Spring 2012 CS61C Final

Your Name: SOLUTION

Your TA: Rimas Scott Alan Eric Paul Ian

Login: cs61c-___

This exam is worth 110 points, or about 20% 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 5 points 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	MIN	MAX	Points(Minutes)	AVG	STD DEV Score
0	4	5	5(0)	5	0.1
1	2	15	15(10)	7.7	3.0
2	0	10	10(10)	6.0	2.3
3	0	9	10(9)	5.3	1.8
4	0	14	14(11)	8.2	4.0
5	0	16	16(15)	9.5	4.1
6	0	14	14(11)	7.6	3.7
7	0	10	10(9)	6.6	3.2
8	0	16	16(15)	7.7	3.5
Total	11	100	110(90)	63.5	16.6

Login: cs61c-____

Potpourri: Mostly True or False. (ALAN C)

The different parts of this question are independent.

a. Simplify the following boolean expression.

1pt: promising start
2pts: minor error
3pts: correct answer

$$\overline{(\bar{a}+b)(\bar{b}+c)(\bar{c}+d)}+(\bar{a}+d)$$

$$\equiv$$

$$((a \Rightarrow b)(b \Rightarrow c)(c \Rightarrow d)) \Rightarrow (a \Rightarrow d)$$

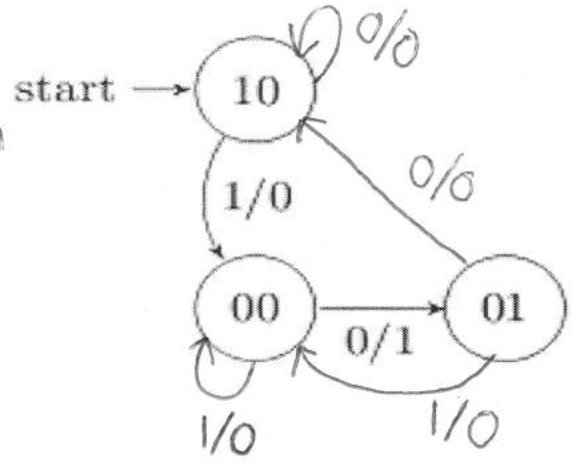
$$\equiv$$

$$1$$

Full credit was also given to $\bar{a}+d$, as many students misread the question as $(\bar{a}+b)(\bar{b}+c)(\bar{c}+d)+\bar{a}+d$

b. Suppose we feed in digits of a number to a finite state machine most significant bit first (e.g. it would see 1 first if we input 1000). Complete the below diagram so that it outputs 1 exactly when the number is divisible by 2, but not by 4. Assume that the machine's starting state is such that it has seen more than four 0's.

1 pt. per transition



1 point was taken for using an extra state

c. How many unique gates with n input bits and m output bits are there ?

3pts: correct answer

$$\#\text{ unique } f: \mathbb{F}_{2^n} \to \mathbb{F}_{2^m} = (2^m)^{2^n}$$

1pt: mix up m, n

d. Circle all modifications to the IEEE754 float standard that would change the proportion of floats with values between -1 and 1.

1 pt. for circling each

- (changing the exponent bias value)
- increasing the number of exponent bits
- (removing the implicit leading one)
- getting rid of denorm values

e. A program tries to load a word at address X that causes a TLB miss but not a page fault or protection violations.

True or False: A TLB miss means that the page table does not contain a valid mapping for virtual page corresponding to the address X

True or False: There is no need to look up the page table because there is no page fault

True or False: The word that the program is trying to load is present in physical memory.

Don't MIPS the Point! (PAUL)

```
typedef struct {
        int val;
        struct node* next;
} node;

/* Removes the first node after cur with a value of x and return a pointer to it. You may
assume cur is not null.*/
node* removeNext(node *cur, int x) {
        node* next = cur->next;
        if (next) {  //have a next node?
                if (next->val == x) {  //remove the next if its val is x
                        cur->next = next->next;
                } else {  //otherwise, keep searching for node to remove
                        return removeNext(next, x);
                }
        }
        return next;
}
```

a) Use the above definition of a linked list node. Fill in the blanks below so that the assembly code does the equivalent of the C `removeNext`. Assume no delayed branches.

```
0  removeNext:
1      addiu $sp $sp -4
2      sw $ra 0($sp)
3      lw $v0 4($a0)          #v0 = next
4      beq $v0 $0 done        #done if next is null
5      lw $t1 0($v0)
6      bne $a1 $t1 recurse    #recurse if not equal to x
7      lw $t2 4($v0)
8      sw $t2 4($a0)          #cur->next = next->next
9  done:
10     lw $ra 0($sp)
11     addiu $sp $sp 4
12     jr $ra                 #return
13 recurse:
14     move $a0 $v0           #set arguments for recursive call
15     jal removeNext         #recurse
16     j done
```

