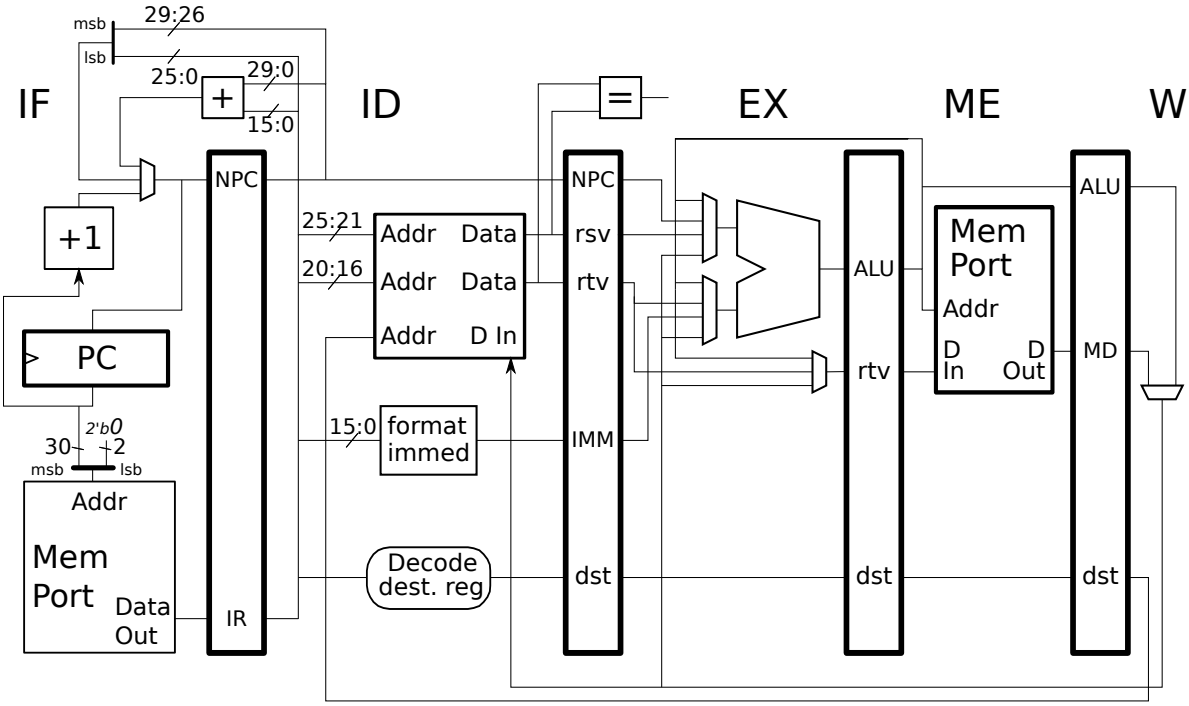
LSU EE 4720 **Homework 2** Solution **Due: 17 February 2017**

Problem 1: The following problems are from the 2016 EE 4720 Final Exam.

(*a*) *The following problem appeared as 2016 EE 4720 Final Exam Problem 2a. (Just 2a, not 2b).* Show the execution of each of the two code fragments below on the illustrated MIPS implementations. All branches are taken. Don't forget to check for dependencies.



The solutions appear below. Notice that there is no need for a stall in the first example because `r1` can be bypassed and that in the second example because `beq` is resolved in `ID` the target can be fetched while the branch is `EX`, cycle 2 below. Instruction addresses have been added to the second code fragment for use in the next subproblem's solution.

```
# CODE SEQUENCE A -- SOLUTION
# Cycle         0  1  2  3  4  5
add r1, r2, r3  IF ID EX ME WB
sub r4, r1, r5     IF ID EX ME WB
```

