LSU EE 4720 Homework 3 Solution Due: 28 March 2024

Collaboration Rules

Each student is expected to complete his or her own assignment. It is okay to work with other students and to ask questions in order to get ideas on how to solve the problems or how to overcome some obstacle (be it a question of MIPS or assembler syntax, interpreting error messages, how a part of the problem might be solved, etc.) It is also acceptable to seek out assembly language resources for help on MIPS, etc. It is okay to make use of AI LLM tools such as ChatGPT and Copilot to generate sample code. (Do not assume LLM output is correct. Treat LLM output the same way one might treat legal advice given by a lawyer character in a movie: it may sound impressive, but it can range from sage advice to utter nonsense.)

After availing oneself to these resources **each student is expected to be able to complete the assignment alone**. Test questions will be based on homework questions and **the assumed time needed to complete the question will be for a student who had solved the homework assignment on which it was based**.

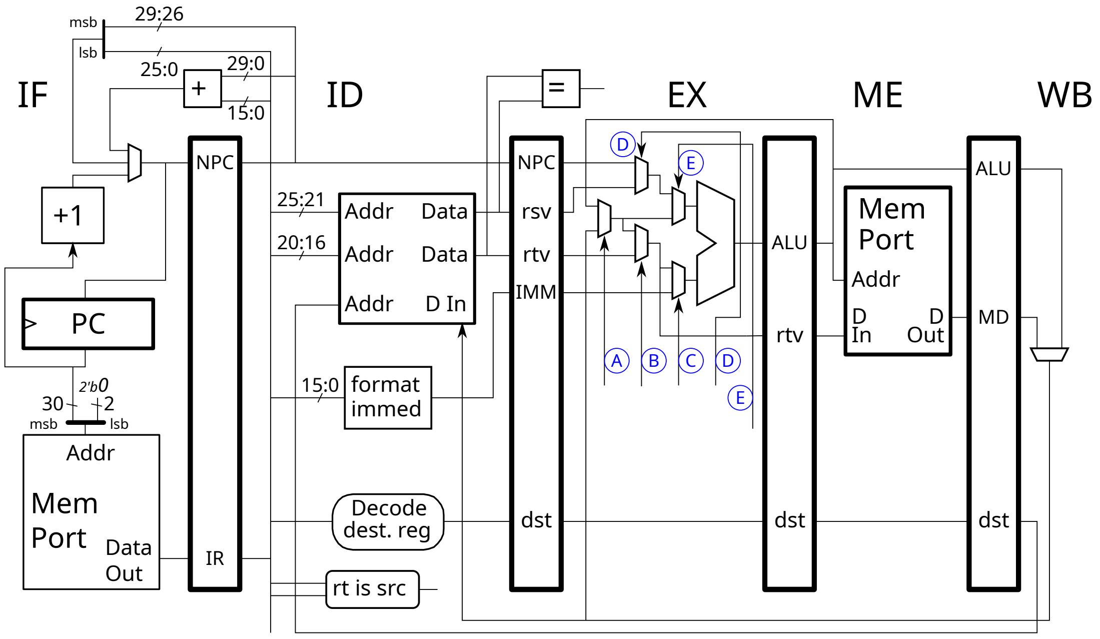
Student Expectations

Some of the problems require thought, and students are expected to persevere until they find a solution. It is the students' responsibility to resolve frustrations and roadblocks quickly, and hopefully with the satisfaction of making progress. There are plenty of old problems and solutions to look at. One way to resolve issues is to ask Dr. Koppelman or others for help.

For the 2020 Final Exam, and other exams and solutions visit `https://www.ece.lsu.edu/ee4720/prev.html`*.*

Problem 1 on the next page.

Problem 1: Appearing below is the slightly lower cost MIPS implementation from the 2020 midterm exam. In the 2020 exam three EX-stage select signals were labeled, (A-C), here all five are, (A-E). Below that is an incomplete pipeline execution diagram (it lacks a code fragment) and a timing diagram showing values on the labeled select signals over time. In 2020 midterm exam Problem 1(a) these signal values had to be found given a code fragment. For this problem, the signal values are given. Write a code fragment that could have produced these signals. Feel free to look at the solution to 2020 Problem 1(a) for help and practice.



