CS 2734, Computer Organization II
Spring Semester, 2002
Final Examination

1. Consider the following MIPS code fragment: **(20)**

```
        .data
A:      .space 40
        .text
# insert MIPS instructions here.
```

For insertion at the comment, write MIPS instructions that will do the following:

(a) Create a loop of 10 iterations that will let the register `$t0` take on values 0, 1, ..., 9.

(b) Store the values of `$t0` into successive words of the array `A`, so that if we printed the 10 locations of `A`, we would print out the numbers 0, 1, ... 9.

(You should not include code to do this printing. 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 parameter **`12`**. Then your code should store the returned value in a variable **`i`** corresponding to register **`$s3`**. You should give the complete code for **`F`**, which should add its argument to itself and return the result. Your code should not invoke any syscall, and it should not have any of the rest of the main function except the call to **`F`** and the storing of the returned value into **`$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 code: **(20)**

```
/* in main */
i = F(12);  /* use $s3 for i */
. . .

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

---

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). **(30)**

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 different control signals. The Control unit will be expanded to include an input of 8, the opcode for **`addi`**. 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.

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`**. (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.)

---

4. **(30)** 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. **(20)** Consider the xerox of Figure 6.42 from your text, showing pipelined execution where forwarding is required. There are two diagrams, an upper one and a lower one, showing successive cycles.

   The pipelined MIPS machine is in the middle of executing the following sequence of instructions:

```
sub     $2, $1, $3
and     $4, $2, $5
or      $4, $4, $2
add     $9, $4, $2
```

Please use a pen and/or marker to show answers directly on Figure 6.42.

(a) The above instructions require *four* instances of data forwarding. Mark the instructions above with circles and arrows to show exactly which registers are being forwarded to which instructions.

(b) In the upper diagram, use a **B** to mark *all* the lines having to do with forwarding from the **and** instruction.

(c) In the upper diagram, use a **C** to mark *all* the lines having to do with forwarding from the **sub** instruction.

(d) In the lower diagram, use a **D** to mark *all* the lines having to do with the forwarding step that is occurring, either from the **and** or the **or** instruction.

(e) In the lower diagram, explain which register is begin forwarded, from which instruction. (The lines used for control and data should have been marked with a **D**.)

---

6. Consider the xerox of Figure 6.47 from your text, showing pipelined execution, where a *stall* is needed. The instructions in execution are: **(20)**

```
lw      $2, 20($1)
and     $4, $2, $5
or      $4, $4, $2
add     $9, $4, $2
```

Please use a pen and/or marker to show answers directly on Figure 6.47. You must decide whether your answers should be placed in the upper diagram or in the lower diagram.

(a) Mark the instruction that makes a stall necessary with an **A**.

(b) Mark the unit that detects that this is an instruction that may need a stall with a **B**. Also mark with a **B** the line that supplies this unit with the information.

(c) Mark the two lines that supply the duplicate register numbers showing that a stall is needed with **C**s.

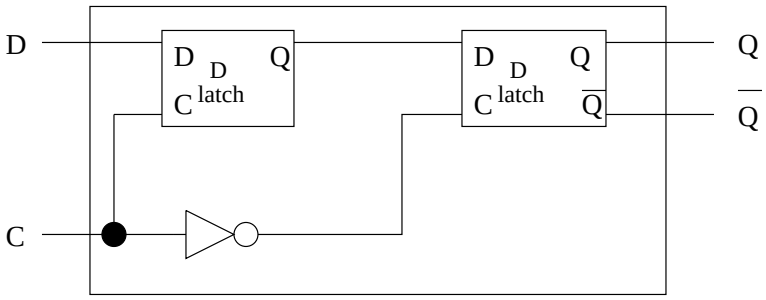
(d) Mark the line(s) used to stall pipeline stage 1 using a **D**. Show the signal(s) that is(are) on these line(s)?

(e) Mark the line(s) used to stall pipeline stage 2 using an **E**. Show the signal(s) that is(are) on these line(s).

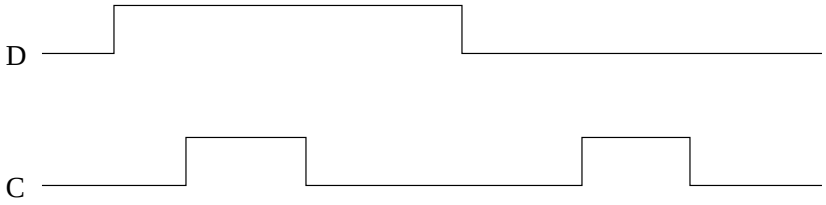
(f) Is the actual forwarding done during one of these two diagrams? (Yes or No). If “No”, say when the forwarding will be carried out.

(15)

7. Consider the following diagram of a D flip-flop:



D flip-flop



Q ——— . . .

This D flip-flop is constructed from two D latches. Recall that a D latch lets the signal D go through if the clock C is asserted (the D latch is *open*), and the output does not change if the clock C is deasserted (the D latch is *closed*). Such a D latch is said to be *transparent.*

(a) Give the output Q of the D flip-flop as it would appear in the figure above. (That is, fill in the output signal Q.)

(b) Explain very carefully why the output is what it is. (Your explanation should refer to the specifics of the diagram above, including the NOT gate and the two D latches.)

(15)

8. This problem deals with the *Hamming Code.*

(a) What error detection and correction is the full binary Hamming code capable of?

(b) Suppose you have the following six data bits: **`0 1 1 0 0 1`**. Insert the proper values for the check bits in the correct positions. (Do not use position **`0`**.)

(c) Suppose the bit in position **`6`** is transmitted in error (not the sixth *data* bit). Show how the Hamming code will be able to correct this error.

---

(10)

9. This problems deals with *buses.*

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

(b) What are the differences between a *synchronous* bus and an *asynchronous* bus? What are the advantages/disadvantages of one over the other, in different curcumstances?