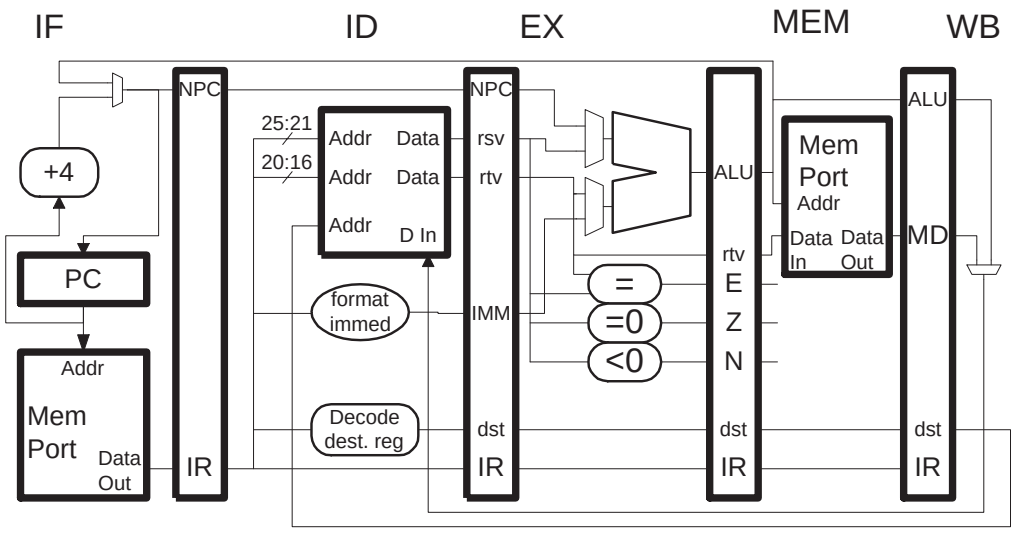
LSU EE 4720 **Homework 3** **Due: 19 March 2003**

Problem 1: Consider the code below.

```
# Cycle              0  1
add $t1, $t2, $t3    IF ID
sub $t4, $t5, $t1
lw  $t6, 4($t1)
sw  0($t4), $t6
```

(*a*) Show a pipeline execution diagram for the code running on the following illustration. Note that the `add` is fetched in cycle zero.

- Take great care in determining the number of stall cycles.

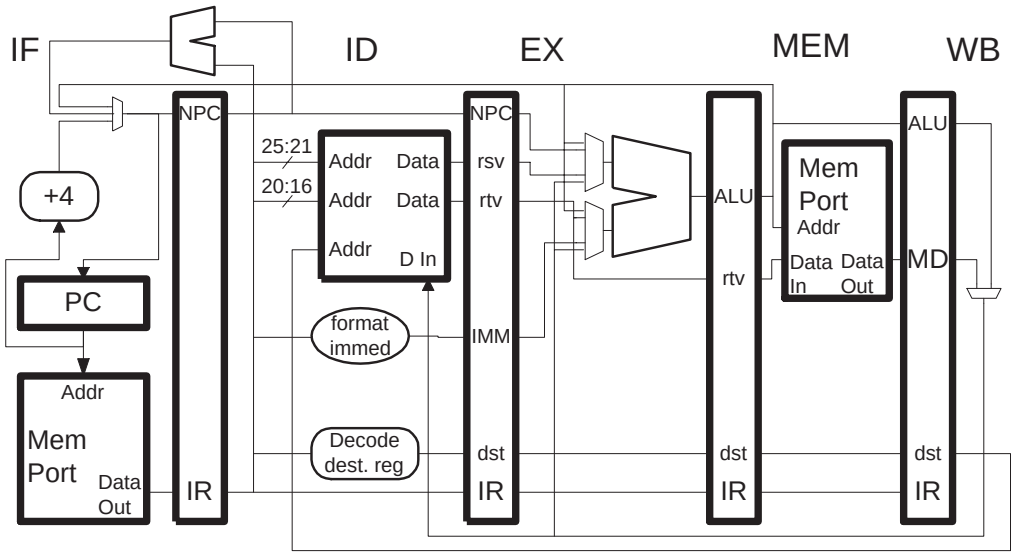


Problem 2: The code below is the same as in the previous problem.

```
# Cycle              0  1
add $t1, $t2, $t3    IF ID
sub $t4, $t5, $t1
lw  $t6, 4($t1)
sw  0($t4), $t6
```

(*a*) Show a pipeline execution diagram (PED) of the code running on the system below.

