CS 2734, Computer Organization II
Fall Semester, 2004
Second Examination

1. For this problem, you are to use the separate xerox diagram (final datapath for the *single*-cycle implementation of MIPS). You will be tracing through the path of the **`lw`** instruction on this single-cycle model. You should use a highlighter to trace the path the instruction and associated data takes through the diagram. (Do *not* show data traveling to "dead-end" components, which will eventually have no effect.) Then write in the values for the *relevant* control signals. (Do *not* give control signals that serve to keep "dead-end" paths from having an effect.) **(35)**

   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. Don't forget the handling of the PC by this instruction. 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 contents of memory at location **`292`** is **`144`** (decimal).

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

   (Don't forget to handle the PC as well as the rest of the instruction.)

---

2. For this problem, you are to use a xerox of the final datapath for the *multi*-cycle implementation of MIPS. You will be tracing through the path of the **`beq`** instruction on this multi-cycle model. You should use highlighters (preferably in several colors) to trace the paths the instruction and associated data takes through the diagram. Do *not* show data traveling to "dead-end" components, which will eventually have no effect. **(35)**

   For this diagram, you do *not* need to 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).

   You may use a single diagram for the whole instruction (preferably using different colors for the different cycles), or you may use a diagram for each cycle, or you may use more than one diagram but with several cycles on a diagram. There is no reward for art work, but your diagrams must be readable. Use the following specific instruction:

   ```
   beq   $t2, $t5, LabelA
   ```

or in machine language form:

```
0x114d0004                              (in hexadecimal)
000100 01010 01101 00000 00000 000100 (fields in binary)
     4    10    13                  4 (fields in decimal)
```

Start at the left side, showing the PC value coming in, and assume this instruction is read from the Instruction Memory. Show what values are traveling along the different lines, assuming the following initial values:

(a) **`$t2`** and **`$t5`** are register numbers **`10`** and **`13`** (decimal), respectively.

(b) Assume that the contents of each of these registers is **`5234`**, so you should assume that the branch is taken.

(c) Assume the PC has value **`20`** (decimal) initially. On the proper line, give the *final* PC value, assuming the branch is taken. Don't forget to highlight the parts related to the PC as well as the rest of the instruction. *Be sure to identify the different cycles.*

---

3. This question is concerned with *exceptions* and *interrupts,* as described in the text for the multi-cycle implementation. **(30)**

Consider the case of an interrupt because of an *integer overflow*. Here the ALU detects the problem and signals the control component. During the fourth cycle (when the overflow occurs), the diagram goes into a new state number 11, which represents the fact that an overflow has occured. This state sets the following control signals:

```
IntCause = 1
CauseWrite = 1
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01 (which tells the ALU to subtract)
EPCWrite = 1
PCWrite = 1
PCSource = 11
```

Assume the value of the *incremented* program counter is **`16`**. Use these components to decide what the hardware does. Referring to the xerox of Figure 5.39 from your text (the multi-cycle datapath with additions for exceptions), trace through the signals, and decide on their effects.

(a) What value is stored into the EPC component? Using a highlighter in one color clearly show all lines having to do with getting the correct value into the EPC, and give the values on these lines, *including control lines*. (You could instead clearly label the lines with the letter "a".)

(b) What value is stored into the Cause register? Using a highlighter in a different color clearly show all lines having to do with getting the correct value into the Cause register, and give the values on these lines, *including control lines*. (You could instead clearly label the lines with the letter "b".)

(c) What value is stored into the PC component? Using a highlighter in a different color clearly show all lines having to do with getting the correct value into the PC, and give the values on these lines, *including control lines*. (You could instead clearly label the lines with the letter "c".)