**RISC-V Class Project Phase 6 – Forwarding Schematic + CA**

Phase 6 of the Class Project adds the forwarding logic to handle data hazards to the Schematic created in Phase 4 and the Codasip implementation created in Phase 5. The schematic update will be described first, followed by the Codasip code update.

There are two types of forwarding which need to be handled. The first is referred to as Data Hazard Forwarding (DHF) and handles the RISC-V Data Hazards described in section 4.7 of the textbook. The second is referred to as Register File Forwarding (RFF), and is not part of the actual RISC-V design but is required due to an issue with the Codasip Register File implementation.

1. **Data Hazard Forwarding Schematic**

Copy the schematic standardname4.xml to standardname6.xml. Alternatively named schematics will not be accepted. Open the schematic in app.diagram.net. The changes required for DHF will all be implemented in the EX Stage.

Figure 1, which is Figure 4.52 from the textbook, shows the components necessary to implement DHF. Add two 3-1 Multiplexor components for the Mux elements in each of the ALU source inputs. Assign appropriately named signals for the multiplexor select signals “ForwardA” and “ForwardB”, such as s\_ex\_fwdA. Note that several of the input signals to these multiplexors come from later stages (ME and WB). Note also that on the src2 side of the ALU, forwarding replaces the register file inputs so that the forwarding multiplexor must be before the immediate multiplexor from Phase 4, which is not shown in Figure 1.

