University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2005 | Instructor: Dan Garcia | 2005-12-17

Last Name	
First Name	
Student ID Number	
Login	cs61c-
Login First Letter (please circle)	a b c d e f g h i j k l m
Login Second Letter (please circle)	a b c d e f g h i j k l m n o p q r s t u v w x y z
*The name of your **LAB** TA (please circle)*	Jeremy Michael Navtej Zhangxi
Name of the person to your Left	
Name of the person to your Right	
*All the work is my own. I have no prior knowledge of the exam contents nor will I share the contents with others in CS61C who have not taken it yet. **(please sign)***	

Instructions (Read Me!)

- This booklet contains 9 numbered pages including the cover page. Put all answers on these pages (feel free to use the back of any page for scratch work); don't hand in any stray pieces of paper.
- Please **turn off** all pagers, cell phones & beepers. Remove all hats & headphones. Place your backpacks, laptops and jackets at the front. Sit in *every other* seat. Nothing may be placed in the "no fly zone" spare seat/desk between students.
- Fill in the front of this page and put your name & login on every sheet of paper.
- You have 180 minutes to complete this exam. The exam is closed book, no computers, PDAs or calculators. You may use two pages (US Letter, front and back) of notes, plus the green reference sheet from COD 3/e.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided. "IEC format" refers to the mebi, tebi, etc prefixes. You have 3 hours...relax.
- **You must complete ALL THE QUESTIONS, regardless of your score on the midterm.** Clobbering only works from the Final to the Midterm, not vice versa.

Problem	**M1**	**M2**	**M3**	**Ms**
Minutes	20	20	20	60
Points	**10**	**10**	**10**	**30**
Score				

F1	**F2**	**F3**	**F4**	**F5**	**Fs**
24	24	24	24	24	120
18	**18**	**18**	**18**	**18**	**90**

Total
180
120

Name: ______________________________ Login: `cs61c-`____

Midterm Revisited

M1) "Doctor, our patient is *encoding*!" (10 pts, 20 min)

a) A Binary Coded Decimal (BCD) uses a dedicated nibble for each decimal digit, so a byte could represent all the numbers from 00-99. We will use our standard MIPS 32-bit word to encode a BCD. What is the ratio (to one significant figure, in decimal) of overall bit patterns to the ones that encode a valid BCD? (E.g., With a *single* decimal digit, it'd be 16/10 ≈ 2.) Show your work. Your answer should not be an expression, it should be a decimal number rounded to 1 significant figure.

____

b) Suppose we have a very small 4 pixel × 8 pixel grayscale video display where each pixel can independently be set to one of 4 shades of gray. How many unique images can possibly be displayed? Leave your answer in IEC form (e.g., 64 kibi images, 8 mebi images, etc).

____

c) If we were to try to compare two `float`s using our MIPS signed integer compare `slt`, when would we get an *incorrect* answer (i.e., describe in English the set of *all possible inputs* that generate incorrect answers)? Assume neither encodes a `NaN` or `±0`.

____

d) Put the corresponding letters for each 32-bit value in order from least to greatest. Hint: the question isn't asking you to write down what each one is, it only asks for the relative order!

A. `0xF0000000` (IEEE float)
B. `0xF0000000` (2's complement)
C. `0xF0000000` (sign-magnitude)
D. `0xFFFFFFFF` (2's complement)
E. `0xFFFFFFFF` (1's complement)
F. `0xF1000000` (IEEE float)
G. `0x70000000` (IEEE float)
H. `0x7FFFFFFF` (2's complement)
I. `0x80000010` (IEEE float)

Least ____ Greatest

Name: ______________________________ Login: cs61c-____

M2) "Those are some big numbers you got there…" (10 pts, 20 min)

A bignum is a data structure designed to represent large integers. It does so by abstractly considering all of the bits in the num array as part of one very large integer. This code is run on a standard 32-bit MIPS machine, where a word (defined below) is 32 bits wide and a halfword is 16 bits wide.

```
typedef unsigned int    word;
typedef unsigned short halfword;
typedef struct bignum_struct {
   int length;    // number of words
   word *num;     // the actual data
} bignum;
```