1pt per line
-2pt if lines 3,7,8 correct except for a consistently wrong offset (ie. 1pt for all 3)

b) Say we change line 15's `jal` to `j`. Which lines, if any, can we remove so that `removeNext` still works properly without editing any other line? List the line numbers. 1, 2, 10, 11, 16

c) Say we want a function remove which does the same as removeNext, except the first node is also a target for removal, and the return type is void. Fill in the prototype below so that it could work properly.

```
void remove(node** cur, int x);
```

1pt 1pt

2pts for all 5, 1pt if correct lines minus incorrect lines is at least 2

On Multiplying Polynomials (OMP) (DAVE)

```
void multiply1(double *A, double *B,
               double *C, int n) {
  // Outer Loop
  for (int i=0; i < n; i++)
    // Inner Loop
    for (int j=0; j < n; j++)
      C[i+j] += A[i] * B[j];
}
```

```
void multiply2(double *A, double *B,
               double *C, int n) {
  // Outer Loop
  for (int i=0; i < 2*n; i++)
    // Inner Loop
    for (int j = MAX(i - n + 1, 0);
         j < MIN(i + 1, n); j++)
      C[i] += A[j] * B[i-j];
}
```

a) Suppose OpenMP pragmas are placed on the outer and inner loops according to the table below. Evaluate if multiply1 and multiply2 are guaranteed to return the correct answer with the given pragma placements. Circle your answer in each of the rightmost six cells. A, B, and C point to non-overlapping memory regions.

1 POINT PER CORRECT ANSWER

Outer Loop	Inner Loop	multiply1		multiply2	
<leave empty>	parallel for	*correct*	*incorrect*	~~*correct*~~	*incorrect*
parallel for	<leave empty>	*correct*	*incorrect*	*correct*	~~*incorrect*~~
parallel for	parallel for	*correct*	*incorrect*	*correct*	*incorrect*

b) Suppose that we execute the multiply2 in the following configuration. We have parallelized only the outer loop, our machine has 64B cache lines, is running 8 threads, and n = 1024. In scheduling scheme A we allocate indices to threads round robin (ie, thread *p* takes care of indices *p*, *p* + 8, *p* + 16, etc). In B we allocate work in contiguous chunks. Which configurations exhibit false sharing?

2 POINTS FOR CORRECT ANSWER

- A and B
- A only
- B only
- Neither A nor B

c) Circle all statements that are **true** for both multiply1 and multiply2 unless otherwise noted.

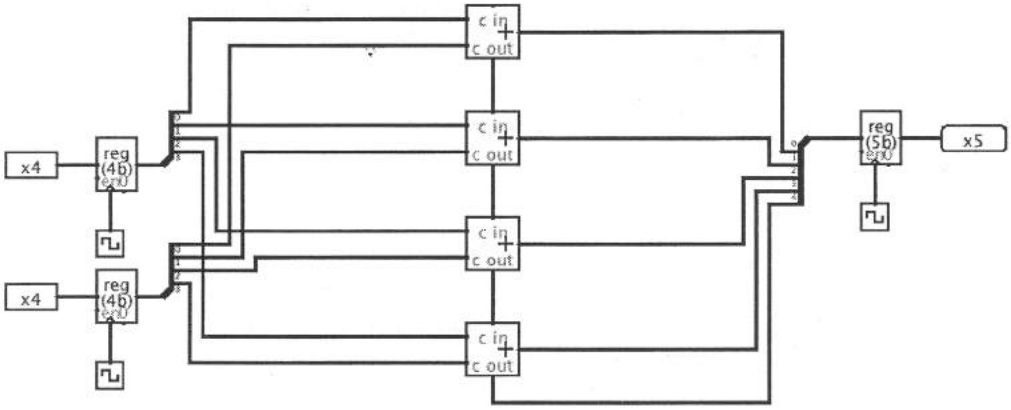
- Threads can see different "i" values even if the parallel for is only on the inner loop.
- A #pragma omp barrier on the innermost statement will fix all data races from part (a).
- Ignoring correctness, parallelizing both inner & outer loops will give a speedup for large n.
- The synchronization overhead of parallelizing an inner loop with a given number of threads is amortized for large n.