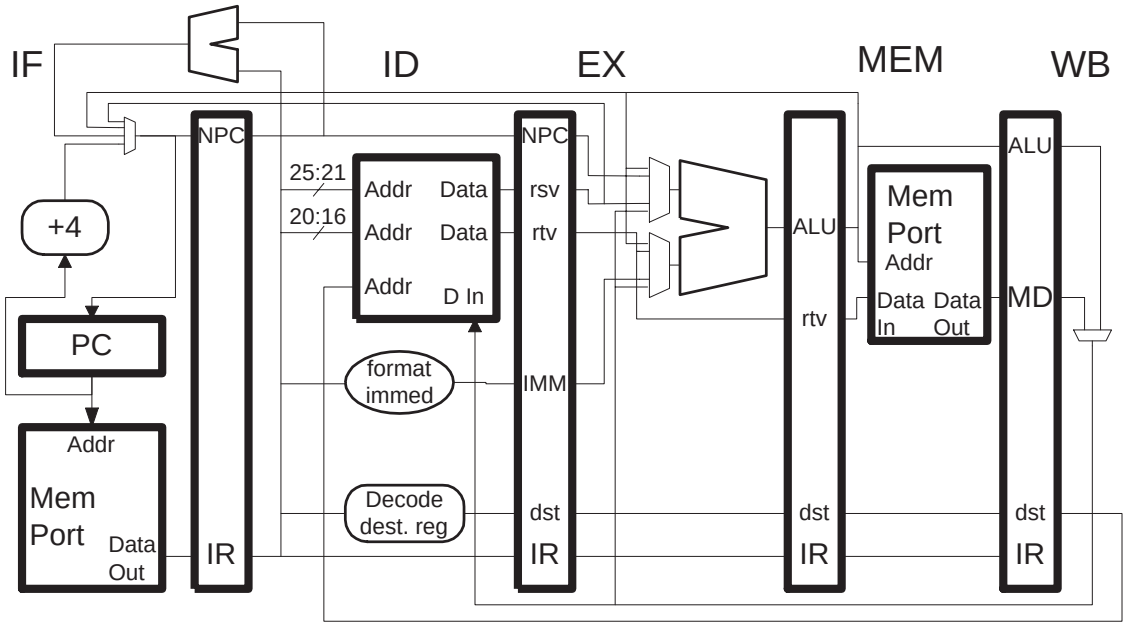
(*b*) In the PED circle each stage that *sends* a bypassed value. In the diagram label each bypass path with the cycle in which it is used. To avoid ambiguity, label the end of the path (at the mux input).



The problem below is tricky. If necessary use Spring 2001 Homework 2 problem 3 for practice.

Problem 3: The program below has an infinite loop and runs on the bypassed implementation below.

```
        # Initially $t0 = LOOP (address of jalr)
LOOP:
        jalr $t0
        addi $t0, $ra, -4
        bne $t0, $0 LOOP
        addi $t0, $t0, -4
```



(*a*) Show a pipeline execution diagram for this program up to a point at which a pattern starts repeating. Beware, the loop is tricky! Read the fine print below for hints.

Note that `jalr` reads and writes a register. The `jalr` instruction should be fetched twice per repeating pattern. The `addi` instruction should be fetched three times per repeating pattern.

(*b*) In the PED circle each stage that *sends* a bypassed value. In the diagram label each bypass path with the cycle in which it is used. To avoid ambiguity, label the end of the path (at the mux input).

(*c*) Determine the CPI for a large number of iterations.

Problem 4: SPARC V9 has multiple floating-point condition code (FCC) registers. See the references pages for more information on SPARC V8 and V9.

(*a*) Write a program that uses multiple FCC's in a way that reduces program size. As an example, the SPARC program below uses a single FCC. (To solve this problem first find instructions that set and use the multiple FCC registers in the SPARC V9 Architecture Manual. Then write a program that needs the result of one comparison (say, $a < b$) several times while also using the result of another (say, $c > d$). A program not using multiple condition code registers should have to do the comparison multiple times whereas the program you write does each comparison once.)

```
!   10       !    {
!   11       !      sum = sum + 4.0 / i;   i += 2;

                       .L77000016:
/* 0x0020   11 */ fdivd %f4,%f30,%f6
/* 0x0024      */ faddd %f30,%f2,%f8

!   12       !      sum = sum - 4.0 / i;   i += 2;

/* 0x0028   12 */ faddd %f8,%f2,%f30
/* 0x002c      */ fcmped %f30,%f0
/* 0x0030      */ fdivd %f4,%f8,%f8
/* 0x0034   11 */ faddd %f10,%f6,%f6
/* 0x0038   12 */ fbl .L77000016
/* 0x003c      */ fsubd %f6,%f8,%f10

!   13       !    }
```

(*b*) SPARC V9 is the successor to SPARC V8, which has only one FCC register. (SPARC V9 implementations can run SPARC V8 code.) Did the addition of multiple FCC's require the addition of new instructions or the extension of existing instructions? Answer the question by citing the old and new instructions and details of their coding.

(*c*) Do you think the designers of SPARC V8 planned for multiple FCC's in a future version of the ISA?