This function shows how bignums are used:

```
void print_bignum(bignum *b) {
   printf("0x"); // Print hex prefix
   for (int i = b->length-1; i>=0; i--)
      printf("%08x", b->num[i]);
}
```

a) Is the ordering of words in the num array BIG or LITTLE endian? (circle one)

b) How many bytes would be used in the *static*, *stack* and *heap* areas as the result of lines 1, 3 and 4 below? **Treat each line independently!** E.g., For line 3, don't count the space allocated in line 1.

	static	*stack*	*heap*
Line 1			
Line 3			
Line 4			

```
1 bignum biggie;
2 int main(int argc, char *argv[]) {
3   bignum bigTriple[3], *bigArray[4];
4   bigArray[1] = (bignum *) malloc (sizeof(bignum) * 2);
```

b) Complete the add function for two bignums, which you may assume **are the same** length. Our C compiler translates z = x + y (where x,y,z are words) to add (not addu, as is customary) and thus could generate a hardware (HW) overflow we don't want, as we're running on untrusted HW. Your code should be written so that words never overflow in HW (so we do all adding in the halfword).

```
void add(bignum *a, bignum *b, bignum *sum, word carry_in, word *carry_out) {

   // reserve space for num array. Remember a and b are the SAME length...

   sum->num =

   for (int i=0; i < a->length; i++) {  // word-by-word do addition of lo, hi halfwords

      // add lo halfwords of a,b

      word lo      =

      // add hi halfwords of a,b (but in the safe, low halfword area so no HW overflow)

      word hi      =

      // combine low and hi halfwords (put back in their places), like a lui-ori

      sum->num[i] = (hi << 16) | (halfword) lo;

      // what's the carry_in for the next word?

      carry_in     =
   }
   sum->length = a->length;
   *carry_out  = carry_in;
}
```

Name: ______________________________ Login: cs61c-____

M3) "Fenry Hord invented the *disassembly* line…" (10 pts, 20 min)

a) Given the MIPS code below, write the equivalent C function below in the structure we've provided. Feel free to add comments to help your disassembly. To aid readability, you must use the variable names from our comments below in your C solution where appropriate.

```
foo:
addiu $sp, $sp, -12
sw    $a0, 0($sp)        # src
sw    $a1, 4($sp)        # size
sw    $ra, 8($sp)
move  $a0, $a1           #
addiu $a0, $a0, 1        #
jal   malloc             #
move  $t0, $v0           # dest
lw    $t1, 0($sp)        # src
lw    $t2, 4($sp)
addu  $t2, $t2, $t1      # end

foo_loop:
beq   $t2, $t1, foo_end #
lbu   $t4, 0($t1)        #
ori   $t4, $t4, 0x20     #
sb    $t4, 0($t0)        #
addiu $t0, $t0, 1        #
addiu $t1, $t1, 1        #
j     foo_loop

foo_end:
sb    $0, 0($t1)         #
lw    $ra, 8($sp)        #
addiu $sp, $sp, 12
jr    $ra
```

```
________ foo (________ src, ________ size) {



  for(__________; ___________; ___________) {



  }

}
```

b) If `src` contained *letters*, what is a more appropriate name for the subroutine `foo`? (i.e., what would "`jal foo`" do, from the point of view of the caller?) Hint: you might find the green sheet handy here.

c) What if we called `foo` from `printf` as so: `printf("…format string…", foo(source, size))`. *Why is this bad form?* Hint: think about what would happen if this were done many times.

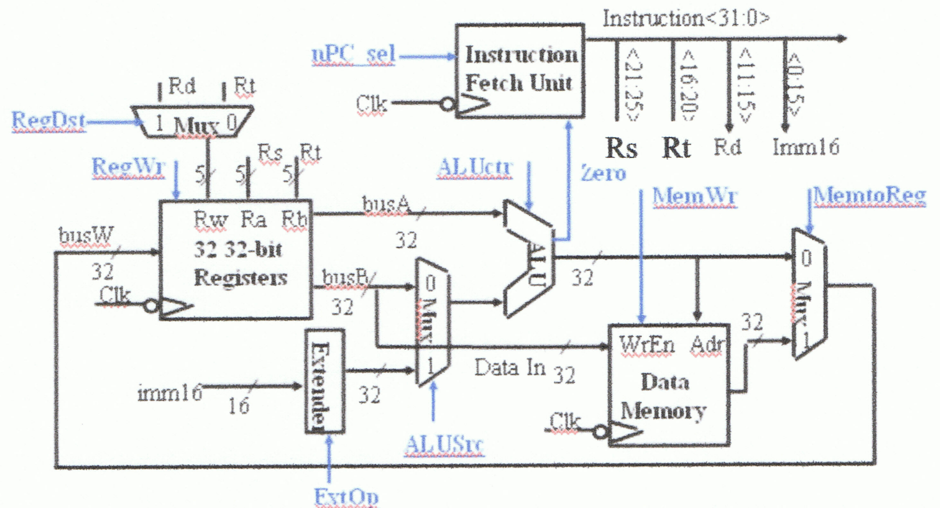
d) Let's say we removed the "**`sb $0, 0($t1)`**" instruction and then made the same call to `foo` from `printf` as in question (c) above: What are *all* the things that could happen?