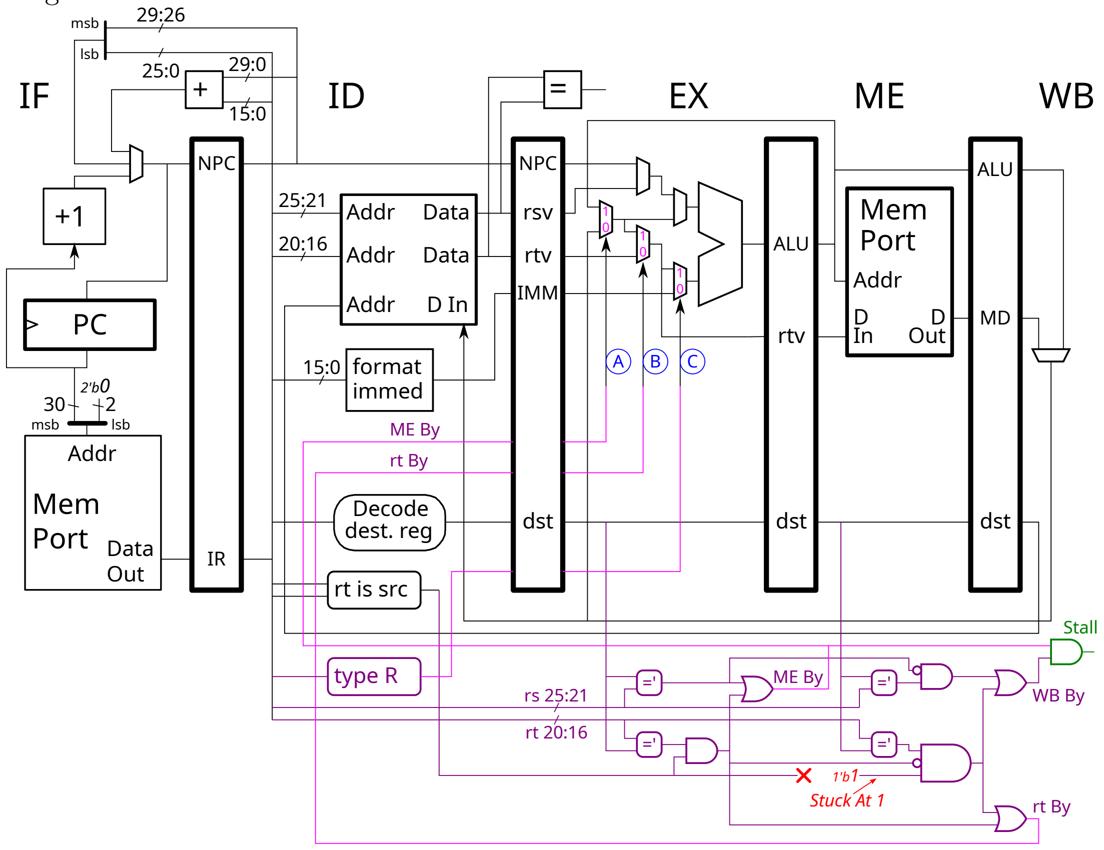
☑ Write a program that could have resulted in these select signal values.