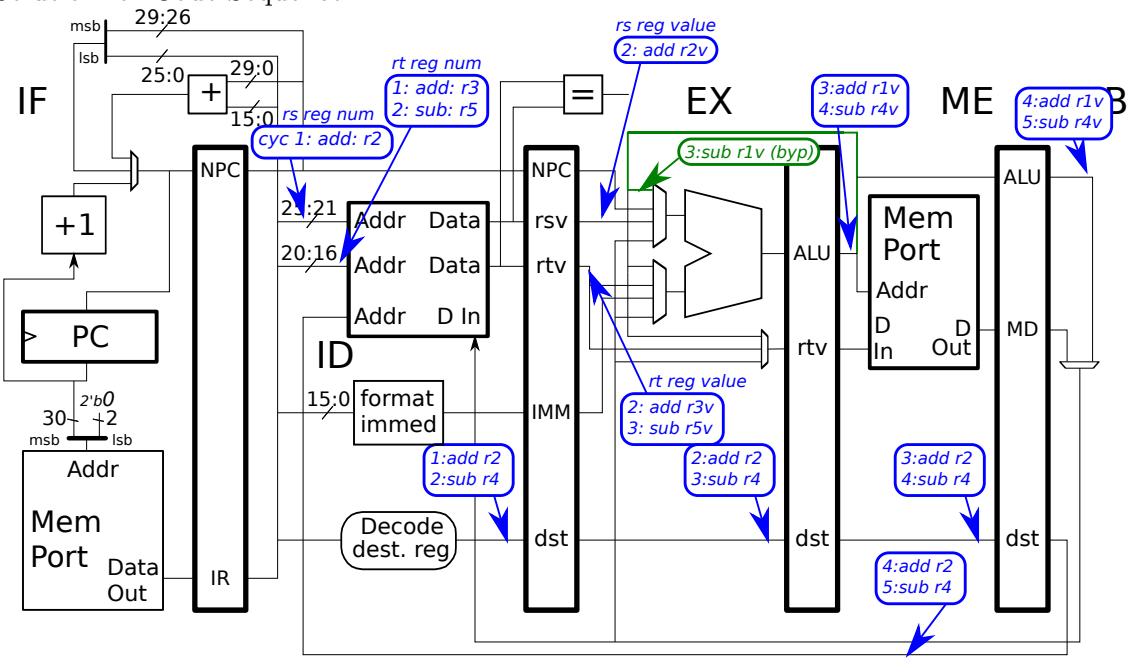
Show execution of the following code sequence.

```
# CODE SEQUENCE B -- SOLUTION
# Cycle           0  1  2  3  4  5  6
beq r1, r1 TARG   IF ID EX ME WB
or  r2, r3, r4       IF ID EX ME WB
sub r5, r6, r7
xor r8, r9, r10
TARG:
lw r10, 0(r11)          IF ID EX ME WB
# Cycle           0  1  2  3  4  5  6
```

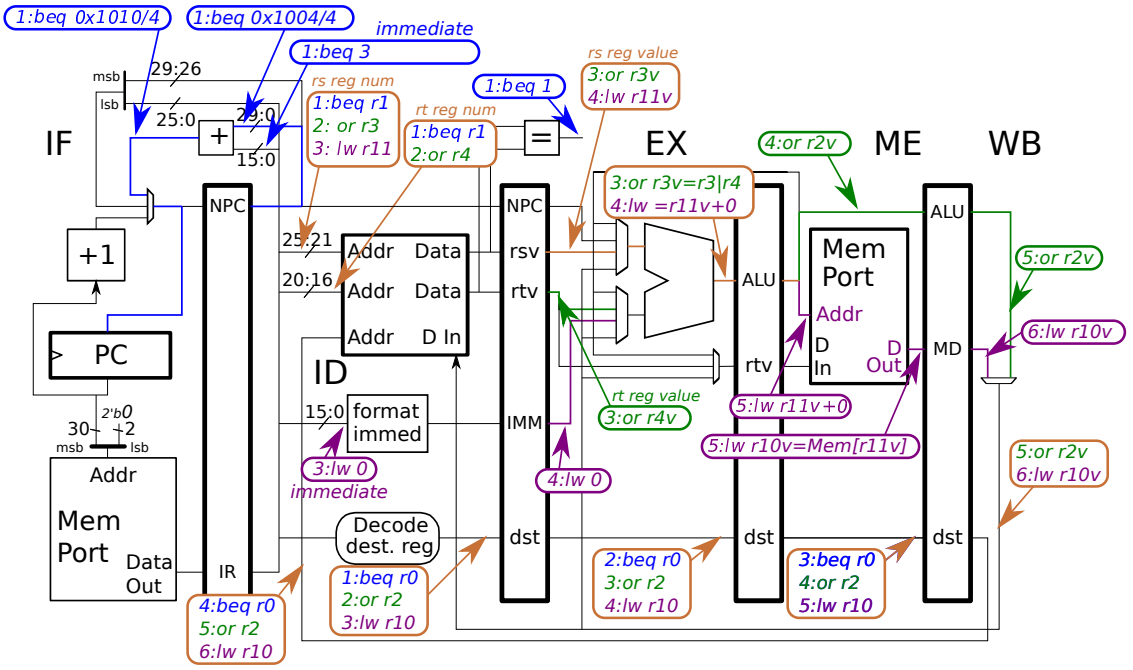
(*b*) For each of the two code sequences above label each used wire on the diagram below with: the cycle number, the instruction, and if appropriate a register number or immediate value. For example, the register file input connected to `ID.IR` bits 25:21 should be labeled $\boxed{\text{1: add, =2}}$ because in cycle 1 the `add` instruction is using that register file input to retrieve register `r2`. See the illustration below. **Only** label wires that are **used** in the execution of an instruction. For example, there should not be a label $\boxed{\text{2: sub, =1}}$ because the value of `r1` will be bypassed. Instead, label the bypass path that is used. Pay particular attention to wires carrying branch information and to bypass paths. Look through old homeworks and exams to find similar problems.

