EE 4720 **Homework 3** Solution **Due: 12 March 2001**

Problem 1: Consider three variations on the Chapter-3 DLX implementation. In implementation I the FP Add unit has an initiation interval of 2 and a latency of 3. In implementation II there are two FP Add units, **each unit** has an initiation interval of 4 and a latency of 3. In implementation III the FP Add unit has an initiation interval of 1 and a latency of 3. Other features of the implementations are identical. All implementations are fully bypassed.

Write two programs. Program $\mathcal{A}$ should run slower on implementation I than on implementations II and III. Program $\mathcal{B}$ should run the same speed on implementations I and II and faster on implementation III. For this problem base program speed on the time from the fetch of the first instruction to the WB of the last instruction.

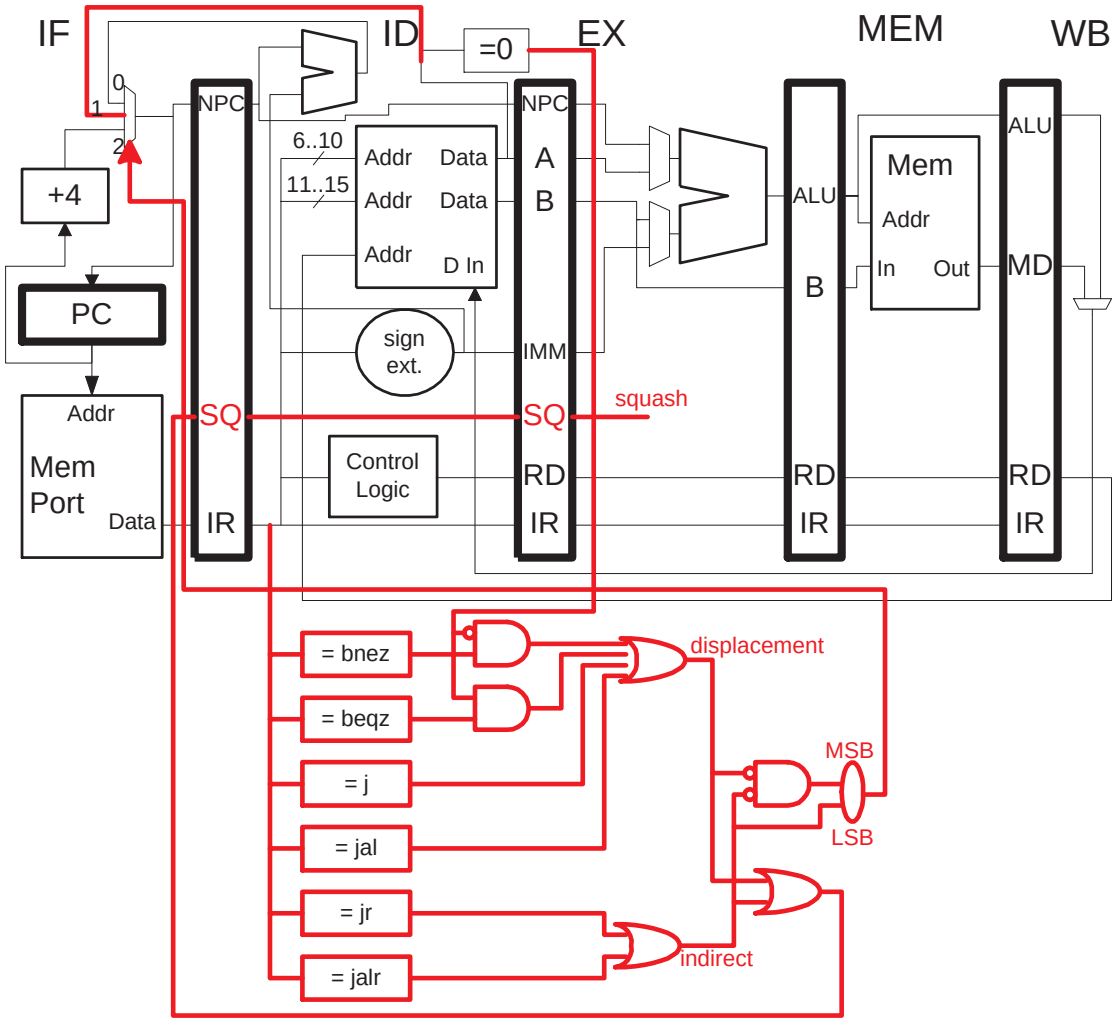
Show pipeline execution diagrams for each program on each implementation. The programs need be no longer than four instructions each.

```
! Solution
! Program A

! I
addd f0, f2, f4      IF ID A1 A1 A2 A2 WB
addd f6, f8, f10        IF ID -> A1 A1 A2 A2 WB

! II
addd f0, f2, f4      IF ID A  A  A  A  WB
addd f6, f8, f10        IF ID B  B  B  B  WB

! III
addd f0, f2, f4      IF ID A1 A2 A3 A4 WB
addd f6, f8, f10        IF ID A1 A2 A3 A4 WB

! Program B
! I
addd f0, f2, f4      IF ID A1 A1 A2 A2 WB
addd f6, f8, f10        IF ID -> A1 A1 A2 A2 WB
addd f12, f14, f16         IF -> ID -> A1 A1 A2 A2 WB

! II
addd f0, f2, f4      IF ID A  A  A  A  WB
addd f6, f8, f10        IF ID B  B  B  B  WB
addd f12, f14, f16         IF ID ----> A  A  A  A  WB

! III
addd f0, f2, f4      IF ID A1 A2 A3 A4 WB
addd f6, f8, f10        IF ID A1 A2 A3 A4 WB
addd f12, f14, f16         IF ID A1 A2 A3 A4 WB
```