The solution appears below. Registers carrying dependencies are shown in upper case. The last instruction had to be some kind of a store because the A and B signals, in cycle 5, indicated that a value bypassed from ME was needed, but because C=1, that value could only be needed for the EX/ME.rtv latch, which is used for the store value.

```
# SOLUTION
# Cycle                    0     1     2     3     4     5     6     7
addi R1, r2, 3             IF    ID    EX    ME    WB
sub r4, R1, r3                   IF    ID    EX    ME    WB
add R5, R1, R1                         IF    ID    EX    ME    WB
sw R5, 2(r3)                                 IF    ID    EX    ME    WB
# Cycle                    0     1     2     3     4     5     6     7

A                                      X     0     1     0
B                                      X     1     0     0
C                                      1     0     0     1
# Cycle                    0     1     2     3     4     5     6     7
D                                      1     X     X     1
E                                      0     1     1     0
# Cycle                    0     1     2     3     4     5     6     7
```

Problem 2: Appearing below is the solution to 2020 Midterm Exam Problem 2, showing control logic for those slightly lower cost bypass paths, with one unfortunate change. The bottom input to the 3-input AND gate is supposed to connect to the $\boxed{\text{rt is src}}$ logic block. Due to some defect that input is stuck at 1. (This is known as a *stuck-at* fault.) This stuck-at fault is shown on the diagram.



Continued on Next Page

☑ Write a code fragment that will not execute as intended on this hardware due to the stuck-at fault. *Note: In the original assignment the phrase "execute correctly" was used instead of "execute as intended".*

Solution appears below. In the code fragment registers written with upper-case letters were specially chosen to expose the flaw (by setting up certain dependencies). To understand the solution work out the select signal values (those labeled A-E) in cycle 3. They should show that `Stall` is 1 when it should be 0, the problem that the code fragment is exposing. The `ori` instruction is the victim in this code fragment, in that its control signals are wrong, including the erroneous stall.

To expose the flaw three conditions have to be satisfied. These are explained briefly here, and in detail below. Condition (1): The last instruction had to be a type I instruction that writes a register, `ori` is chosen here. Condition (2): The same register number must be used `rt` register of the last instruction and the destination of the first (of three) instruction. The first instruction is `add` and the matching register is `r1`. When the first two conditions are satisfied the output of the 3-input AND gate and the `rt By` signal will be 1 when they should have been 0. Condition (3): There must be a dependency between the second and third instructions. In the solution that dependence is carried by `r4`. The second instruction is `sub` but any arithmetic or logical instruction will do. As a result of the third condition (combined with the first two) there will be a stall that would not occur without the stuck-at fault.

Here is a more detailed explanation. The bottom input to the 3-input AND gate is stuck at 1. For something to go wrong we need a situation in which the bottom input should have been 0. The bottom 3-input AND gate input connects to $\boxed{\text{rt is src}}$, which is 0 when there is a type-I arithmetic or logical instruction in ID. In the solution below an `ori r1, r4, 7` is chosen, notice that it is in ID in cycle 3. The output of the 3-input AND gate is used to compute the `rt By` signal (for the B mux) and to compute the stall signal. To cause execution to differ a code fragment must be found which will result in `Stall` being 1 when it should have been 0. For that we need the output of the 3-input AND gate to be 1 (when it should have been 0) and for the upper input to the Stall (green) AND gate to be 1.

To get the output of the 3-input AND gate to be 1, we need a match between the `rt` register of the `ori` instruction (`r1`) with the destination of the instruction in ME in cycle 3. To get that match an `add r1, r2, r3` instruction is chosen. The `ori` we have chosen will result in the middle input to the 3-input AND being 0, and so with the `ori` and `add` instructions the 3-input AND gate output will be 1 when it should have been 0.

To get the `Stall` signal to be 1 when it should have been zero we need to set up conditions for an ME bypass (legitimately, not due to the fault). As a result of the true ME bypass and the flaw-induced WB By there will be a stall that would not occur otherwise. To get ME By to be 1 in cycle 3, the destination of the second instruction is chosen to match the `rs` source of the `ori`. A `sub r4, r5, r6` achieves this.

```
# SOLUTION
# Cycle         0  1  2  3  4  5  6  7
add R1, r2, r3  IF ID EX ME WB
sub R4, r5, r6     IF ID EX ME WB
ori R1, R4, 7         IF ID -> EX ME WB  # Unnecessary stall in cycle 3.
```

As luck would have it this defect has occurred in a computer that's on Mars. The computer can't be fixed, but it is possible to download new software to this computer.

