Test #1

Name: _____Key_____

CS 341l, test #1. 100 points total, number of points each question is worth is indicated in parentheses. Answer all questions. Be as concise as possible while still answering the question. **You have 50 minutes exactly, I'll collect the tests at 1:50pm.** Closed book, closed notes, closed calculator, closed everything.

#1. (12 points, 2 points each) Circle T for true or F for false.

*A carry-lookahead adder is typically faster than a ripple carry adder because the carry lookahead adder doesn't need to wait for the carry bits to propagate through every bit in the word that is being added.

T

* "Assembly" and "microcode" mean the same thing. F

*Non-volatile memory retains data even in the absence of a power source. T

*The `jal Label` instruction differs from the `j Label` instruction in that it saves the return pointer on the stack to support subroutine calls.

(saved to $ra, not the stack) F

*An instruction placed in the branch delay slot is only executed if the branch is not taken.

F

*Von Neumann et al.'s original conception of what computing machinery should look like didn't include any ideas of registers or stored programs being stored in the same memory as data, so in terms of performance and usability Von Neumann machines were largely a failure and are no longer very common.

F

#2 (10 points, for each of the following, circle one of R, I, J-type or pseudo-instruction based on how it would be encoded, 1 point each).

`bne $t0,$t1,Label`	R-type	**I-type**	J-type	Pseudo-instruction
`j Label`	R-type	I-type	**J-type**	Pseudo-instruction
`jal Label`	R-type	I-type	**J-type**	Pseudo-instruction
`addi $t0,$t0,55`	R-type	**I-type**	J-type	Pseudo-instruction
`slt $t5,$t6,$t7`	**R-type**	I-type	J-type	Pseudo-instruction
`slti $t5,$t6,-77`	R-type	**I-type**	J-type	Pseudo-instruction
`la $s0,Label`	R-type	I-type	J-type	**Pseudoinstruction**
`lw $t0,16($s0)`	R-type	**I-type**	J-type	Pseudo-instruction
`lw $t0,0($s0)`	R-type	**I-type**	J-type	Pseudo-instruction
`sll $t0,$t1,5`	**R-type**	I-type	J-type	Pseudo-instruction

#3. (10 points, 2 points each match)

```
0       int w, x, y, z, A[100];
10      int i = 10 * 5;
20      for (loop = 0; loop < 100; loop++)
30      {
40          x = i + 5;
50          y = i * loop;
60          z = loop * 32;
70          A[w] = A[w] + x + y + z;
80          if (i > 100) i -= 10;
90      }
```

In the above C code with line numbers, match the compiler optimization with the line number to which it would most appropriately be applied, by writing the corresponding letter. Loop unrolling is already done for you as an example. There is a one-to-one correspondence between a-f and the line numbers to the right, so choose the most

appropriate mapping even if the optimization seems to be possible in other places.

a. Code motion	Line 60	_f_
b. Dead code elimination	Line 10	_e_
c. Common subexpression elimination	Lines 20 through 90	_d_
d. Loop unrolling	Line 40	_a_
e. Constant folding (or constant propagation)	Line 80	_b_
f. Strength reduction	Line 70	_c_

#4. (10 points) Draw lines to divide the following MIPS code into basic blocks, and then indicate below the total number of basic blocks. One point for each division line you draw in a correct place and the rest of the points for getting the total right. Note that this is one block of MIPS code though it may span multiple pages on this test, *i.e.*, ignore the page break, unless it also happens to be a basic block break.

```
ascii_loop:                          # Ascii to integer loop
        li    $t4, 10                # Load 10 into temporary register
        addi  $t2, $0, 0             # t4. Init t2 to 0

        move  $t0, $a0               # move contents of a0 into t0
                                     # a0 is the address in memory that
                                     # contains the characters we
                                      # read in

        jne   $s3, $zero, skipcheck
-------------------------------------------------------------------------
loop:   lb    $t1, ($t0)             # Load first byte into t1
        li    $t3, "\n"              # Load a end of line character \n
                                      # into t3

        beq   $t1, $t3, ascii_end    # Check if the character read is
-------------------------------------------------------------------------
                                     # to the end of the loop.
        addi  $s3, $zero, 0
-------------------------------------------------------------------------
```

```
skipcheck:
        li    $t3, "0"                  # Load the integer value of the
                                        # ASCII character 0 into register
                                         # t3

        blt   $t1, $t3, ascii_error     # Check if value in t1 is less
                                        # then the value in t3.  If it is
                                        # then something is wrong, go to
                                         # error routine
-------------------------------------------------------------------------
        li    $t3, "9"                  # Load the integer value of the
                                        # ASCII character 9 into register
                                         # t3

        bgt   $t1, $t3, ascii_error     # check if value in t1 is greater
                                        # then the value in t3.  If it is
                                         # then something is wrong, go to
                                              # error routine
-------------------------------------------------------------------------
        addi  $t3, $t1, -48             # Convert from ascii to integer
        mul   $t2, $t2, $t4             # Multiply to put integer in
                                         # right place
        add   $t2, $t3, $t2             # Adding total together
        addi  $t0, $t0, 1               # Increment to next byte in memory
        j     loop                      # Jump back to loop to work on
                                         # next char
```

How many basic blocks are there total? _____6_____

#5. (10 points) Using the following equation for Ahmdahl's law, and showing your work, answer the following question.