2 POINTS FOR CORRECT ANSWER
−1 POINT PER INCORRECT ANSWER
MINIMUM SCORE IS 0

14 points Total

Additional Logic (IAN)

For this problem we will look at a simple carry bit adder. Each adder block is simply a one bit adder. This circuit will add a series of pairs of 4 bit inputs and output 5 bit outputs (instead of having an overflow bit). The inputs will change once each clock cycle.



2 points

a) We want to pick the maximum clock frequency such that we can guarantee this circuit works correctly. The adder block has a delay of time *a*. The registers have a clock-to-q delay of time *q*, as well as a setup time *s*, and a hold time *h* where $h \leq q$. All times given are in seconds.

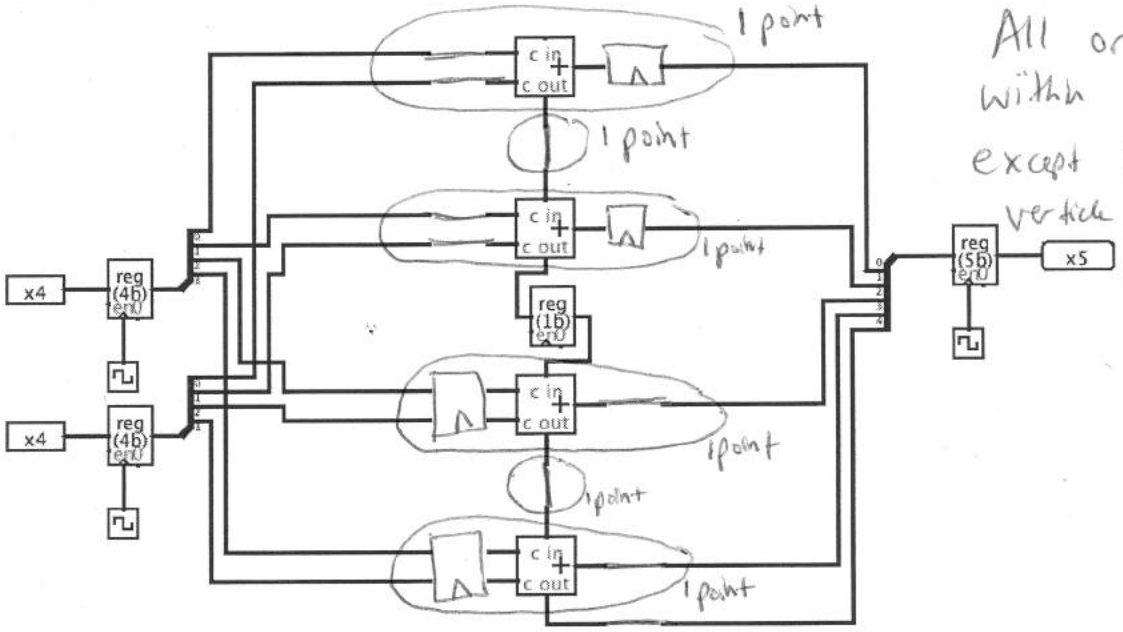
Max frequency = 1/[$4a + q + s$] Hz

1 for $4a$
1 for $q+s$

6 points

b) Alyssa P Hacker realizes that she can improve this circuit by pipelining it. She starts by inserting a register between the carry out of the second adder and the carry in of the third adder (see next page).

Your job is to add additional registers to make this circuit correctly perform addition again. You are to either place a register in each empty space (any box will be taken to be a register) or place a wire across it.



All or nothing with circle, except if both vertical left blank
-1 not -2

2 points c) Now what is the max frequency:

Max frequency = 1/[2a+2+5] Hz

- 2 points for correct answer
- 1 point for dividing 4a by something other than 2
- 1 point for dividing the whole expression not just 4a by 2

2 points d) In terms of time, the latency of a single addition has increased / decreased / stayed the same. Circle one.

All or nothing

2 points In terms of additions per time, the throughput of this circuit has increased / decreased / stayed the same. Circle one.

All or nothing

Hazardous Question (DAVE)

Consider each of the following sets of instructions. Decide the minimum number of stalls needed in each case with and without forwarding. Assume the five stage MIPS pipeline discussed in class and that equality for branches are checked at the Decode stage. Assume delayed branching.

Number of stalls required?