The solutions appear on the next page. For clarity, some paths are highlighted with the same color as the instruction that uses the path.

For Solution for Code Sequence A:



Solution For Code Sequence B:



Problem 2: *The following problem is from 2016 EE 4720 Final Exam Problem 5a.* Suppose that a new ISAis being designed. Rather than requiring implementations to include control logic to detect dependencies the ISA will require that dependent instructions be separated by at least six instructions. As a result, less hardware will be used in the first implementation.

(a) Explain why this is considered the wrong approach for most ISAs.

Short Answer: Because ISA features should be tied as little as possible to implementation features. The separation requirement in the new ISA is based on a particular implementation, one with five or six stages and that lacks dependency checking.

Long Answer: An ISA should be designed to enable good implementations over a range of cost and performance goals. It is reasonable to assume that low hardware cost was a goal of the first implementation and, as stated in the problem, the ISA's separation requirement helped achieve that goal. So the separation requirement would be a good idea **if there was only going to be one implementation or if cost constraints and technology wouldn't change**. However, ISAs are usually designed for a wide product line and a long lifetime. The separation requirement would become a burden if in the future a higher-performance, higher-cost implementation was needed.

(b) What is the disadvantage of imposing this separation requirement?

Short Answer: It would make a higher-performance implementation that included bypass paths pointless since the compiler would be forced to separate dependent instructions, possibly by inserting `nop`s, and so the bypass paths would never be used.

Long Answer: Call the ISA with the separation requirement, ISA S (strict), and one that lacked the separation requirement but was otherwise identical, ISA N (normal). Consider a low-cost implementation of each ISA. Neither implementation would include bypass paths but the implementation of ISA N would need to check for dependencies and stall if any were found, making the cost slightly higher than that of the implementation if ISA S. Now consider a code fragment compiled for both ISAs in which the version for ISA S requires `nop`s to meet the separation requirement. (The version for ISA N has fewer instructions.) Now consider their execution on their respective implementations. The execution times should be the same because for each `nop` in the ISA S implementation there will be a stall in the ISA N implementation.

So the performance of the low-cost implementations of the ISAs are about the same, but the cost of the ISA N version is slightly higher. (Note that the hardware for checking dependencies is of relatively low cost since it operates on register numbers, 5 bits each in many ISAs. That is in contrast to the cost of the hardware for bypasses, which are 32 or 64 bits wide and include multiplexors.)

Next, consider high-performance implementations. For ISA N we can add bypass paths, as we've done in class. As a result, some instructions that would stall in the low-cost implementation would not stall in the high-performance implementation, and so code would run faster. In contrast, code for ISA S would still have `nop` instructions since the ISA itself imposes the instruction separation requirement. That would make it much harder to design higher performance implementations.

Therefore, the disadvantage of the instruction separation requirement is that it leads to much higher-cost high-performance implementations with only a small cost benefit for the low-cost implementation.

Grading Note: Several students incorrectly answered that a disadvantage of the separation requirement is that it would be tedious and error prone for hand (human) coding, and that it would require that the compiler schedule instructions rather than having the hardware check for dependencies and stall when necessary.

It is 100% true that hand-coding assembly language with such a requirement would be tedious, especially considering branch targets. However, an ISA is designed to facilitate efficient implementations, not to improve assembly language programmers' productivity. (An assembler can still be helpful by pointing out likely separation issues.) Also, if something can be done equally well in the compiler and in hardware, it should be done in the compiler because the cost of compiling is borne beforehand, by the developer. In contrast, the higher cost of the hardware is payed by every customer and the energy of execution is expended each time the program is run.