Spring 2012 CS61C Midterm

Your Name: RUBRIC

Your TA: Rimas Scott Alan Eric Paul Ian

Login: cs61c-___

This exam is worth 100 points, or about 10% of your total course grade.

The exam contains 8 questions.

This booklet contains 12 numbered pages including the cover page. Put all answers on these pages, please; don't hand in stray pieces of paper.

You will receive 1 point for properly filling out your name, TA, and login (login must be filled in properly on every page of the exam). This is the exam's 0th question.

Question	Points(Minutes)	Score
0	1	
1	6(3)	
2	7(5)	
3	10(4)	
4	12(5)	
5	21(13)	
6	8(4)	
7	20(14)	
8	15(9)	
Total	100(56)	

1. True or False (1pt each)

a. T / (F): A mapper can output at most one key value pair per key value pair given as input.

b. (T) / F : The shuffle phase cannot group key value pairs in such a way as to send (a,7) and (a,4) to separate instances of reduce in one round of map-reduce.

c. (T) / F : Assuming there is more than one worker, a map-reduce job can successfully complete even if one of the workers fails.

d. (T) / F : Both the map and reduce tasks can run on multiple machines to exploit parallelism.

e. T / (F): SIMD is an example of data level parallelism that performs different operations on a single input data value to produce multiple output data values.

f. (T) / F : The amount of parallelism achieved by a single SSE instrinsic is limited by the width of registers.

2. Potpourri

(1 pt) a) Which of the following is NOT a consequence of Moore's Law?

A) The same computer will get cheaper over time
B) Transistors get smaller over time
C) Transistors will become twice as energy-efficient every 18 months
D) Computer chips can get physically smaller over time

b) Moore's law states that the transistors (1 pt) per chip will double (1 pt) every 18 or 24 (1 pt) months.

(1 pt) c) Both the ARM and MIPS architectures were announced about 27 years ago. According to Moore's Law, approximately how many more times transistors per chip do we have today than when these RISC instruction sets were developed?

A) 64 times more transistors today
B) 1,000 times more transistors today
if 24 mo C) 16,000 times more transistors today
if 18 mo D) 256,000 times more transistors today

(2 pts) d) What is the formula that the assembler should use to calculate the value to place in the address field of a beq or bne machine language instruction? Assume DestAddress is the address of the destination if the branch is taken and PC points to the beq instruciton.
For example,

Address	Label	Instruction
10000		beq $t1,$t2, DestAddress
…		…
10080	DestAddress:	addu $t1, $$t2, $t3

A) DestAddress
B) DestAddress / 4
C) DestAddress * 4
D) DestAddress – PC
E) (DestAddress – PC)/4
F) DestAddress – PC – 4
G) (DestAddress – PC – 4) / 4
H) (DestAddress – PC – 4) * 4
I) None of the above

3. Memory Hierarchy

10 POINTS, 1 PER QUESTION

Short answer:

1) Caches are designed to exploit TEMPORAL and SPATIAL locality in memory access patterns.
[GET 1/2 POINT IF ONE CORRECT]

2) The _____ level of the memory hierarchy has the highest cost per bit of storage.
(closest/furthest from chip)

3) The _____ level of the memory hierarchy has the highest capacity.
(closest/furthest from chip)

4) In a 16kB byte-addressed, direct mapped cache that uses 32-bit addresses and 64 byte cache lines, the address is divided into 6 offset bits, 8 index bits and 18 tag bits.
[GET 1/2 POINT IF ONE CORRECT]

5) Holding capacitance and power constant, a CPU's voltage must be decreased by a factor of $\sqrt{2}$ to allow for a doubling of frequency.

Multiple choice:

For each of the following questions, select the option which is most often true:

A) increases
B) decreases
C) does not effect

6) Increasing the capacity of a cache A its access time

7) Decreasing the capacity of a cache A its miss rate

8) Decreasing the capacity of a cache C its miss penalty

9) Increasing a cache's hit rate B the effective average memory access time

10) Assuming a constant capacity, increasing a cache's block size (cache line size)
B OR C the amount of tag storage required
[DEPENDS ON IF TALKING ABOUT TAG AREA ⇒ B
OR IF " " TAG WIDTH ⇒ C]

4. Compiling Linking Interpreting etc.

a) In one word each, name the most common producer and consumer of the following items:

Choose from: linker loader compiler assembler programmer

(item)	This is the output of:	This is the input to:
bne $t0,$s0,done	COMPILER	ASSEMBLER
char *s = "hello world"	PROGRAMMER	COMPILER
app.o string.o	ASSEMBLER	LINKER
firefox	LINKER	LOADER

↑1 POINT PER CORRECT ANSWER PER 6 BOXES ↑

b) Circle ***all*** the advantages of interpretation over compilation:

- Interpreted programs often have a higher CPI, resulting in better resource use.
- Interpreted programs do not require compilation.
- An interpreted interpreter can interpret itself, eliminating any need for compilers.
- An interpreted program can run on many different platforms without modification.

c) Circle ***all*** the reasons why many programs are compiled instead of interpreted:

- Compilers produce programs that often execute faster.
- It is easier to write code when using a compiler than when using an interpreter.
- Interpreted programs cannot be scaled to many machines.
- It's harder to reverse engineer a compiled program than an interpreted one.