	Without forwarding	With forwarding

`addiu $t1 $t0 5`
`addu $t4 $t3 $t1`
`addu $t0 $t3 $t0`
	2	0

`lw $t0 4($s0)`
`addu $t1 $t2 $t3`
`beq $t0 $t1 DONE`
`addu $t5 $t5 $5`
	2	1

`lw $t1 0($s0)` `sw $t1 0($s1)`	2	0
`addu $t0 $s0 $t1` `lw $t1 8($t0)`	2	0

- 2 POINTS PER CORRECT ANSWER
- 4 POINTS IF ALL 4 "WITHOUT FORWARDING" CHOICES WERE "3 STALLS" (CONFUSION ABOUT WRITEBACK STAGE & DECODE IN SAME CLOCK CYCLE)
- ~~2 POINTS IF ALL 4 "WITHOUT FORWARDING" CHOICES WERE~~

Counting Cache (SCOTT)

(6) **A)** A *word-addressed* cache has 8 words of capacity, 2 word blocks, and uses a LRU replacement policy. Assume it started empty and then the warming sequence was executed. For each cache organization given, determine whether each access will cause a hit or a miss. Indicate the result with a "H" or "M" in each box.

Warming sequence: 1, 4, 7, 3

	Query Sequence	5	4	10	2	0
(2)	Direct Mapped	H	H	M	M	H
(2)	2-way Set Associative	H	H	M	H	H
(2)	Fully Associative	H	H	M	H	M

(2) **B)** How many tag (T), index (I), and offset (O) bits does a *byte addressed* cache have that is 8MB, 4-way set associative, 32B block, and has a 48b address?
T : I : O = 27 : 16 : 5

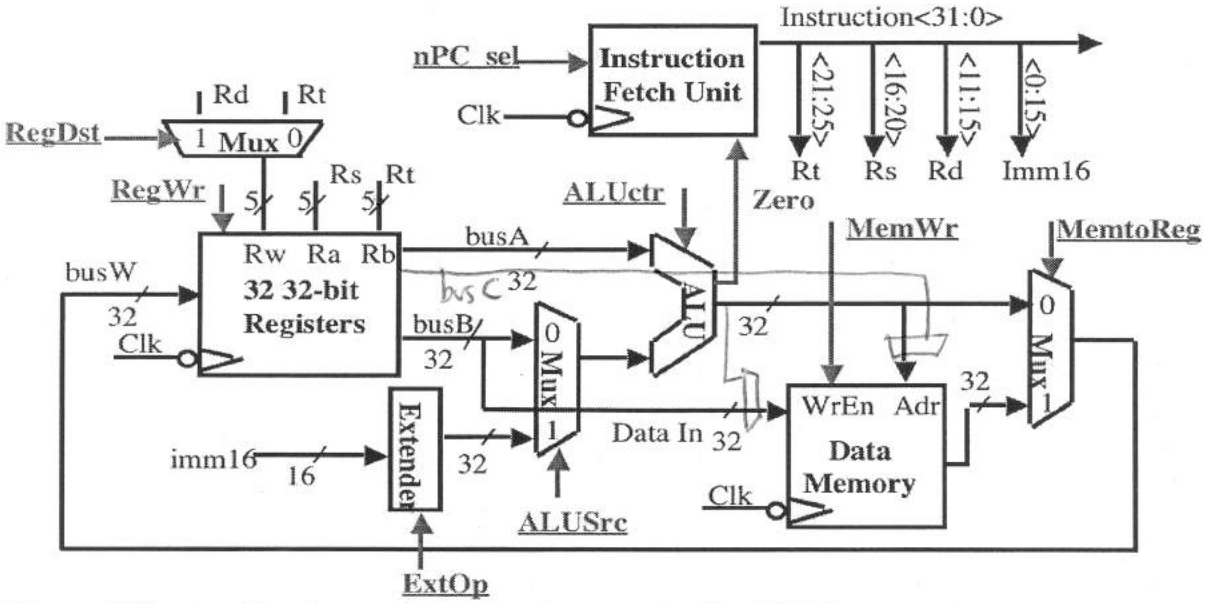
2/2 all correct
1/2 two wrong, but adds to 48
0/2 2 wrong

(6) **C)** For each of the following modifications to a set associative cache, indicate how it will affect the different miss rates. Notice that some parameters are held constant. Use (+) for increase, (=) for stays the same, and (-) for decrease.

	Miss Type	Compulsory	Conflict	Capacity
(2)	Double associativity (capacity and block size constant)	=	-	=
(2)	Halve block size (associativity & # of sets constant)	+	= - (alternate)	+
(2)	Double # of sets (block size & capacity constant)	=	+	=

For A & C, 2pts per row
1 pt off per mistake (no negative)
⇒ for A, 5/5 = 2pts 4/5 = 1pt 0-3/5 = 0pts
⇒ for C, 3/3 = 2pts 2/3 = 1pt 0-1/3 = 0pts

Additional Instruction (RIMAS)



We would like to add an instruction to the above **unpipelined** CPU.
The new instruction is addm rd, rs, rt. This instruction is like a normal add but instead of writing the result of adding the contents of the rs and rt registers to a register we write it to a location in memory specified by rd (Note, there is no offset, it is simply written to the address held in rd).

