CS152 Homework I, Fall 2005

Name: _____

SSID: _____

Homework I is due in class on Thursday September 29 at 11:10 AM. This class is the Mid-term I review session.

Late homeworks are NOT accepted. Thus, if you will not be attending the review session, you MUST make arrangements to hand off the homework to the instructor before classtime.

Homework will be graded on effort (did you make an honest attempt to solve each problem?), not correctness. We will distribute the correct answers for the homework in the review session, but we will probably not return the homework you hand in until after the exam. So, you may wish to make a copy for reference before you hand it in.

This homework will count for approximately 1% of your final grade.
The homework is based on the Mid-term I exam from Spring 05.
You may discuss the homework problems with fellow students and the TAs, but what you write down must be your own work (no copying the answers from someone else's homework). Good luck! John Lazzaro

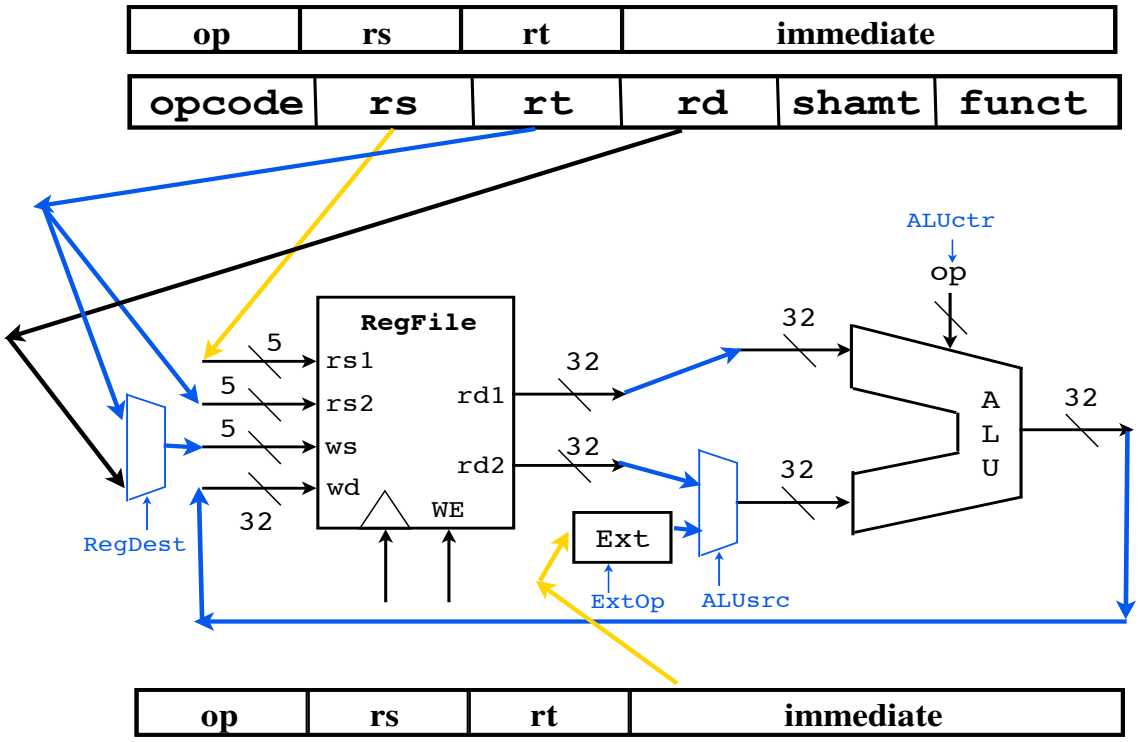
1 Logic Design (5 points)

We wish to use pipeline registers with a write enable control line in our design. However, our component library does not include registers with write enable, and our CAD system does not offer access to the clock input of sequential logic (thus, no manual clock gating).

Using a register and one other "standard" combinational component, design the write-enabled register. Show the schematic below.