Name: ______________________________ Login: cs61c-____

Post-Midterm Questions

F1) "Where's the sofr (sophomore, freshman) instr?" (18 pts, 24 min)

On the right is the single-cycle MIPS datapath presented during lecture. Your job is to modify the diagram to accommodate a new MIPS instruction. Your modification may use simple adders, shifters, mux chips, wires, and new control signals. If necessary, you may replace original labels.



Function calls in MIPS typically end with stack restoration and `jr $ra` as shown below. Because this happens often, we want to do this in *one* instruction instead of *two*. We'll design a new I-type instruction, `srjr` (stack restore, jump register), as follows:

```
addi  $sp, $sp, 16         ➔     srjr $ra $sp 16
jr    $ra
```

a) What is the RTL for `srjr` that will allow it to have the **widest range of stack restoration**? Hint: stack *restoration* always occurs in *non-negative word units*... The `jr` is done already.

____ ; PC = R[rs]

b) What is the most `$sp` could change as a result of a `srjr` call (in Bytes)? Use IEC format.

______________________________ Bytes (plus or minus a byte).

c) **Modify the picture above** and list your changes below. You may not need all the boxes. Please write them in "pipeline stage order" (i.e., changes affecting IF first, MEM next, etc)

(i)	
(ii)	
(iii)	
(iv)	
(v)	
(vi)	

d) 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 any *new* control signals you added.

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg			

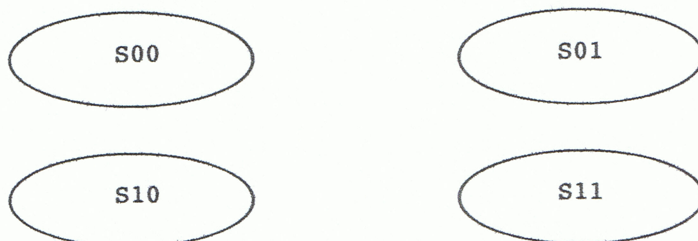
Name: ______________________________ Login: cs61c-____

F4) Synchronous Digital *Circus* (18 pts, 24 min)

We are designing a circuit with a 1-bit input (`I(t)`) and a 2-bit output (`O(t)`), that will produce, at time `t`, the number of *zeros* in the set `{I(t-2), I(t-1), I(t)}`. As an example,

the input: `I: 1 1 0 0 1 0 0 1 1 0 1 1 1 0 0 0`
...will produce the output: `O: 0 0 1 2 2 2 2 2 1 1 1 1 0 1 2 3`

a) Complete the FSM diagram below. Our states have been labeled `Sxy` indicating that the previous 2 bits,`{I(t-2), I(t-1)}` would be `{x, y}`. Fill in the truth table on the right. The previous state is encoded in (`P1`,`P0`), the next state is encoded in (`N1`,`N0`), and the output is encoded as (`O1`,`O0`). Make sure to indicate the value of the *output* on your state transitions.



P1	P0	I	O1	O0	N1	N0
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

b) Provide *fully reduced* (i.e., fewest gates to implement…you can use any *n*-input gates) Boolean expressions for the Output (`O1`,`O0`) and Next State (`N1`,`N0`) bits. If there is a name for any of the circuits, write it on the left. E.g., "The always-1", "3-input NAND", etc. A 2-input XOR has the symbol of "⊕".

Scratch space

________________ O1 =

________________ O0 =

________________ N1 =

________________ N0 =

c) Draw the overall circuit using the *fewest* gates possible with and without feedback below. You *may* add registers. "Feedback" means outputs are somehow fed back into inputs. Assume we've correctly implemented the answer to (b) as a black box in the middle.