M[rd] <- R[rs] + R[rt]

a) Change as *little as possible* in the datapath above **(draw your changes right in the figure)** to enable `addm` and list all changes below. Your modification may use muxes, wires, constants, additional read or write ports on the register file, and additional ALU's (again, use as little as you need!). You may not need all boxes, and do not need to mention the addition of the control signal **addm,** which is 1 iff the executing instruction is `addm`.

7 pts

(i)	mux adr, bus C
(ii)	mux data in, ALU out
(iii)	add reg out port for Rd
(iv)	

-2 pts
-2 pts
-3 pts
0 pts if no drawing

b) We now want to set all the control lines appropriately. List what each signal should be: an intuitive name or {0, 1, x – don't care}. Include a new control signal **addm.**

3 pts

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg	addm
	0	PC+4	X	0	add	1	X	1

1 pt deducted for every ~2 wrong

Bit by Bit (ERIC)

In a hamming code, with log(n) extra bits, you can detect and correct any single bit error in a n-bit string. The code requires computing log(n) parity groups p1, p2, p4, p8, etc. where the parity of each group is zero if the entire codeword is correct. If there is an error you can use the non-zero parities to find out which bit is corrupted.

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Code bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity Group	p1	X		X		X		X		X		X		X		X		X		X		...
	p2		X	X			X	X			X	X			X	X			X	X		
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X	X	X						
	p16																X	X	X	X	X	

a) Finish the implementation of fix_single_error() below. Implement by putting a simple bitwise expression or value in each blank.

You are given a single helper function hamming_parity() that will compute and return the parity of the hamming groups $p1$, $p2$, $p4$, $p8$ etc. in a 32-bit word, where $p1 \in \{0,1\}$ is stored in the least significant bit, $p2$ in the second bit, $p4$ in the third, etc. The input is a pointer to the 'code' **byte array** of length ⌈n/8⌉, which contains a hamming code n bits long. The least significant bit of our hamming data is the least significant bit of code[0], is also $p1$.

```
// HINT: xor'ing (^'ing) a bit with a 1 flips the target bit.
void fix_single_error(uint8_t *code, size_t n) {
  uint32_t parity_vector = hamming_parity(code, n);

  if (parity_vector != ____0____) {   2 pts

      parity_vector -= 1; // adjust for zero-indexing

    code[ parity_vector >> 3 ] ^= 1 << (parity_vector & 0x7)
  }
}
```

1 pt for parity vector on left

1 pt for "1" on right

2 pts each for correct bitwise exprs

b) Answer the questions about the following (terribly slow) implementation of hamming_parity(). Assume that the #defined macros are implemented correctly for you using standard C expressions which should look similar to your solution to part (a).

```
1   // ^'s the kth bit in vector with the bit b (0 or 1)
2   #define XOR_THIS_BIT(vector, k, b)
3   // returns the nth bit counting from ptr (0 or 1)
4   #define GET_BIT(ptr, n)
5   // returns 1 if the ith bit is counted in the jth hamming group, else 0
6   #define IS_HAMMING_PARITY_BIT(i, j)
7
8   uint32_t hamming_parity(uint8_t *code, size_t n) {
9      uint32_t parity_vector = 0x00000000;
10     size_t logn = 1 + log2(n);
11     for (size_t i=0; i < n; i++) {
12        #pragma omp parallel for
13        for (size_t j=0; j < logn; j++)
14           if (IS_HAMMING_PARITY_BIT(i, j))
15              XOR_THIS_BIT(parity_vector, j, GET_BIT(code, i));
16     }
17     return parity_vector;
18  }
```

i) If you unrolled the loop on line 13, this would help with ___control___ hazards. 2 pts

ii) You can eliminate the if statement on line 14-15 in favor of a bitwise ___&___ operator. 2 pts

iii) Propose a single line deletion and single line insertion that will greatly speed up hamming_parity(). Your solution must not cause any data races or other incorrect behavior.
Delete which line? ___12___
Insert after line ___10___ this: ___#pragma omp parallel for reduction (^:parity_vector)___

4 pts
3 pts for parallel for in right place
1 pt for recognizing you need reduction

other answers that produced some speedup received 1-3 pts