O.S. = 1/((1 – f)+(f/s))

Divide instructions are 20% of the instruction mix for a benchmark that you are

interested in for testing your CPU design. If you decide to improve the speed of divides in your ALU, what, theoretically, is the highest overall speedup you could possibly achieve for that particular benchmark?

Important to show your work, pluf in f = 0.2, s = inf, and solve for O.S.

#6. (10 points) What will the output of the following C code be (assuming I had all the proper headers and it compiled okay) on a machine with a typical floating point implementation (such as a MIPS or a Pentium)?

```
float f = 0.0;
int i;
for (i = 0; i < 10; i++)
{
    f = f + 0.1;
}
if (f == 1.0)
    printf("Insert your ATM card.");
else
    printf("Feed me a stray kitten.\n");
```

What will the output be?: _____Feed me a stray kitten._____

#7. (5 points) Multiple choice, circle only the one correct answer. What happens when I run the following code on a 32-bit machine with a typical C compiler?

```
unsigned int i, j = 0x80000000, k=0xb0000000;
i = j + k;
printf("If I've gotten this far, i is %d\n", i);
```

a. The program prints output with a number for `i` and then exits normally.

b. The program crashes because of an overflow exception.

c. The program prints `i` as `NaN`.

#8. (5 points) Multiple choice, circle only the one correct answer. What happens when I run this code on a 32-bit machine with a typical C compiler?

```
signed int i, j = 0x80000000, k=0xb0000000;
i = j + k;
printf("If I've gotten this far, i is %d\n", i);
```

a. **The program prints output with a number for `i` and then exits normally.**

b. The program crashes because of an overflow exception.

c. The program prints `i` as `NaN`.

#9. (6 points, 1 point each.) Write the name of the addressing mode to which each constraint applies.

Limited to whatever 256MB block of memory the PC currently is in, not relative to the PC or any register:

_____Psuedo-direct addressing_____

Limited to 32 registers: _____Register addressing_____

Limited to a immediates between -2^15 and (2^15-1): ___Immediate addressing_____

Limited to branching to within about a 256 KB range relative to the current program counter plus 4:

_PC-relative addressing_____

Limited to loading or storing a word, halfword or byte that is + or – about a 2^15 offset from an address that is contained in a register:

_____Base (or displacement) addressing___

#10. (8 points, 2 points each) Circle one of "stack," "heap," or ".data section" based on where each type of memory allocation would be stored.

Activation records	**stack**	heap	.data section
malloc()/free()ed memory	stack	**heap**	.data section
Global variables	stack	heap	**.data section**
Functions' local variables	**stack**	heap	.data section

#11. (4 points) Multiple choice, circle only the one correct answer. Someone tells you that since their RISC machine has a lower CPI and a higher clock rate than your CISC machine, it is much faster than your CISC machine. What is the main fallacy of their argument?

a. IPC is a better indicator of performance than CPI.

b. **The CISC machine can get the same work done with fewer instructions.**

c. RISCs typically don't have higher clock rates than CISCs.

d. RISC designs are often microcoded, while CISC designs are not.

#12. (3 points) Since `j` and `jal` instructions cannot reach the entire 32-bit address space, how can a MIPS programmer jump to an arbitrary 32-bit address?

Put address in a register use jr instruction

#13. (3 points) Other than rs, rt, rd, and funct, what is the fifth feild of an R-format instruction and what is it used for?

opcode used for saying which instruction, or shamt used for shifting in sll, for example

#15. (1 point) Of the two authors of the book (David Patterson and John Hennessy), which one is known for developing the MIPS architecture?

Hennessy

#16. (1 point) Which two MIPS registers are not restored on return from exceptions/interrupts?

$k0 and $k1

#17. (1 point) What popular embedded microprocessor, released by Intel in 1980, has among its oddities that you can address registers with memory addresses in addition to their register numbers?

The 8051

#18. (1 point) Give an example of an x86 instruction with base+scaled index addressing.

mov eax,[ebx+ecx<<4]