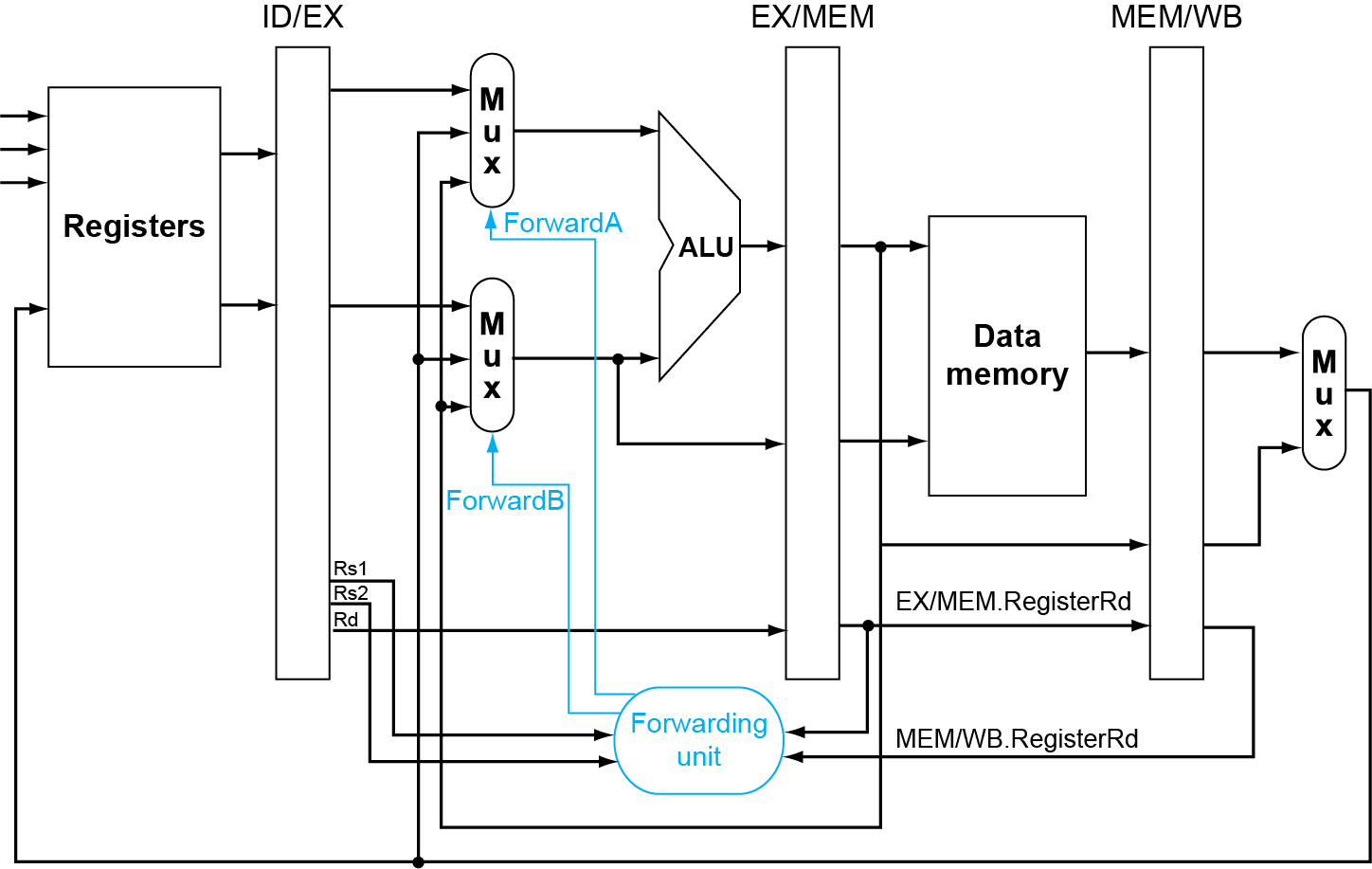


Figure 1

Add a control element named FWDCTL with the desired inputs and outputs to create the Forwarding Unit and name it appropriately. This control block will implement the functions in section 4.7 of the textbook. The multiplexor selects are described in Figure 4.53 in the textbook, replicated in Figure 2 below. One set of equations in Figure 3 below shows the required input signals. The source select signals in the EX stage must be pipelined from the ID stage, so these pipeline registers must be added in the ID stage.

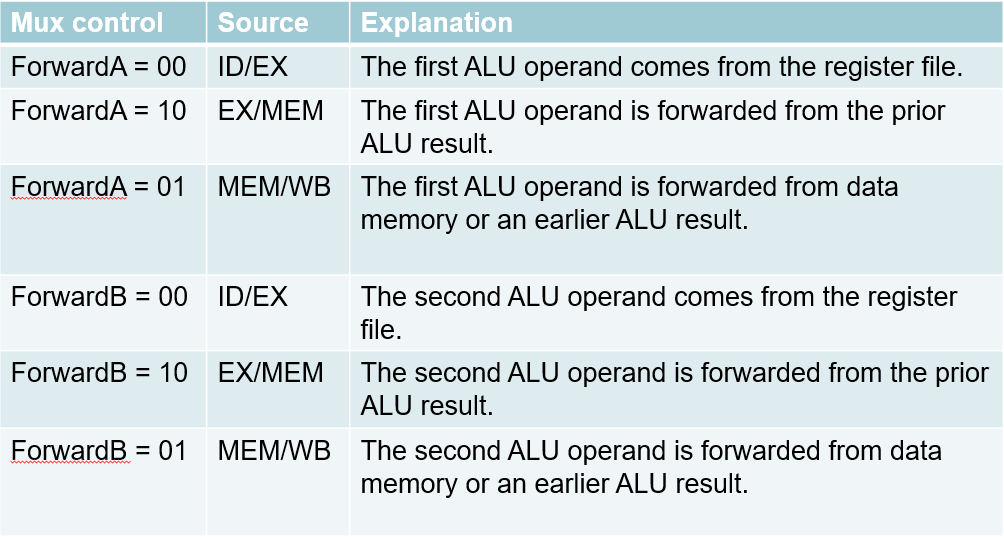


Figure 2

**MEM hazard**

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.RegisterRs1))

and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) 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.RegisterRs2))

and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01

Figure 3

1. **Register File Forwarding Schematic**

RFF is not part of a normal RISC-V implementation, but is necessary in order to overcome a functional issue with the Codasip Register File module. In the textbook section 4.6, the Register File is assumed to write the result data into the register selected by the rd field in the middle of the clock cycle in the WB stage. This means that if a register written in the WB stage of an instruction is read in the ID stage of the instruction which is three cycles later (i.e. read and write occur in the same clock cycle) the read data will be the data written in that cycle.

