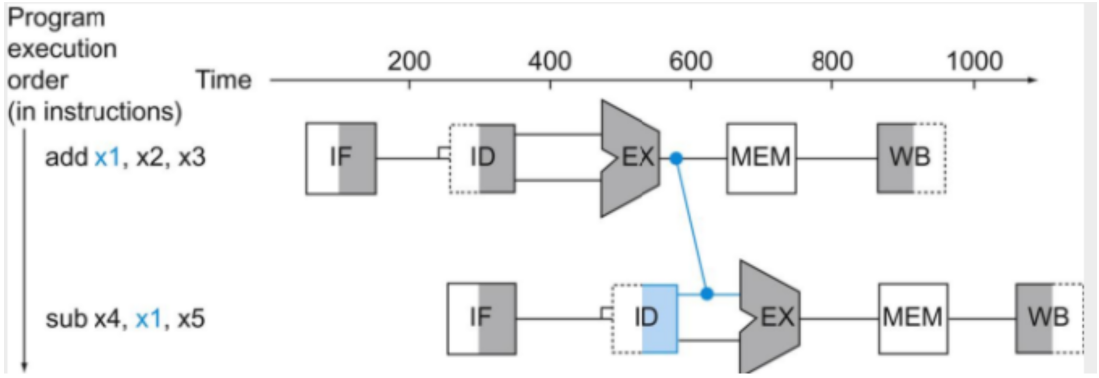


We can pipeline the tasks as long as we have separate resources for each stage. Pipelining improves performance by increasing the instruction throughput. The idea behind pipelining is to keep all the stages busy at all the times. As an example, when an instruction is using the ALU, the register file and the instruction memory are used by the other instructions. Every instruction seems to have its own datapath. All instructions advance during each cycle from one pipeline register to the next. In this case, all the required information of an instruction such as control signals and registers need to be stored in pipeline registers. Therefore, the instruction is able to restore all the required information for the next pipeline stage. According to the following figure, all the datapath information as long as control outputs generated by Controller are stored in pipeline registers for every instruction.

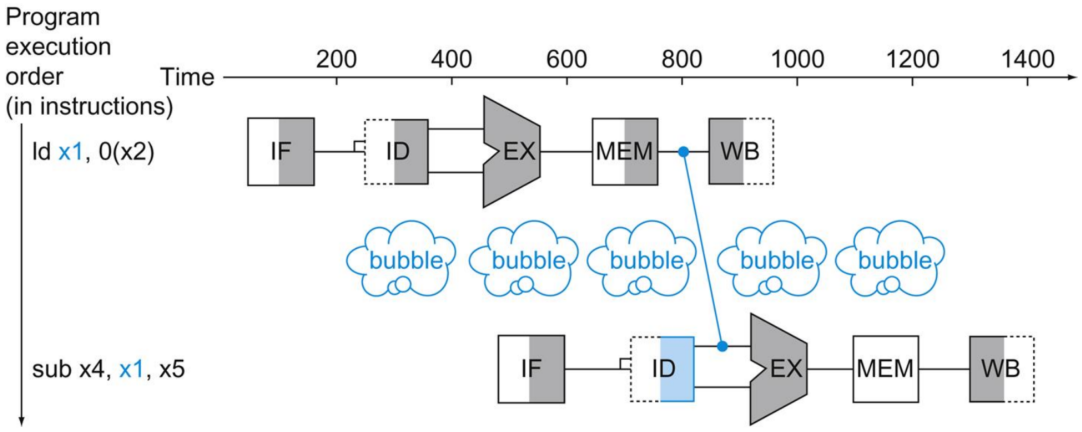


2 Data Hazards

Data hazard occurs when a planned instruction cannot execute in the proper clock cycle because data that are needed to execute the instruction are not yet available. In other words, pipeline should be stalled because one instruction waits for another to complete. Data hazards occurs because of dependence of one instruction to an earlier one that is still in the pipeline. As an example, for the following two instructions, sub instruction needs to decode the new value of X1 updated by the add instruction. The add instruction doesn't write back until the fifth stage meaning that we need to stall the pipeline for three cycles because the value of X1 is read from register file in ID stage.
add X1, X2, X3
sub X4, X1, X5
Forwarding simply means that to retrieve the missing data from internal buffers rather than waiting for it to arrive from programmer-visible registers or memory. With forwarding, the new value of X1 is ready to be decoded by the sub instruction right after it passes the EXE stage of the add instruction.