2 Single Cycle Design (10 points)

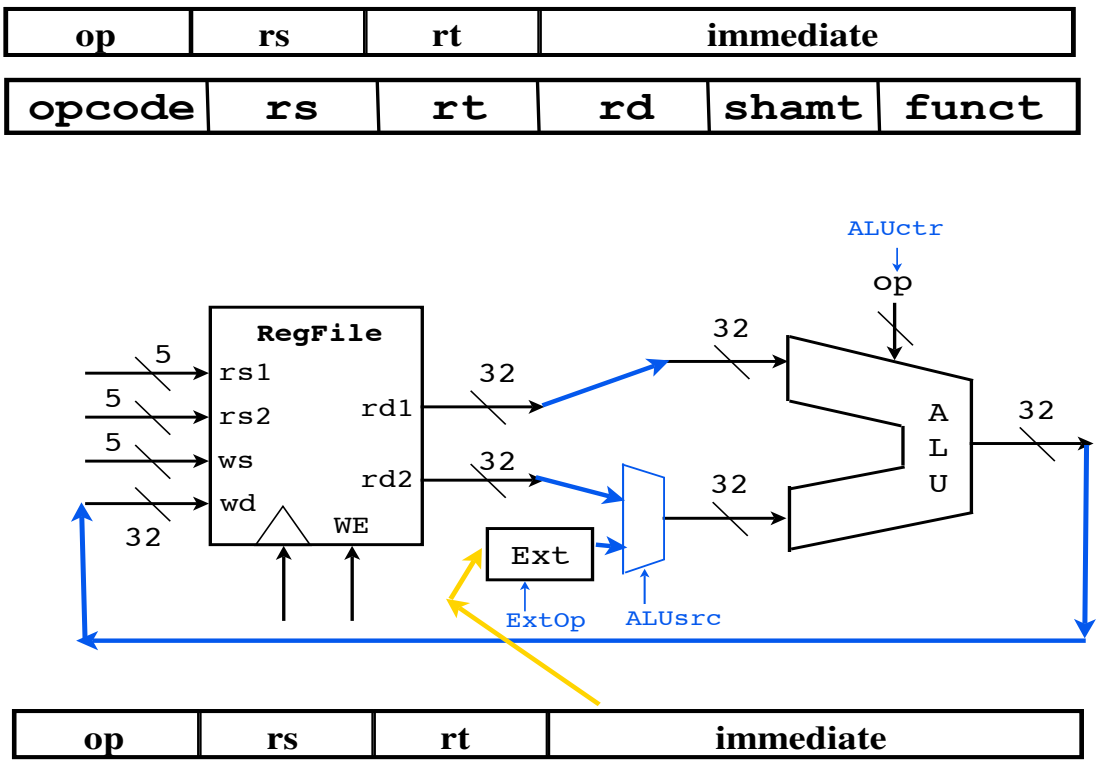
Below, we show a single-cycle datapath for R-format and I-format ALU instructions, which we derived in class.



This datapath implements the MIPS register semantics: the **rs** field always code the first source operand of an instruction, but the role of the second and third operands (source or destination) depends on whether the ALU instruction is R-format or I-format.

Imagine an alternative register semantics for MIPS. In this semantics, the **rs** field is always the destination register, the **rt** field is always the first source operand.

Question 2a (5 points). Redraw the datapath to implement the new semantics, adding wires and elements to the skeleton below. As in the original datapath, your revised datapath only needs to implement R-format and I-format ALU instructions (NOT branches, memory instructions, etc). Add the **minimal** number of elements needed to implement the new semantics.



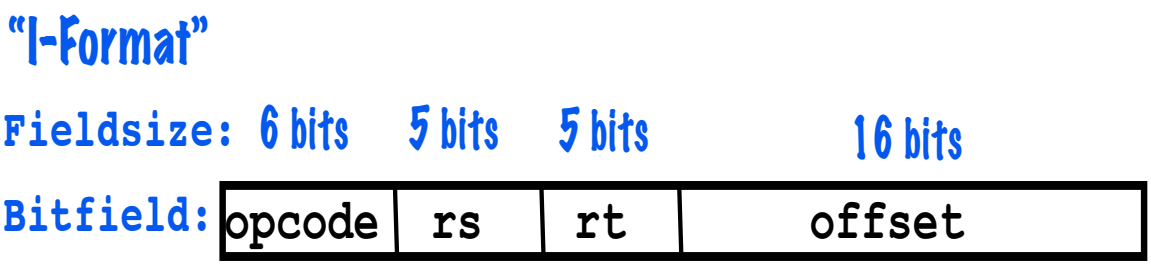
Question 2b (5 points). Is the critical path (CP) for the new datapath the same speed as the old datapath? Is it faster? Is it slower? Circle one of the answers below, then justify your answer in 15 words or less. To compare critical paths, only consider fan-out (wires that drive more gates are slower) and gate delay (adding an extra gate in a path slows it down). Assume the control signals for datapath elements (such as the RegDest, ExtOp, ALUsrc, ALUctr, and WE signals) DO NOT affect the critical path. Think carefully about this question - it is tricky.