In the Codasip Register File model, however, the write data is written to the register selected by rd at the end of the clock cycle in the WB stage. This means that if that register is read in the ID stage of the instruction which is three cycles later, the read data will be the old data in the register (before the write) and the behavior will not match the RISC-V architectural assumptions.

From a software standpoint, the code sequence is, for example:

1. add xA, xB, xC
2. instruction not writing or reading xA
3. instruction not writing or reading xA
4. add xD, xA, xA

In this case the fourth instruction should use the value of xA written in the first instruction. However, with the Codasip Register File, the fourth instruction will use the value in xA PRIOR to the execution of the first instruction.

RFF is necessary to allow the Codasip implementation to match RISC-V, and will be implemented in the ID stage. This will be very similar to DHF – each Register File output will have a multiplexor selecting either the Register File output or the data to be written, and there will be a control block (RFFWD) which receives a set of inputs and produces the control signals for each of the muxes. Be sure to name any new signals which are created. Also be sure to maintain our convention that the input signal to every pipeline register has the same name part as the register output.

1. **Examples**

Figure 4 shows rough examples of what the ID Stage and EX Stage schematic pages should look like.

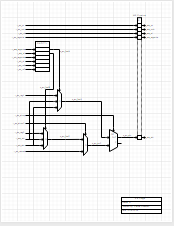
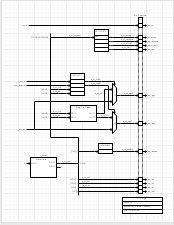


Figure 4

1. **Submit the Updated Schematic**

Although not required, it is recommended that you submit the updated schematic as soon as it is complete, as standardname6.xml via a Slack DM to Steve Sheafor. This will allow for feedback prior to creating the Codasip code. Schematic scoring is described in Section 8.

1. **Update the Codasip Files**

Copy the standardname5 project to standardname6. For Phase 6, as can be seen from the schematics, only the ID and EX stages need to be modified. It is recommended to print these two schematic pages and mark changes as they are made to the Codasip files.

* 1. **Data Hazard Forwarding**

Data Hazard forwarding is implemented in the EX stage, so update the following files.

* + 1. **ca\_defines.codal**

This phase adds the two 3-1 multiplexors on the inputs to the ALU, so add the defines for the multiplexor select signals and create their width definitions. These are control signals (not data path signals) although they don’t come from the Decoder, so they must be assigned enums.

* + 1. **ca\_resources.codal**

Add the definitions for all of the new signals which are required, which will include both the multiplexor control signals and the new data path signals. Add the new register definitions for the pipelined source selects.

* + 1. **ca\_pipe\_stage3\_ex.codal**

Add the two multiplexors as described in Phase 5, using the definitions created in section 5.1.1 above. Note that these muxes must be above the other logic in the EX stage because they create signals assigned in that logic. Note that some signals come from other pipeline stages (ME and WB) which is acceptable because these pipeline stages are to the right of the EX stage.

Add the logic for the FWDCTL block. This is simply logic which creates the multiplexor select signals. One way to implement this is:

if (Conditions enabling EXMEM forwarding) s\_ex\_fwdA = (EXMEM Constant from defines)

else if (Conditions enabling MEMWB forwarding) s\_ex\_fwdA = (MEMWB Constant from defines)

else s\_ex\_fwdA = (NOFWD Constant from defines)

The forwarding block must be above the multiplexors to avoid forward references.

* 1. **Register File Forwarding**

Register File forwarding is implemented in the ID stage, so update the following files in a similar way.

* + 1. **ca\_defines.codal**

This phase adds the two 2-1 multiplexors on the outputs of the Register File, so add the defines for the multiplexor select signals and create their width definitions.

* + 1. **ca\_resources.codal**

Add the definitions for all of the new signals which are required, which will include both the multiplexor control signals and the new data path signals.

* + 1. **ca\_pipe\_stage2\_id.codal**

Add the two multiplexors as described in Phase 5, using the definitions created in section 5.2.1 above. Note that these muxes must be below the Register File read logic in the ID stage because they use signals assigned by that logic. Note that some signals come from another pipeline stage (WB) which is acceptable because that pipeline stage is to the right of the ID stage.

