CS 2733/2731, Computer Organization II
Fall Semester, 2003
Final Examination

1. Consider the following MIPS code fragment: **(15)**

```
        .data
# stored in A are squares of first 7 primes
A:      .word  4, 9, 25, 49, 121, 169, 289
        .text
# insert MIPS instructions here.
```

For insertion at the comment, write a *single* MIPS program that will do the following:

(a) Put the starting address of `A` into register `$s1`.

(b) Inside a loop, access each element of `A` and add these values, leaving the result in register `$s2`. [You must use a loop for this.]

(c) Print the resulting sum, using `syscall`. [Recall that `syscall` requires `$v0` equal to 1 to print the value in `$a0`.]

Your MIPS code should do what is asked for above and *nothing more*.

---

2. Write portions of MIPS assembler language that will call a function **`F`** with parameters **`25`** and **`144`**. Then your code should store the returned value in a variable **`i`** corresponding to register **`$s3`**. You should give the complete code for **`F`**, which should add its two arguments and return the sum. Your code should not invoke any syscall, and it should not have any of the rest of the main function except the call to **`F`** and the storing of the returned value into **`$s3`**. The code for **`F`** should be complete. *For full credit, you must follow the MIPS conventions for passing parameters and for returning a value from a function.* (You do not need to explicitly save any registers on the stack.) Thus you should be implementing the following C/Java code: **(15)**

```
/* in main */
i = F(25, 144);  /* use $s3 for i */
. . .

int F(int a, int b) {
    return a + b;
}
```

(25)

3. For this problem you will be including an additional instruction into the *single*cycle datapath described in the text, one not included in the examples in the text. The new instruction is **`addi`** (add immediate).

   You are to use a xerox of Figure 5.29 (the final datapath for the singlecycle implementation of MIPS). No new datapaths will be needed, but you will need control signals different from the instructions you have studied. The Control unit will be expanded to include an input of 8, the opcode for **`addi`**. *You must show the control signals as they should be set.*

   Use a highlighter to show the *data* lines traversed during execution. You should also write in the values of all signals. You should *not* highlight the control lines.

   Specifically, suppose the instruction is **`addi $t3, $t2, 45`**, which in machine language form is:

```
214b002d                            (in hexadecimal)
001000 01010 01011 0000000000101101 (fields in binary)
     8    10    11               45 (fields in decimal)
```

   In marking the values traveling along data lines, assume that **`$t2 = $10`** holds the value **`51`** and that **`$t3`** is **`$11`**. (Note that the ALUOp control can be two 0 bits, so that the input to the ALU from ALU control will then be 010, that is, an add.)

---

(25)

4. For this problem, you are to use one or more xeroxes of Figure 5.33 (final datapath for the *multi*-cycle implementation of MIPS). You will be tracing through the path of the **`lw`** instruction on this multi-cycle model. You should use several colors of highlighters to trace the paths the instruction and associated data takes through the diagram. (Or you can trace these paths using more than one diagram.) Do *not* show data traveling to "dead-end" components, which will eventually have no effect. In particular, do not show the computation of the branch address, which will not be used in this case.

   For this diagram, do *not* give the values of control signals.

   Below the diagram, or in some other way, carefully identify *which cycle* (or step) of handling the instruction belongs to each part of the highlighted datapath (just for data, not control). Thus you should identify *Cycle 1*, *Cycle 2*, *Cycle 3*, and perhaps *Cycle 4* and *Cycle 5* (if the instruction uses Cycles 4 and 5).

   Use the following specific instruction:

```
lw $t5,92($t1)
```

   or in machine language form:

```
0x8d2d005c                          (in hexadecimal)
100011 01001 01101 00000 00001 011100 (fields in binary)
    35     9    13                92 (fields in decimal)
```

   Start at the left side, showing the PC coming in, and assume this instruction is read from the Instruction memory. *Be sure to identify the different cycles.* Don't forget the PC. Show what values are traveling along the different lines, assuming the following initial values:

(a) **`$t1`** and **`$t5`** are registers numbers **`9`** and **`13`** (decimal), respectively.

(b) The contents of register **`9`** is **`200`** (decimal).

(c) The PC has value **`16`**.

---

5. Consider the xerox from your text (top part of Figure 6.42, page 487), showing one stage of pipelined execution where forwarding is required. The instructions are shown with each stage. **(20)**

   Please use a pen and/or marker to show answers directly on the xerox, identified by page 487. There are *two* instances of forwarding here.

   (a) For each instance identify exactly what register is being forwarded, what stage it is forwarded *from* and what stage it is forwarded *to*. (The pipeline stages are: IF, ID, EX, MEM, and WB.)

   (b) Use a letter **A** to mark *all* the lines having to do with forwarding for the first instance. (Be careful, there are several lines involved.)

   (c) Use a letter **B** to mark *all* the lines having to do with forwarding for the second instance. (Be careful, there are several lines involved.)

---

6. Consider the xerox of the lower part of Figure 6.47 from your text, showing pipelined execution, where a *stall* is needed. The instructions in execution are shown above each pipeline stage. **(20)**

   Use a pen and/or marker to show some answers directly on Figure 6.47. (Some answers are with the question.)

   (a) Which instruction makes a stall necessary?

   (b) Which unit detects that this is an instruction that may need a stall?

   (c) What information goes into this unit that indicates that a stall is needed? This information goes in on three lines. Carefully mark with a letter **A** these three lines and show what values are on the lines.

   (d) Three lines actually allow the stall to take place. Carefully mark these three lines with a letter **B** and show what values are on the lines.

   (e) Say briefly how the lines and values in (d) cause the stall to occur.

   (f) Is the actual forwarding done during one of these two diagrams? (Yes or No). If "No", say when the forwarding will be carried out.

(15)

7. Consider the following code, which is the simplified trap handler from the take-home quiz:

```
17          .kdata
18  Duhh:   .asciiz "\nDuhh-hhhhh!\n"
19          .ktext 0x80000080
20          li $v0 4        # syscall 4 (print_str)
21          la $a0 Duhh     # print "Duhh-hhhhh!"
22          syscall
23          mfc0 $k0 $14
24          rfe
25          addiu $k0 $k0 4
26          jr $k0
27          .text
28          .globl __start
29  __start: jal main
30          li $v0 10
31          syscall
```

   (a) Under what circumstances do you start executing at line 29? What do lines 29-31 do? (What are they for?)

   (b) Under what circumstances do you start executing at line 20?

   (c) What do lines 20-22 do?

   (d) What is the significance of line 25? (What is it for?)

   (e) What does line 26 do?

---

(15)

8. Consider the xerox of Figure 7.7 from the text, showing the diagram for cache memory.

   (a) What is the size of the field used for an *index* (that is, how many bits give the location of the entry in the cache)? From this, say how many words of data this cache holds.

   (b) Why does this hardware not pay attention to the low order 2 bits?

   (c) Why is the *Tag* field in the cache 20 bits wide?

   (d) Consider the process that occurs with a cache *hit*. How does the hardware know there is a hit? (Be careful, there are *two* conditions.) How does the hardware manage to get to the correct data without some kind of search?

   (e) Describe what the text says happens on an instruction cache *miss*. (Four steps.)

---

(10)

9. This problems deals with *buses*.

   (a) What is a *bus* is a computer system?

   (b) What is the difference between a *synchronous* bus and an *asynchronous* bus?

   (c) Give one advantage and one disadvantage of each type of bus compared to the other.