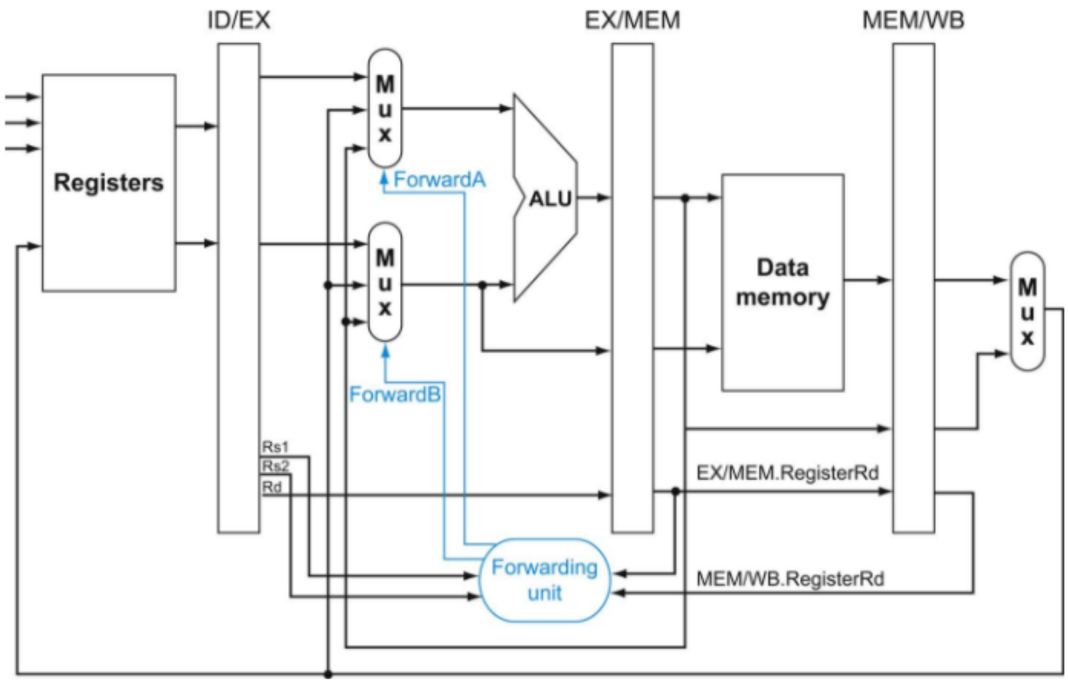


RISC-V supports two types of forwarding. First, forwarding from an EXE stage of an earlier instruction to the EXE stage of the following instruction. Second, forwarding from a MEM stage of instruction entering the pipeline at time n to the EXE stage of another instruction entering pipeline at time n + 2. The second scenario is used for an R-format instruction following a load instruction. In this case, a bubble is required since the updated data is available at the end of the MEM stage.



3 Forwarding Unit

To implement forwarding, we need to a) detect the data dependencies by adding the forwarding unit, and b) add the required paths to enable forwarding. The conditions leading to the detection of EXE and MEM hazardz are presented below.