☑ Can the software be re-written to avoid this stuck-at fault? ☑ Explain.

No problem. Have your compiler people avoid type-I destination registers that match the destination of the instruction two instructions back (the instruction before the previous one). In the solution to the previous part that would mean the destination of the `ori` would need to be changed from `r1` to some other free register, say `r9`. The change from `r1` to `r9` would need also to be made in instructions that follow the `ori`.

Problem 3: Appearing below is the slightly lower cost MIPS implementation, including the control logic from the 2020 Midterm Exam solution. Design the control logic for the select signal labeled E. *Hint: Not much needs to be added if some existing logic is used.* The SVG source for the diagram can be found at `https://www.ece.lsu.edu/ee4720/2024/hw03-lite-logic-e.svg`.

☑ Design control logic for select signal E.

Solution appears below in blue, along with a code fragment to help explain the solution. The lower input of the E mux is used if a bypass is needed from either ME or WB for the `rs` register. In the example code fragment that bypass is from ME, from the `add` to the `sub`. There is already logic to detect the dependencies between the `rs` source and the two preceding instructions. A new OR gate checks whether either dependence is present, producing the new `rs Byp` signal which will be used for the E select signal. Of course, `rs Byp` is computed when the instruction needing the bypass (such as `sub`) is in ID, so the signal is put through the ID/EX pipeline latch so that it can be used when the instruction is in EX.

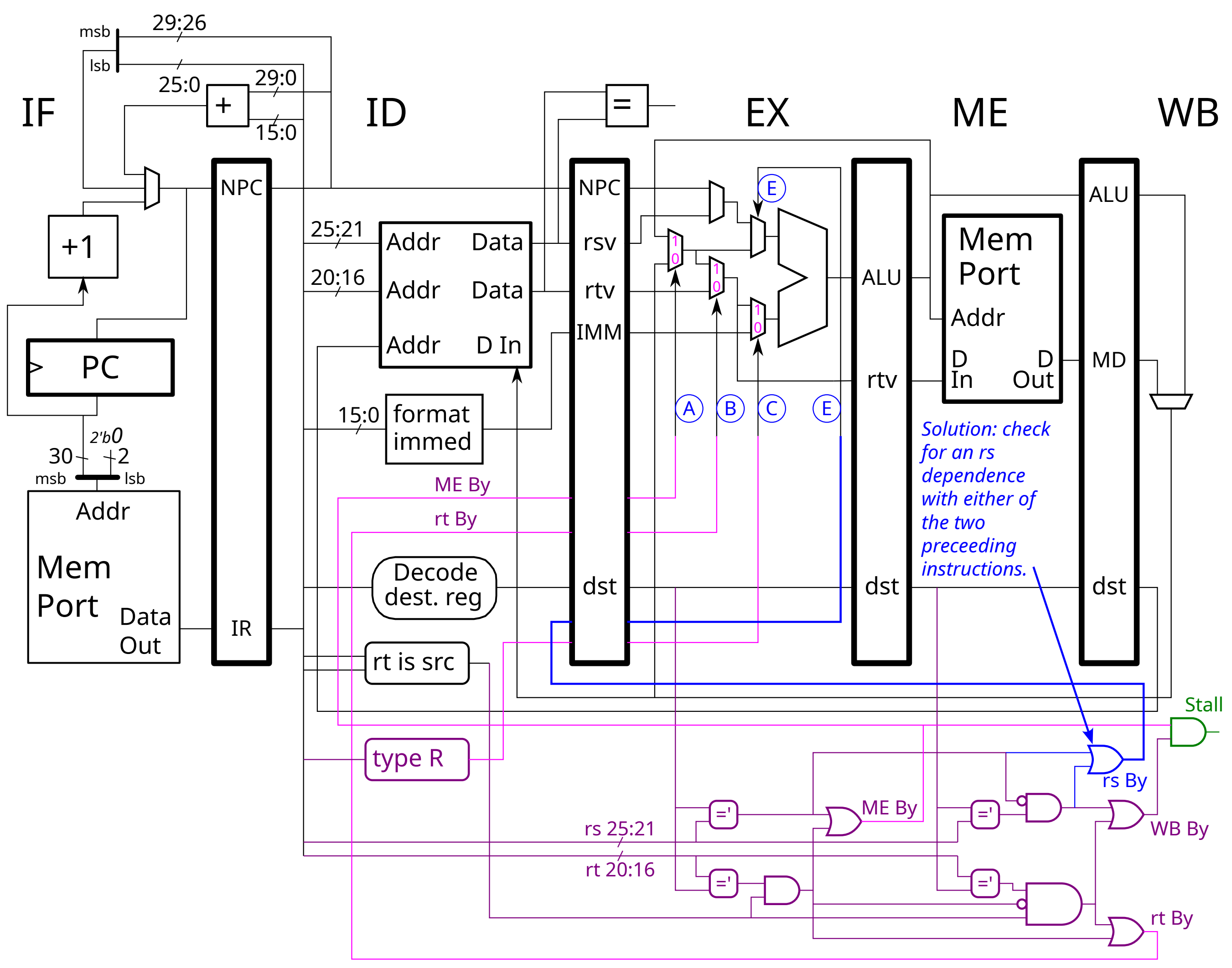
In the example below, E should be 1 for the `sub` instruction (due to dependence through `rs` register) but 0 for the `xor` instruction (no dependence through the `rs` register). For the `sub` instruction the output of the new OR gate is 1 in cycle 2 (detecting the dependence carried by `r1`), the bypass is used when the `sub` is in EX, in cycle 3. For the `xor` instruction the output of the new OR gate is 0 in cycle 3 (neither of the last two instructions writes `r7`, E doesn't care about the `rt` register, `r4`), the `EX.rsv` value is used when the `xor` is in EX, in cycle 4.