`Circle one:  New CP faster      New CP slower         Same speed.`

Write justification below:

3 MIPS ISA and Pipelining (20 points)

Consider a MIPS implementation whose only memory instructions are LW and SW. Assume the ISA semantics are "no load-delay slot". Recall the instruction format of LW and SW instructions:



Consider an implementation that only permits the index registers of LW and SW to be R0. Thus:

```
LW $1, 4($0)
SW $1, 4($0)
```

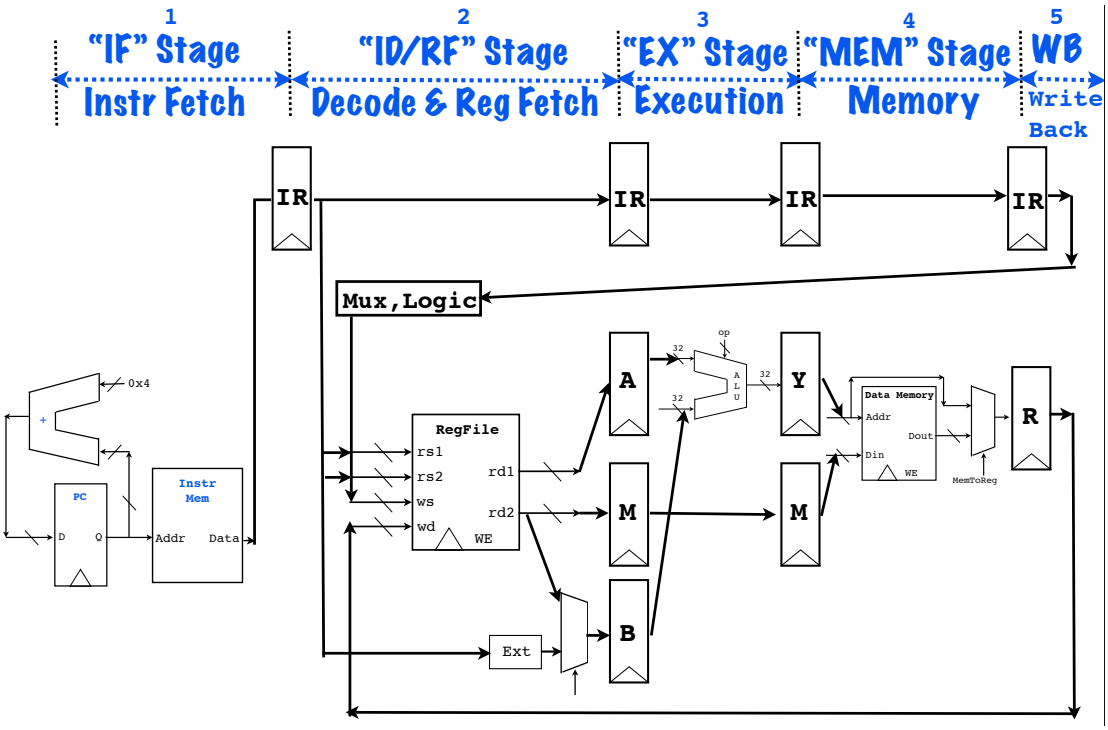
Would be legal instructions, but

```
LW $1, 4($2)
SW $1, 4($2)
```

would not. The questions below all refer to this implementation.

Question 3a (2 points). Is it possible for programs in this machine to load and store values for all 2^{32} memory locations to/from registers?

```
Circle one:     Yes     No
```



Question 3b (6 points). When the datapath above runs this code:

```
LW $1, 4($0)
ADD $3 $1 $1
```

it is necessary to stall the ADD in order to let the (no-load-delay) LW write the register file. If only R0 may serve as the LW index register, it is possible to modify the datapath to eliminate the stall. Describe, in 25 words or less, how to change the datapath to eliminate the LW stall.

Do not discuss datapath changes for other stalls (branches, etc). Just the stall triggered by the code above. Write your answer below:

Question 3c (6 points). We wish to hand-compile this C function to run on the implementation:

```
int a[1024];  /* global array, ints are 32 bit */

int total(int k)  {

int i = k;
int sum = 0;

if ((k < 0) || (k > 1008)) return 0;

while (i < k + 16)
  sum += a[i++];

return sum;
}
```

Assume there is no way for running programs to write the contents of instruction memory (a TFTP-like mechanism loads in the program once)

At the time of hand-compilation, you do NOT know the value k that will be passed in, and you do NOT know the values stored in a[]. As part of hand compilation, you ARE able to place a[] anywhere in memory you wish.

Is it possible to create this program, given that only R0 may be used as the index register for loads and stores? If not, explain why not. If so, describe how the hand-compiled program would get around the restriction. Use 35 words or less, plus one annotated figure (if you wish).

Question 3d (6 points). Reconsider Question 3c, but permit running programs to write the contents of instruction memory. Is is now possible to hand-compile this C function? If not, explain why not. If so, describe words how s program that took advantage of "self-modifying code" would work. Use 35 words or less, plus one annotated figure (if you wish).