Problem 2: Modify the pipeline below so that it can execute `jr` instructions and add PC mux control logic.

- Modify the pipeline so that it can execute `jr` instructions. (See Spring 1999 Homework 3, `http://www.ee.lsu.edu/ee4720/1999/hw03_sol.pdf`.)
- Include control logic for the multiplexor that connects to `PC`. The control logic should correctly handle branch and jump instructions. Interrupts should be ignored. To recognize instructions use boxes such as `= bnez`, the outputs will be 1 if the instruction matches.
- Show the logic for a `squash` signal for use in EX to squash the fall-through instruction on a taken branch. (The fall through instruction could have been squashed in IF or ID, but for this problem it will be squashed in EX.)

Changes shown in red.



Problem 3: How would the hardware designed above have to be modified if DLX had two-slot (yes, two slots!) delayed branches? Jumps still have no delay slots. Ignore interrupts, they will be considered in the next problem.

Problem 4: In an ISA without delayed branches it would be sufficient for the hardware to save the PC when an exception occurs. Why would this not be sufficient on a system with delayed branches. Provide an example illustrating what might go wrong.

One could not properly resume execution if the faulting instruction were in a branch delay slot. To resume execution properly the exception handler needs the PC of the faulting instruction and the PC of the next instruction to execute. In most cases the next instruction to execute is at PC+4 (assuming four-character instructions) but if the faulting instruction were in the delay slot of a taken branch the next instruction to execute would be the branch target.

Suppose that `lw` raises an exception in the example below. If the handler only saves the address of `lw`, `0x1004`, then when execution resumes the branch will not be taken. By saving `0x1004` and the address of the next instruction, `0x2000`, the handler can restore execution so that the branch will be taken.

```
 0x1000: beqz r0, TARGET
 0x1004: lw   r2, 0(r3)
 0x1008: add  r3, r3, r4

TARGET:
 0x2000: or   r5, r6, r7
```

Problem 5: The Hewlett Packard *Precision Architecture RISC 2.0 (PA-RISC 2.0)* uses an *instruction address offset queue* rather than a plain-old program counter. See the PA-RISC 2.0 Architecture [Manual], `http://devresource.hp.com/devresource/Docs/Refs/PA2_0/acd-1.html`. Ignore the material on [address] space IDs and privilege levels. Concentrate on the material in Chapter 4 and 5 and use the index.

PA-RISC 2.0 has delayed branches. Explain how the use of an instruction address offset queue rather than a PC helps with the difficulty alluded to in the previous problem.

The address of the executing instruction and the next instruction can be saved and restored as a unit.

Problem 6: Explain the relationship between the terms *interrupt*, *hw interrupt*, *exception*, and *trap* provided in class and the terms *interruption*, *fault*, *interrupt*, *trap*, and *check* defined for PA-RISC 2.0. Explain the *relationships*, do not simply provide definitions.

A PA-RISC interruption is analogous to the term interrupt used in class.

A PA-RISC fault is a category of exception, as used in class. A PA-RISC trap is another category of exception, as used in class. That is, some of what are called exceptions in class are called faults in PA-RISC, and other exceptions are called traps. (Traps are usually due to programmer error or bad input, while faults indicate that the OS has to take some routine action to keep the program running.) Note that the meaning of the term trap used in class is completely different from a PA-RISC trap.

A PA-RISC interrupt is analogous to the term hardware interrupt used in class.

A PA-RISC check is a specific type of hardware interrupt, no special term was used in class.

Problem 7: Name a difference between the trap table used in Sun SPARC V8 (presented in class and described in the SPARC Architecture Manual V8, `http://www.ee.lsu.edu/ee4720/sam.pdf`) and the interruption vector table used in PA-RISC 2.0.

The Sun SPARC table holds four instructions, the PA-RISC table holds eight instructions, otherwise they are very similar.