+1 FOR CORRECT ANSWER
−1 FOR INCORRECT ANSWER
MIN IS 0, MAX IS 2

5. C Question - Graph Representation

```
// vertex_t declared elsewhere, sizeof(vertex_t) == 16
// struct holding two pointers to vertex_t's
typedef struct edge {
   vertex_t *first;
   vertex_t *second;
} edge_t;
```

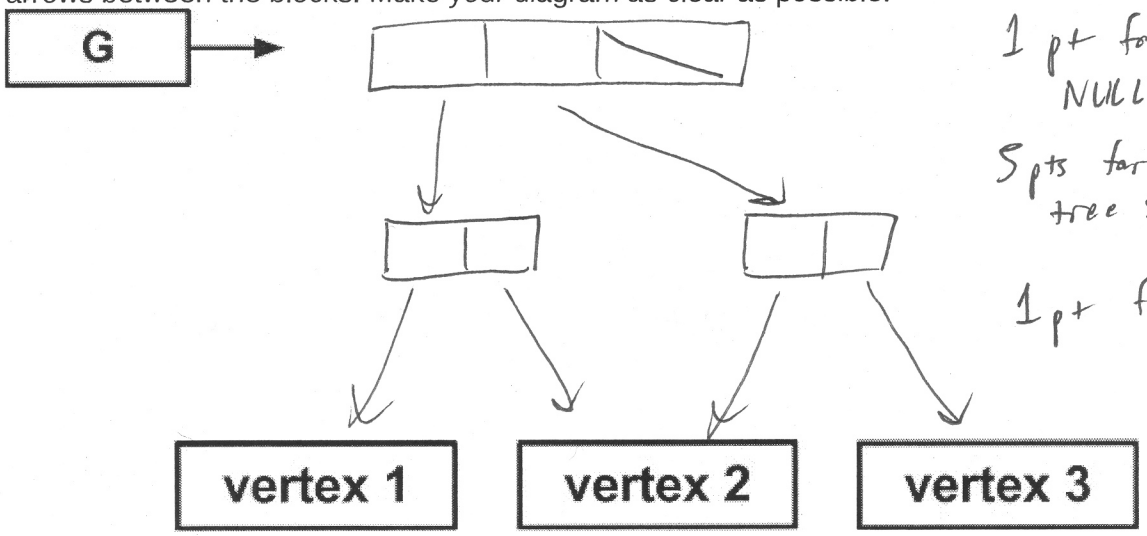
```
// A graph is represented as an array of edge_t pointers terminated by a null pointer
(similar to how a c string is terminated by a null character)
// The following pointer to a graph is declared:
edge_t** G;
```

a) Determine the value of the following expressions, assuming a 32-bit architecture and tightly-packed structs:

sizeof(&G) = 4
sizeof(G[0]) = 4
sizeof(**G) = 8

3 pts for exact answers
- partial credit if units given

b) Draw a possible pointer diagram starting from G, assuming the graph it points to has exactly two edges and three vertices. G and the three vertices are given - your job is to fill in the rest. In a pointer diagram blocks of contiguous memory are drawn together, and pointers are drawn as arrows between the blocks. Make your diagram as clear as possible.



1 pt for correct NULL termination

5 pts for 2-layer tree structure

1 pt for referencing vertices

c) How much space does it take to store the graph you just drew (including G, the pointer to the graph)? Show your work.

2 pts for correct computation (partial credit if very close)
1 pt for correct units given for all elements in summation

d) Given print_vertex(vertex_t *v), fill in the blanks in this function:

For example, an output for a graph with three edges could be:

(a,b)
(b,c)
(c,a)

```
void print_graph(edge_t **graph) {
  while (__*graph__) {
    printf("(");
    print_vertex(__(*graph)->first__);
    printf(",");
    print_vertex(__(*graph)->second__);
    printf(")\n");
    __graph++__
  }
}
```

2 pts for pointer increment
2 pts for correct conditional statement
3 pts for print arg (all or nothing)
1 pt for "second" print arg

penalties for introducing extra variables

6. SIMD String

Below is some MIPS from lecture for string copying. $s1 holds the address of string we are copying, and $s2 holds the address we are copying to. We'll define a new kind of string to allow faster copying through SIMD. Our new kind of string is much like the regular C strings except:

- The string starts on a word-aligned address; in other words, the last two bits of its 32 bit address are 0s.
- The string ends with a word-aligned 32 bit null value.

Using the idea of SIMD, change at most four instructions to make a faster C string copy. To modify an instruction fill in the replacement instruction in the blank on the same line as the original instruction. Do not fill in a blank if you do not wish to modify its corresponding instruction.

