Name _____

Computer Architecture
EE 4720
Midterm Examination

Friday, 25 October 2002, 10:40–11:30 CDT

Problem 1	_____	(30 pts)
Problem 2	_____	(13 pts)
Problem 3	_____	(13 pts)
Problem 4	_____	(44 pts)
Exam Total	_____	(100 pts)

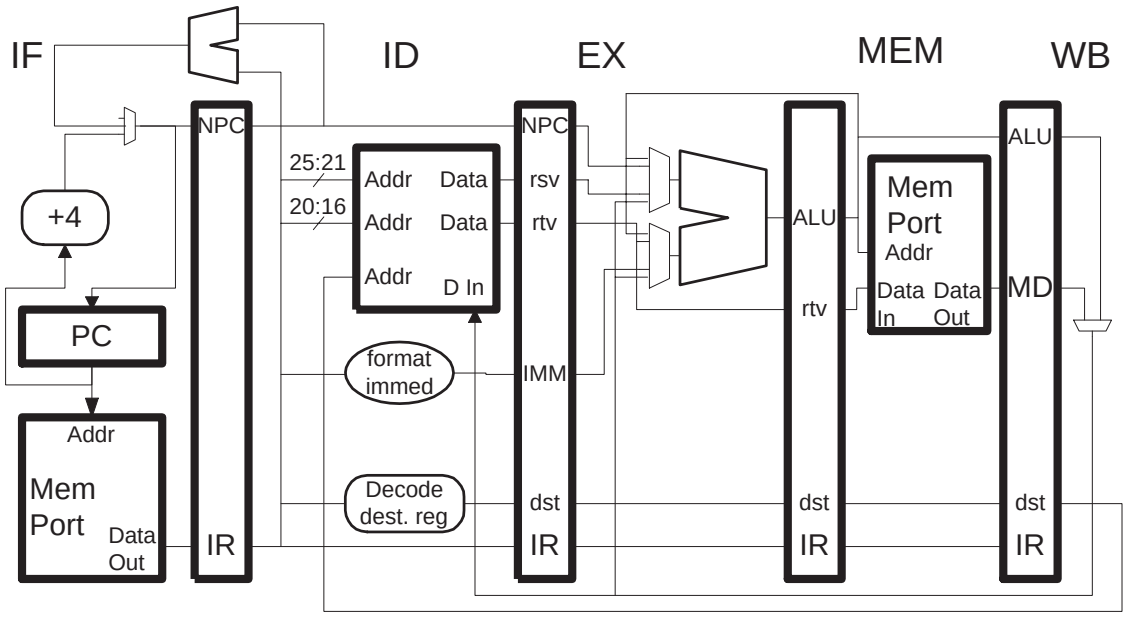
Alias _____

Good Luck!

Problem 1: A new MIPS branch instruction, `bieq rt,(rs) disp` (branch indirect equal) compares a register value to a memory location, if they are equal the branch is taken. The target is computed in the same way as other branches. In the code below, the contents of register `$s1` is compared to the contents of the memory location at address `$s2`. Like all MIPS control transfers, `bieq` has one delay slot. [30 pts]

(*a*) Modify the pipeline so that it can execute this new instruction. Show comparison units, multiplexors, and wires. **Do not** show control logic. For **partial credt** replace `bieq ...` with `bneq $s1,$s2, LOOP`.

- ☐ Use as much existing hardware as possible. **Do not** add a new memory port.
- ☐ The change should not reduce the clock frequency.
- ☐ Include the comparison unit for the branch condition.
- ☐ Add bypass paths so that the code below executes as shown.
- ☐ Label bypass paths with the cycle in which they are used. Include existing and any added bypass paths.
- ☐ Label the path carrying the branch target address with the cycle in which it is used.




```
LOOP:  # Assume bieq always taken.
 # Cycle               0    1    2    3    4    5    6    7    8    9   10
 srl  $s1, $s1, 1      IF   ID   EX   ME   WB

 addi $s2, $s2, 1           IF   ID   EX   ME   WB

 bieq $s1, ($s2), LOOP      IF   ID   EX   ME   WB

 sw   $s1, 100($s2)

 add  $0, $0, $0

 sub  $0, $0, $0
 or   $0, $0, $0
 xor  $0, $0, $0
```

Problem 1, continued:

(*b*) The pipeline execution diagram on the previous page only shows the first iteration.

☐ Continue the pipeline execution diagram to the beginning of the second iteration (when `srl` is fetched), *consistent with your solution to the first part.*

☐ Show which instruction is in each stage of the pipeline a cycle eight. Include squashed instructions, if any.

☐ Determine the CPI for a large number of iterations.