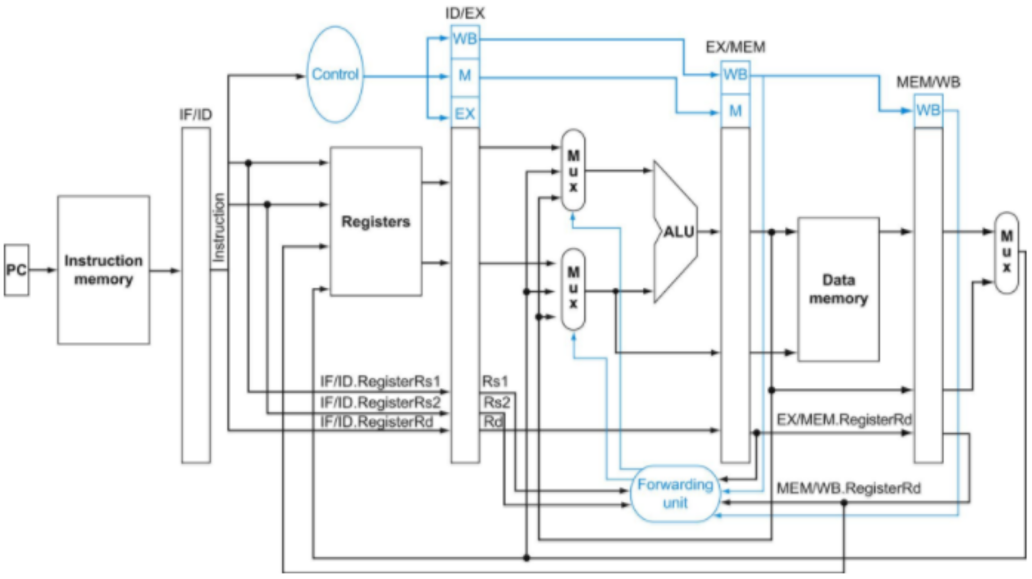
3.1 EX Hazard

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 10
if  (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10
```

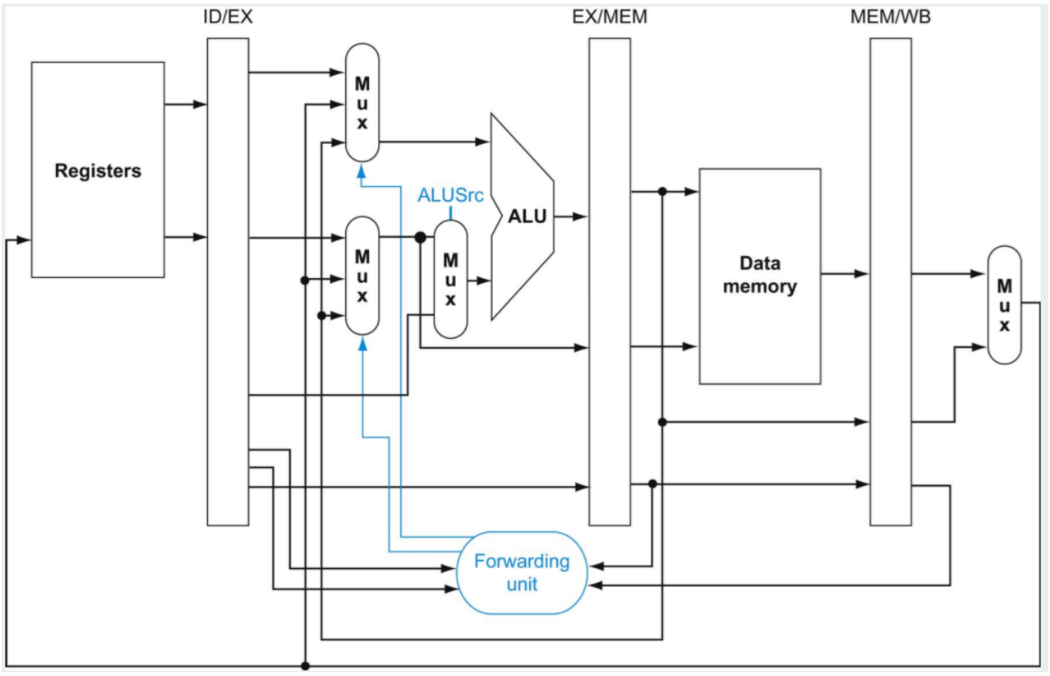
3.2 MEM Hazard

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01
```

To add the required paths, multiplexers located in the input of ALU are then extended, resulting in the following datapath.



In addition to the previous multiplexers, forwarding multiplexers are added in the ALU inputs, similar to the multiplexer responsible for choosing between Reg2 or Immediate value, according to the following figure.