Add the logic for the RFFWD block. This is simply logic which creates the multiplexor select signals. Make sure this logic doesn’t create any forward references.

The forwarding block must be above the multiplexors to avoid forward references.

In the id\_output event, add any new pipeline register assignments.

* 1. **Register File Issue**

The RISC-V Register File definition includes one special function which is not handled correctly by the Codasip Register File component. Identify and repair this issue, which will cause the test to fail if it is not fixed. Do NOT ask questions about this in any general Slack channel.

* 1. **Update for - -info 7**

If the signal inputs to the ALU block have changed, the - -info 7 print function must be updated.

* + 1. **other/ca\_utils.codal**

The ca\_utils.codal file is shown in Figure 5. In line 35, the first two signals must be the source 1 and source 2 signal inputs to the ALU. At least one of these signals will have changed from Phase 5, so update this file with the correct signals. This will allow the - -info 7 debug print function to display the correct signals.

A screenshot of a computer

Description automatically generated

Figure 5

* 1. **Unacceptable Modifications**

It may be necessary to change files other than those described above (e.g. in a stage other than ID and EX). If that is done, the resulting Codasip code must continue to match the schematics. This means that no new control blocks may be added, and no inputs or outputs may be added to any existing control blocks.

1. **Build the Project**

Once all of the CA files are updated, build the project by first double clicking the button to the left of Model Compilation (ca) (NOT Model Compilation (ia)) in the Task window of the Codasip Perspective. Any build errors will appear in the Console window, so correct any missing assignments, syntax errors, etc. Continue until the Model Compilation builds correctly.

Build Simulator (ca) in the Task window, and again correct any syntax errors.

1. **Run the Test Program**

Once both the Model Compilation (ca) and Simulator (ca) tasks finish successfully, the next step is to run the test program phase6\_test. Import this project, which should include the object (\*.xexe) files.

Execute Run -> Debug Configurations as in Phases 1/2/3. The C/C++ Project should be phase6\_test, and the Application should be Debug/phase6\_test.xexe. For the debugger, select standardname6.ca.standardname6-ca-simulator. If this is not a choice, the project build process has not completed successfully. Make sure that you are not using the IA Simulator.

The running and stepping controls are the same as described in Phase 2. The debugging functions described there will be useful. Setting info to 3 (INFO\_RF\_WRITE) can be particularly valuable for this test. The test should terminate in the 3rd nop after the halt (line 103), and all registers must match the values shown in the comments at the top of the test program. After the test completes, open the phase6\_test Console in the Console window, and verify that there are no warnings there.

1. **Scoring the Project**

The project should be submitted when the test program phase6\_test passes when using the CA Model (be sure to select this in Debug Configurations and not the IA model which was used in Phases 2 and 3). Unsuccessful submissions will be rejected. The hardware project counts for 80% of the total Phase score.

The score for a successful Project submission will be determined by the time of submission relative to the Target Date. A bonus of 1% of the CA Project score (i.e. 0.8% of the Phase 6 score) will be added for each day earlier than that the successful project is submitted. A penalty of 4% of the CA Project score (i.e. 3.2% of the Phase 6 score) per day will be imposed after the Target Date.

The score for the schematic will be determined by the time of submission with deductions for any errors. Error deductions will be similar to those for Phase 4 (1 point for a bad signal name, 2 points for a bad connection, etc.) but the deduction will be from the entire Phase 6 score, not just the 20% which is for the schematic. A bonus of 1% of the schematic score (i.e. 0.2% of the Phase 6 score) will be added for each day earlier than the Target Date a fully correct schematic is received, up to a maximum of 7%. A penalty of 4% of the Schematic score per day will be imposed after the Target Date. Submitting the schematic early not only gains schematic bonus points but produces a correct schematic which aids in creating the actual hardware project.

1. **Exporting the Project**

Once the test program is running, the project should be Exported as standardname6.zip to the G:/Submissions folder.