Problem 2: Convert the MIPS program below to SPARC, making use of condition codes. [13 pts]

- ☐ Make reasonable guesses about instruction names. Reasonable guesses will receive full credit.
- ☐ For the branch instructions, explain what the name stands for.
- ☐ Explain how each line of code uses the condition codes.
- ☐ Use the minimum number of registers.
- ☐ Do not rearrange instructions.

```
sub $s6, $s1, $s2
blt $s6, TARGET1
and $s3, $s6, $s4
beq $s3, $0, TARGET2
nop
beq $s6, $0 TARGET3
add $s5, $s6, $s3
```

Problem 3: Sometimes it's necessary to jump to a location specified in the program. [13 pts]

(*a*) SPARC V8 includes an instruction that can jump anywhere in the address space using an address specified in the instruction, for example, `call 0xabcd1234`.

☐ How does the `call` instruction, which is 32 bits, manage to specify a 32-bit jump target?

☐ Switching now to MIPS, show the minimum number of instructions needed to jump to address `0xabcd1234`. (If `jr` is used the instructions must put the address in a register. The address cannot be loaded from memory.)

(*b*) Without introducing a new format, add a new instruction to MIPS that would allow a jump to an arbitrary address using two instructions. The address must be specified in the instructions themselves.

☐ Use the instruction in an example.

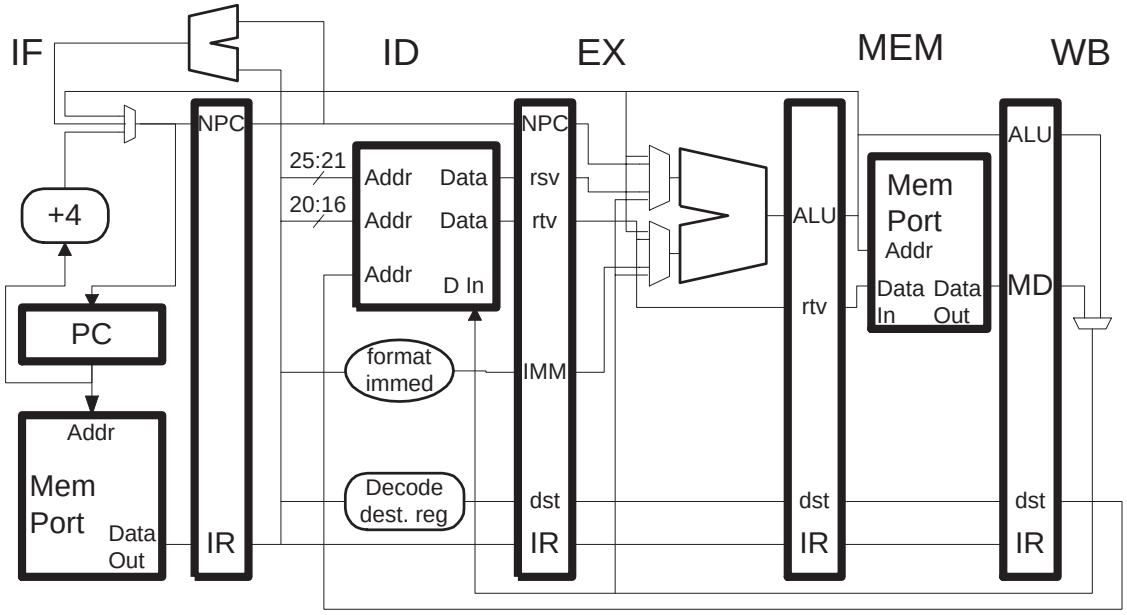
☐ Show the coding for the new instruction.

Problem 4: Answer each question below.

(*a*) Certain RISC ISA features are intended to facilitate pipelined implementations. [12 pts]

☐ Using the diagram below, briefly explain how RISC implementations benefit from fixed-length instructions and how things would be different with variable-length instructions.

☐ Using the diagram below, briefly explain how RISC implementations benefit from load/store instructions (restricting memory access to load instructions) and how relaxing the restriction would complicate things.



(*b*) Before there was MMX there was BCD. [12 pts]

☐ Why are binary coded decimal (BCD) and packed-integer data types superficially similar? What is an important difference between them?

☐ Describe a computation in which packed integer data types yield a performance advantage over ordinary integers. Show a brief advantage of the computation.

☐ Why were BCD data types added to some ISAs?

(*c*) Describe two compiler optimizations that reduce instruction count. [8 pts]

(*d*) The BAPCO benchmarks used by PC magazine and others evaluates a computer by timing the execution of a suite of benchmarks. The suite of benchmarks is drawn from common applications, such as Adobe Photoshop. The applications are in executable form, source code is not used.

Like BAPCO, SPEC CPU2000 consists of a suite of common applications, unlike BAPCO source code is used. Assume that the BAPCO and SPEC benchmarks are both well chosen and aimed at roughly the same user community. [12 pts]

☐ How is source code used in preparing the SPEC CPU 2000 benchmark results?

☐ Consider two processors, A and B. Suppose BAPCO ranks A faster but SPEC ranks B faster. How might the use of source code account for this difference?

☐ Taking in to account the differences due to the use of source code, who should make buying decisions based on BAPCO results? Who should make buying decisions on SPEC results?