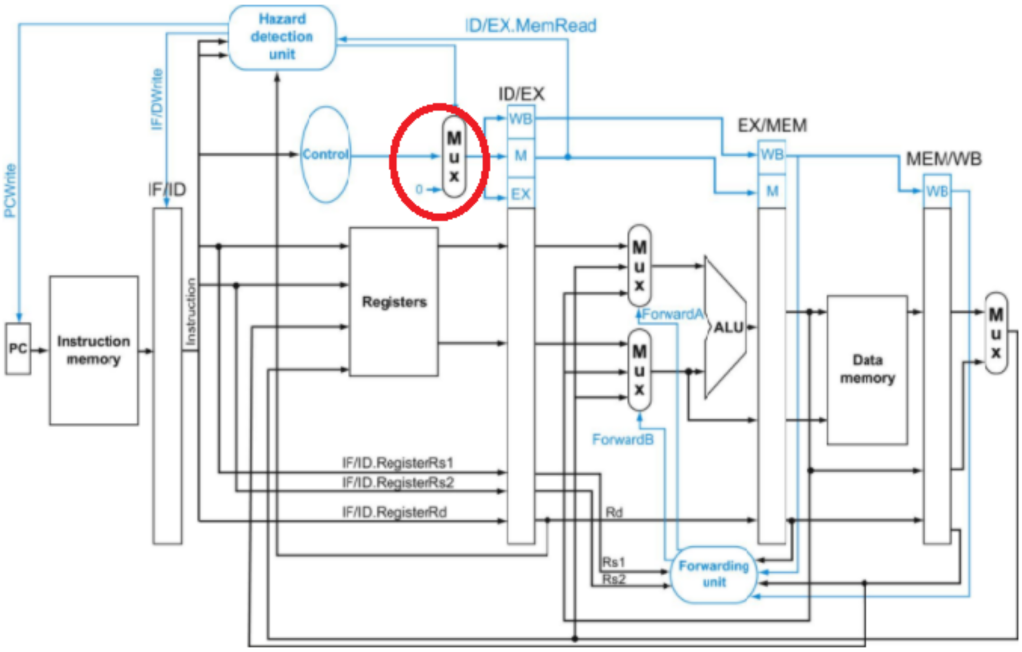
4 Hazard Detection Unit

We start the discussion with data hazard, which occurs when the source register of an instruction in an early stage of the pipeline is the same as the destination register of another instruction at a later stage of the pipeline. As an example, consider the scenario when the destination register of a Load instruction is the same as one of the source registers of an ADD instruction in the ID stage (Based on the forwarding unit conditions explained before, this data data dependency can not be resolved by the forwarding unit.):
load X1, X2(100)
add X2, X1, X3
sub X4, X5, X6
The successful execution of this sequence requires the ADD instruction to be immobilized in the pipeline for one cycle, i.e. stalling, while the Load instruction moves forward one stage. Specifically, suppose that at cycle n, the load instruction is in EXE stage, the add instruction is in its ID stage, and the sub instruction is in IF stage. Hazard detection unit detects the data hazard on X1 register according to the following conditions.

```
if (ID/EX.MemRead and
  ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
   (ID/EX.RegisterRd = IF/ID.RegisterRs2)))
   stall the pipeline
```

Because of data dependency, the add instruction can not be executed. In other words, when the haz-

ard is detected in cycle n, the hazard detection unit activates the IF/IDWrite and the PCWrite signals, which results in the values on IF/ID pipeline register and the PC register not being updated with the next clock edge. Also, the new multiplexer chooses the value of all zeroes for the control signals instead of the values generated by the control unit. As a result, in cycle n + 1, the add instruction is still in its ID stage, and a neutral replica of the add instruction, i.e. a NOP, occupies the EXE stage.



5 Material to be submitted

- Your pipeline datapath needs to support forwarding and hazard detection unit.

- Simulation [60% of your total grade]

  – We are going to test your code. Each test has some points based on the total number of tests.

  – No points if your code does not work or if there is some error during simulation.

  – The initial delay before the rst output is not important. We are going to test your pipeline implementation, forwarding and hazard units. Implementing the control hazard has bonus points.

- Synthesis [20% of your total grade]

  – Synthesize your design and compare the clock period, and area reports with the single cycle processor.

- Report [20% of your total grade]

  - Put the Latex or pdf version of your report in the doc folder.

  - Block diagram of the design with the detailed explanation of how your processor works. Your block diagram should contain all your design modules. This is your block diagram. Do not copy it from the lab manual.

  - After synthesis, put the riscv.elab.rpt, and riscv.qor.rpt in the doc folder of your project.

- Important notes

  - Submit the project in zip format.

  - The name of the uploaded zip file should be your group name if you are working in a group.

  - The report should contain name and student ID (on the cover page).

  - No points if you submit a wrong file by mistake, we do not accept any changes in code after the deadline.

  - NO late submission is permitted.

6 References

[1] Waterman, Andrew, et al., The RISC-V Instruction Set Manual, volume I: User-level ISA, version 2.0, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54 (2014).