Common Errors: Signal E should be 1 *only if* there is a dependence through the `rs` register. It is wrong to set E to 1 if there is a dependence with the `rt` register but not with the `rs` register.

Another common error was to connect the control logic directly to the multiplexor select signals. *(I will not show an example of this incorrect connection lest anyone remember the connection but not that its wrong.)* The control logic is computing select signals for the instruction in ID, those select signals will be used one cycle later when the instruction is in EX, and for that reason they pass through the ID/EX pipeline latch. If those control signals instead were to connect directly to the multiplexor select signals they would be affecting the previous instruction. That's like ordering cheese in one of those assembly-line sandwich shops, and having them put the cheese not on your sandwich, but the sandwich ordered by the person immediately ahead of you in line.

```
# Cycle          0  1  2  3  4  5      # Example used to explain solution.
add R1, r2, r3   IF ID EX ME WB
sub R4, R1, r5      IF ID EX ME WB     # E=1 was computed in cyc 2, used in cyc 3.
xor r6, r7, R4         IF ID EX ME WB  # E=0 was computed in cyc 3, used in cyc 4.
E                      0  1  0         # Cycle in which E is used.
# Cycle          0  1  2  3  4  5
```

Changes appear below in blue. Explanation is on previous page.


IF
ID
EX
ME
WB
msb
lsb
29:26
25:0
29:0
15:0
+
+1
PC
NPC
IR
30
2
2'b0
msb
lsb
Addr
Mem Port
Data Out
25:21
20:16
Addr
Data
Addr
Data
Addr
D In
15:0
format immed
=
NPC
rsv
rtv
IMM
dst
ALU
rtv
dst
Mem Port
Addr
D In
D Out
ALU
MD
dst
A
B
C
E
E
Solution: check for an rs dependence with either of the two preceeding instructions.
ME By
rt By
Decode dest. reg
rt is src
type R
Stall
rs By
ME By
WB By
rt By
rs 25:21
rt 20:16
='