EE 4720 **Homework 3** **Due: 20 March 2002**

Problem 1: The exception mechanism used in the MIPS 32 ISA differs in some ways from the SPARC V8 mechanism covered in class. See Chapter 7 in `http://www.ece.lsu.edu/ee4720/sam.pdf` for the SPARC V8 exception information and `http://www.ece.lsu.edu/ee4720/mips32v3.pdf` for a description of the MIPS mechanism. The MIPS description is a bit dense, so start early and ask for help if needed.

(*a*) Describe how the methods used to determine which exception was raised differ in SPARC V8 and MIPS 32. Use an illegal (reserved) instruction error as an example. Shorter answers will get more credit so concentrate on explaining how the processor identifies the exception (was it an illegal instruction, an arithmetic overflow, etc) and avoid irrelevant details. For example, details on how the processor switches to system mode is irrelevant.

(*b*) Where do the two ISAs store the address of the faulting instruction? Both ISAs have delayed branches, so why does SPARC store two return addresses while MIPS gets away with one?

(SPARC registers are organized like a stack, on a procedure call a `save` instruction "pushes" a fresh set of registers on the stack, and a `restore` instruction "pops" the registers, returning to the previous set. The set of visible registers is called a window. This mechanism reduces the need to save and restore registers in memory. This piece of information is needed for the previous problem.)

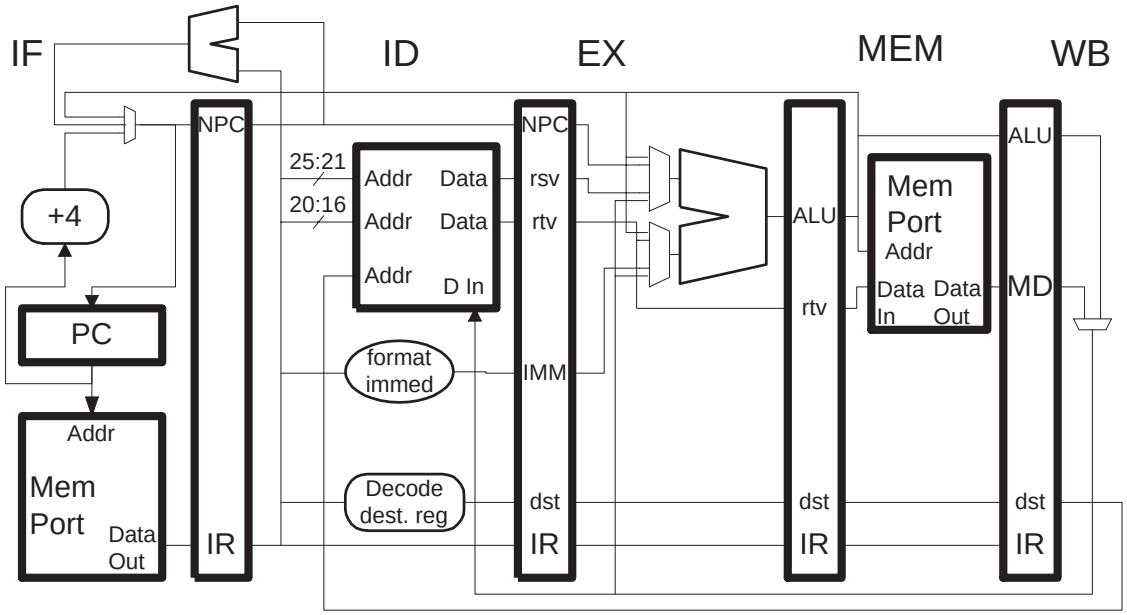
Problem 2: The pipeline execution diagram below is for code running on a MIPS implementation developed just for this homework problem! Note that the program itself is missing. The dog deleted it. The `M_` and `A_` refer to parts of the multiply and add functional units with segment numbers omitted for this problem. A `WBx` indicates that an instruction does not write back to avoid a WAW hazard.

```
IF ID M_ M_ M_ M_ M_ M_ WB
   IF ID ----> M_ M_ M_ M_ M_ M_ WB
      IF ----> ID ----> A_ A_ WB
               IF ----> ID M_ M_ M_ M_ M_ M_ WBx
                        IF ID A_ A_ WB
                           IF ID A_ A_ WB
```

(*a*) Write a program consistent with the diagram. Pay attention to dependencies.

(*b*) Identify the latency and initiation interval of the functional units. Fill in the segment numbers.

Problem 3: In the MIPS implementation below (also shown in class) branches are resolved in the ID stage. Resolution of a branch direction (determining whether it was taken) must wait for register values to be retrieved and, for some branches, compared to each other. Suppose this takes too long.



(*a*) Show the modifications needed to do the equality comparison in the EX stage. The modified hardware must use as little additional hardware as possible and, to maximize performance, should only do an EX-stage equality comparison when necessary. Don't forget about branch target address handling. *Hint: The modifications are easy.*

(*b*) Write a code fragment that runs differently on the two implementations and show pipeline execution diagrams for the code on the two implementations.

(*c*) The table below lists SPARC instructions and indicates how frequently they were used when running TEX to prepare this homework assignment. (Many rows were omitted to save space, so the "%exec" column will not add to 100%.) Suppose that the instruction percentages are identical for MIPS (which means totally ignoring the `cc` instructions). Assume that SPARC `be` and `be,a` are equivalent to MIPS `beq`, SPARC `bne` and `bne,a` are equivalent to MIPS `bne`, and that the other branch instructions (they begin with a b), are equivalent to branch instructions that compare to zero (`bgez`, etc.).

Suppose the clock frequency of the original design is 1.0000 GHz. Based on the data below and making any necessary assumptions, for what clock frequency would the new design run a program in the same amount of time as the old one? What column would you add (what additional data do you need) to the table to make your answer more precise?

Assume that floating-point instructions are insignificant and that there are no stalls due to memory access.

opcode	#exec	%exec
subcc	4659360	12.6187%
lduw	4521722	12.2459%
add	4159629	11.2653%
or	3110542	8.4241%
sethi	3066797	8.3056%
stw	1848293	5.0056%
sll	1402122	3.7973%
be	1393475	3.7739%
jmpl	1140223	3.0880%
call	1088068	2.9467%
ldub	1064918	2.8841%
bne	936493	2.5362%
stb	687981	1.8632%
srl	609402	1.6504%
save	526477	1.4258%
restore	526474	1.4258%
bne,a	453545	1.2283%
nop	433253	1.1734%
bge	429978	1.1645%
ldsb	429497	1.1632%
orcc	382947	1.0371%
and	370967	1.0047%
be,a	360057	0.9751%
sub	354847	0.9610%
ba	321970	0.8720%
bl	297715	0.8063%
andcc	270465	0.7325%
bgu	235304	0.6373%
bl,a	216074	0.5852%
sra	204610	0.5541%
ble	198154	0.5366%
xor	185137	0.5014%
bcs	182153	0.4933%
addcc	155156	0.4202%
bleu	142755	0.3866%
bg	117582	0.3184%
mulscc	88681	0.2402%