Loop:

lb $t2,0($s1)	# $t2 = *p	lw $t2, 0($s1) or lh
sb $t2,0($s2)	# *q = $t2	sw $t2, 0($s2) or sh
addi $s1,$s1,1	# p = p + 1	addi $s1 $s1 4
addi $s2,$s2,1	# q = q + 1	addi $s2 $s2 4
beq $t2,$zero,Exit	# if *p == 0, go to Exit	_____
j Loop	# go to Loop	_____

2 pts a line

"correct formula" implies more than simply copying from a sheet. Understanding of the formula must be demonstrated

Login: cs61c-____

7. How Fast

Suppose you are running code on a machine with a two level data cache. It also has an instruction cache, which always hits, so we disregard it for our calculations.

- L1$ has a local hit rate of 50%, and hits in 1 cycle.
- L2$ has a local hit rate of 95%, and hits in 10 cycles.
- Main memory has a hit rate of 100%, and hits in 100 cycles.

Now, consider the following MIPS function:

```
        jal foo             #begin function call
        .
        .
        .
foo:
        addiu $sp, $sp, -8
        sw $s0, 0($sp)
        sw $s1, 4($sp)
        lw $s0, 0($a0)
        lw $s1, 0($a1)
        sw $s1, 0($a0)
        sw $s0, 0($a1)
        lw $s0, 0($sp)
        lw $s1, 4($sp)
        addiu $sp, $sp, 8
        jr $ra              #end function call
```

1pt

a. Briefly describe what foo does.

Swaps the values pointed to by $a0, $a1. Does not swap $a0, $a1.

2pts - correct formula
1pt - correct numerical substitution
1pt - correct answer/units

b. Calculate the AMAT for the architecture provided (*Reminder:* we are ignoring instruction loads for our calculations).

$AMAT = 1 + 0.5(10 + .05(100)) = 8.5$ cycles

c. Assuming that the program's CPI would be 1 in the absence of cache misses, calculate the CPI of a call to foo (*note that the function call includes the jal*).

$CPI = 1 + (\frac{2}{3})(0.5(10 + .05(100))) = 6$ cycles/instruction

2pts - Correct formula

1pt - correct substitution

1pt - answer and units

-2 pts per incorrect line
minimum of 0/8

d. The following function is supposed to do the same thing as foo, but doesn't. Without adding any instructions , repair footoo so that it does. To modify an instruction fill in the replacement instruction in the blank on the same line as the original instruction. Do not fill in a blank if you do not wish to modify its corresponding instruction.

```
        jal footoo      #begin function call    _____________
        .
        .
        .
footoo:
        lw $s0,0($a0)                           lw $t0, 0($a0)
        lw $s1,0($a1)                           lw $t1, 0($a1)
        sw $s1,0($a0)                           sw $t1, 0($a0)
        sw $s0,0($a1)                           sw $t0, 0($a1)
        jr $ra          #end function call      _____________
```

e. Assuming that a program spends 20% of it's time making calls to foo, what is the speedup of switching to footoo?

The ratio of ~~loads to~~ loads and stores to other instructions is constant moving from foo to footoo, so $CPI_{foo} = CPI_{footoo}$.

So we can apply Amdahl's law directly:

$$\text{speedup} = \frac{1}{(1-0.2) + \frac{0.2}{2}} = \frac{10}{9} = 1.\overline{1}$$

1 pt - correct formula
1 pt - correct numerical substitution
1 pt - correct answer

8. Ackermann

The Ackermann function A is defined as follows:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

Fill in the following C function so that it computes A(m, n).

```
unsigned int A(unsigned int m, unsigned int n) {
  if (          m == 0          ) {
      return n+1;
  } else if (     n == 0          ) {
      return A(m-1, 1);
  } else {
      return A(m-1, A(m,n-1));
  }
}
```

1 Point per line, -1 per type of mistake
Common mistakes: miss word return, use = instead of ==, use n++ or post-increment operator elsewhere

Now you're going to translate that C into an equivalent MIPS function. We've built a skeleton once again, but you're going to have to fill in the blanks to flesh it out.

A:

```
a) addiu $sp, $sp, -12      <- not +12
   sw $s0, 0($sp)
   sw $s1, 4($sp)
   sw $ra, 8($sp)

   addu $s0, $a0, $0
   addu $s1, $a1, $0

   beq $s0, $0, L1
b) beq $s1, $0, L2
      or $a1
```

#continued on next page#

```
addu $a0, $s0, $0 #does nothing, included for clarity
addiu $a1, $s1, -1
jal A
```

c) addu $a1, $v0, $0

d) addiu $a0, $s0, -1

```
jal A
j Exit
```

L1:

e, f) addiu $v0, $s1 or $a1, 1

```
j Exit
```

L2:

```
addiu $a0, $s0 ,-1
addiu $a1,$0, 1
jal A
j Exit
```

Exit:

g) lw $s0, 0 ($sp)

h) lw $s1, 4 ($sp)

i) lw $ra, 8 ($sp)

j) addiu $sp, $sp, 12

```
jr $ra
```

1 point for b, e, f, a+j (must get both)

2 points for c, d, g+h+i (if reversed -1)