CS 2733/2731, Computer Organization II
Fall 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 **`$t2`** take on values 2, 4, 6, 8, 10, 12, 14, 16, 18, 20.

(b) Store the values of **`$t2`** into successive words of the array **`A`**, so that if we printed the 10 locations of **`A`**, we would print out the numbers 2, 4, ..., 20.

(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 a *single* MIPS assembler language program that will call a function **`PrintSum`** with parameters **`55`** and **`89`**. The function **`PrintSum`** should add its parameters and print the result. It should then call a function **`PrintNewLine`** whose code is given below. You should give the complete code for the function **`PrintSum`** and for the call to **`PrintSum`**. Because **`PrintSum`** calls another function, it needs to save the return address at the start and restore the return address at the end. Thus you should be implementing the following C code: **(20)**

```
void PrintSum(int a, int b) {
    int c = a + b;
    printf("%i", c);
    PrintNewLine();
}
void PrintNewline(void) {
    printf("\n");
}
. . .
/* in main */
int x = 55, y = 89;
Sum(x, y);
. . .
```

Assume that the function **`PrintNewline`** is implemented with:

```
PrintNewline:
      la     $a0,   Newl
      addi   $v0,  $0, 4
      syscall
      jr     $ra
     .data
Newl:  .asciiz "\n"
```

(30)

3. For this problem, you are to use a xerox of Figure 5.29 (final datapath for the *single*cycle implementation of MIPS). You will be tracing through the path of the **`sw`** instruction on this single-cycle model. You should use a highlighter in one color 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 without using a higlighter (you may use another color of highlighter if you wish), show the *relevant* control signals. (Do *not* show control signals that serve to keep "dead-end" paths from having an effect. The input to ALU control is 00, and from that component, the input to the ALU is 010(add).)

   Use the following specific instruction: **`sw $t1, 12($t0)`**, or in machine language form:

```
0xad2a000c                          (in hexadecimal)
101011 01000 01001 0000000000001100 (fields in binary)
    43     8     9               12 (fields in decimal)
```

   Start at the left side, showing the PC 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) **`$t0`** and **`$t1`** are registers **`8`** and **`9`** (decimal), respectively.

   (b) The contents of register **`8`** is **`64`** (decimal), and of register **`9`** is **`144`** (decimal).

   (c) The PC has value **`32`**.

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

---

(30)

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

   You are to use one or more xeroxes of Figure 5.33 (the final datapath for the multicycle implementation of MIPS). You should decide how many cycles to use (three, four, or five). No additional datapaths are necessary, but you need to determine the control signals for each cycle.

   Use a highlighter to show the data lines traversed during each cycle, labeling the data line with the cycle number. *You must show the relevant control line values needed for each cycle.*

   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 on data lines, assume that **`$t2 = $10`** holds the value **`51`**.

(20)

5. Consider the xerox of part of Figure 6.41 from your text, showing pipelined execution where forwarding is required. The diagram shows successive pipeline stages during one clock cycle.

   The pipelined MIPS machine is in the middle of executing the sequence of instructions shown at the top of the diagram.

   Consider the 6 inputs to the forwarding unit: label the 4 on the right from top to bottom "A", "B", "C", "D". Label the two inputs on the left from top to bottom "E", "F".

   (a) Explain what each of these 6 inputs is used for.

   (b) Which ones are essential for the particular forwarding step in this figure? What facts are true about these inputs that lead to forwarding?

   (c) Use a highlighter to show the exact line that does the forwarding in this specific case.

---

(20)

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

```
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 simplified trap handler supplied as part of the take-home quiz:

```
        .kdata
Duhh:   .asciiz "\nDuhh-hhhhh!\n"
        .ktext 0x80000080
        li $v0 4        # syscall 4 (print_str)
        la $a0 Duhh     # print "Duhh-hhhhh!"
        syscall
        mfc0 $k0 $14
```

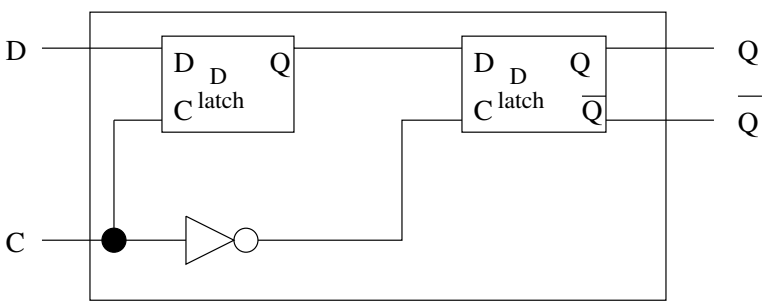
```
24          rfe
25          addiu $k0 $k0 4
26          jr $k0
27          .text
28          .globl __start
29  __start: jal main
30          li $v0 10
31          syscall
```

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

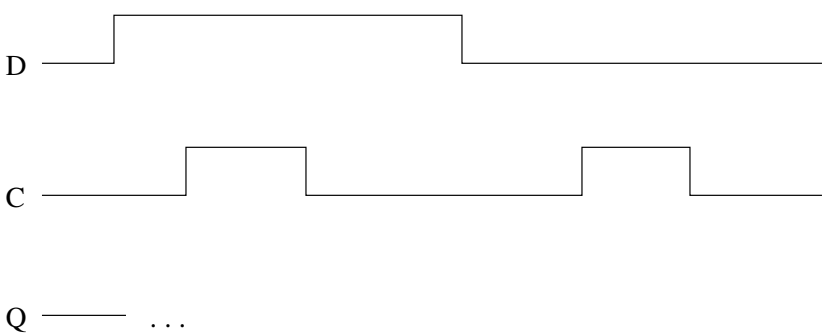
(b) Under what circumstances do you start executing at line 17? What do lines 17-26 do? What are they for?

---

8. Consider the following diagram of a D flip-flop: **(15)**



D flip-flop



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.)