CS 2734, Computer Organization II
Fall Semester, 2001
First Examination

1. Below are questions about number representations and conversions: **(20)**

   (a) Convert the (decimal) number -58 to 16-bit two's complement binary. (The binary representation for 58 is 111010.)

   (b) Consider the floating point number (a `double`) with binary representations:

   ```
   0100 0000 0011 0100  (48 more 0's)  (binary)
   ```

   i. What is the sign of this number?

   ii. 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`.)

   iii. What is the significant part?

   iv. Put i, ii, and iii together to get the number.

---

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

```
        .data
A:      .word 2, 3, 5, 7, 11, 13, 17, 19, 23, 29
        .text
# insert MIPS instructions here.
```

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

   (a) Store the address of `A` into `$s0`.

   (b) Use a loop to add the 10 values of the array `A`.
   (You must access the array `A` and you must use a loop for credit. Your MIPS code should do what is asked for above and *nothing more*.)

---

3. Consider the same MIPS code fragment in the previous question, that defines an array `A` of 10 integers. **(20)**

   Write a MIPS function `Avals` so that

   (a) `Avals` is passed the starting address of `A` as its parameter.

   (b) `Avalues` saves the register `$ra` on the stack.

   (c) `Avals` adds the 0th and 1st array elements, and returns this number. (`Avals` should return `A[0] + A[1]`.)

(d) `Avals` restores the register `$ra` saved above and returns.

(e) Separately show a call to `Avals` with input parameter the starting address of `A`.

Note: You should just give code for the call to `Avals` and for the definition of the function `Avals` that do the above items and *nothing more*. You should follow MIPS parameter convensions.

---

4. In the short segment of assembler code below, indicate the machine code that would be generated for the three given instructions. You can simply give the decimal value of the bits in each of the fields of the instruction, so you do not need to convert to hexadecimal. For the `bne` instruction, assume that the program counter is incremented at the beginning of the instruction. (Note that register `$t0` is 8 and register `$t1` is 9.) **(20)**

```
        addi   $t0, $0, 10
Loop:   add    $t1, $t1, $t0
        bne    $t0, $t1, Loop
```

---

5. Consider the following truth table with inputs A, B, and C, and output D. **(20)**

Inputs			Output
A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

(a) Write the output D using a Boolean algebra equation (that is, involving operators $+$, $\cdot$, and the bar above for negation.

(b) Give a single combinational logic circuit that will produce the desired output D from the given inputs A, B, and C. (This is a circuit involving AND, OR, and NOT gates.)