CS 2733/2731, Computer Organization II
Fall Semester, 2004
Final Examination

1. Write a segment of a MIPS assembly language program that implements a simple loop to initialize an array. Specifically, your segment should do the following, and nothing else: **(15)**

   (a) Allocate an array **A** of 10 32-bit integer locations.

   (b) Load the address of **A** into register **$s1**.

   (c) Use a *loop* to store the integer 47 into each of the 10 locations of **A**. (There are several ways to do this with a loop, but you *may not* use the **.word** directive to get this result without a loop.)

---

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: **(15)**

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

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

---

3. For this problem, you are to use a xerox of the final datapath for the *single*-cycle implementation of MIPS. You will be tracing through the path of the **addi** (add immediate) 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.) **(25)**

   No new datapaths will be needed. 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.)

---

4. For this problem, you are to use one or more xeroxes the 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. **(25)**

   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 following instance of the *Hamming Code*. **(15)**

Position	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Bit value	1	1	0	1	1	0	1	0	1	0	0	1

   (a) Which of the above bits are check bits?

   (b) Verify that the rightmost check bit above has the correct value.

   (c) Suppose the bit in position **`5`** is transmitted in error, that is, transmitted as a 1 instead of a 0. Show how the Hamming code will be able to correct this error.

   (d) Using the original bits above, determine the value of the check bit in position **`0`**. What is this bit used for?

6. Consider the floating point number (a `double`) with representation: **(15)**

```
1011 1111 1101 1100  (48 more 0's)  (binary)
b    f    d    c     (12 more 0's)  (hex)
```

(a) What is the sign of this number?

(b) What is its exponent (power of 2)? (Remember that the bias for a `double` is 1023, and that an exponent of 1 is represented by `100 0000 0000`.)

(c) What is the significant part?

(d) Put i, ii, and iii together to get the number.

---

7. Consider the following three MIPS instructions in the pipelined implementation: **(25)**

```
add   $6, $2, $5
or    $6, $3, $4
and   $8, $6, $4
```

Suppose just these three instructions are executing. It will take 7 pipelined cycles for these instructions to pass through, but only one of these cycles involves *forwarding*. You are to draw only that cycle on the diagram of the pipelined implementation with the forwarding unit.

(a) On the diagram, write in the three instructions in the proper stages where they will be when forwarding occurs.

(b) Identify exactly which register's value 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.)

(c) Carefully highlight the line that is carrying the actual 32-bit quantity that is being forwarded. Label this line clearly with the letter **A**.

(d) Identify the other forwarding line that would be used to forward a 32-bit register value if it were not for the forwarding that is occurring as described and highlighted in part (c) above. Carefully highlight this line (that could have been used to forward a 32-bit quantity, but is not in fact being used here). Label this line clearly with the letter **B**.

(e) There are various lines into the Forwarding Unit that carry 5-bit register numbers. Carefully highlight only those 5-bit lines that are essential for the forwarding step in this cycle, and no other lines. Label these lines clearly with the letter **C**.

(f) Finally, there is a control line that is making the correct forwarding occur. Carefully highlight this line. Label it clearly with the letter **D**.

(25)

8. Consider the following three MIPS instructions in the pipelined implementation:

```
lw    $2, 20($1)
and   $4, $2, $5
or    $7, $6, $2
```

Suppose just these three instructions are executing, and consider the cycle of execution when the **lw** instruction is in the EX stage, so that the **and** is in the ID stage and the **or** is in the IF stage. At this point a *stall* is needed. You are to draw only that cycle on the diagram of the pipelined implementation with the forwarding unit and the hazard detection unit.

(a) Which instruction makes a stall necessary?

(b) Which unit detects that this is an instruction that will 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 highlight these lines, mark them with a letter **A**, and show what values are on the lines.

(d) Three lines actually allow the stall to take place. Carefully highlight these lines, mark them 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 *forwarding* done during the cycle discussed here? (Yes or No). If "No", say what instances of forwarding will be needed, and give the cycle or cycles when the forwarding will be carried out.

---

(15)

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

```
17          .kdata
18  Inside: .asciiz "  Inside local exception handler\n"
19          .ktext 0x80000180
20          li $v0 4        # syscall 4 (print_str)
21          la $a0 Inside
22          syscall
23
24          mfc0 $k0 $14
25          addiu $k0 $k0 4
26          mtc0 $k0 $14
27          eret
28  # Standard startup code.  Invoke the routine main with no arguments.
29          .text
30          .globl __start
31  __start: jal main        # start up main
32          li $v0 10
33          syscall         # syscall 10 (exit)
```

(a) Under what circumstances do you start executing at line 31? What do lines 31-33 do? (What are they for?)

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

(c) What do lines 20-22 do? (What are they for?)

(d) What do lines 24-26 do? (What are they for?)

(e) What is the significance of line 27? (What is it for?)