CS61c-Midterm 1: 9/29/98: SOLUTIONS

Your family name _____ Your given name_____
Your Student ID number _____ login: cs61c_____

Please circle the last two letters of your login name.

b	c	d	e	f	g	h	i	j	k	l	m	n	a	o	p	q	r	s	t	u	v	w	x	y	z
b	c	d	e	f	g	h	i	j	k	l	m	n	a	o	p	q	r	s	t	u	v	w	x	y	z

The reason we asked this was so we would have some recourse if we could not read your handwriting. A surprising number of people left this out, including people whose login was unclear to us, confusing g and q, u and v, l and e, a and o.

Discussion section meeting time_____ TA's name _____

Look at the edge of your seat. Write your row and number. Your row number may not be visible from where you sit, so we will help you later. Row_____ Seat _____

This booklet contains 6 numbered pages including the cover page, plus 3 pages of excerpts from appendix B and C of Goodman and Miller. Put all answers on these pages, please; don't hand in stray pieces of paper. The exam contains 7 substantive questions, plus question 0 and the box immediately below.

I certify that my answers to this exam are all my own work, and that I have not discussed the exam questions or answers with anyone prior to taking this exam. If I am taking this exam early, I certify that I shall not discuss the exam questions or answers with anyone until after the scheduled exam time.

Signature _____

Question	Max Points	Your Points
0	1	
1	17	
2	5	
3	6	
4	6	
5	5	
6	5	
7	15	
8	20	
Total	80	

Question 0 (1 point): Fill out the front page correctly and put your login correctly at the top of each of the following pages..

Question 1 (20 points) These questions pertain to the mostly meaningless "program" on the next page.

a. Place a check in the appropriate column to identify which of the MAL instructions used below must be changed to more than one machine instruction (that is, requires more than one instruction when written in pure TAL). **And write out their translation into TAL in the space at the bottom of the page!**

b. Certain of the commands are not legal assembly language. Identify them with a checkmark as well.

c. At a number of points in the execution of this program, the value in register 4 changes. Place a checkmark in the appropriate column when, after the indicated instruction completes, register 4 has the number 5 in it.

You may assume that the machine you are executing the instructions on is an HP (big-endian) machine.
(By the way, this is irrelevant)

```
        .data
word5:  .word 5
abc5:   .byte 'a', 'b','c',5
zero:   .word 0
        .text
__start:        #                       (1a)              (1b)              (1c)
        # The main routine      #  Check if 2 ops    Check if it        Check if
        #                             in TAL       is  illegal          [$4]=5
main:   lui     $4, 0           1  ------------------|----------------|---------------
        ori     $4, 5           2  ------------------|----------------|------X--------
        lui     $4, 0           3  ------------------|----------------|---------------
        li      $4, 0x50005     4  --------X---------|----------------|---------------
        la      $4, word5       5  --------X---------|----------------|---------------
        lw      $4, word5       6  --------X---------|----------------|---------------
        addi    $4,$0,0         7  ------------------|----------------|-------X-------
        ori     $4, $4,0x101    8  ------------------|----------------|------( )tricky
        andi    $4, $4,0x0      9  ------------------|----------------|---------------
        ori     $4, $4, 0x5     10 ------------------|----------------|------X--------
        lw      $4, main+4      11 --------X---------|----------------|---------------
        sll     $4, $4,16       12 ------------------|----------------|---------------
        srl     $4, $4,16       13 ------------------|----------------|------X--------
        addi    $4, $0,4        14 ------------------|----------------|---------------
        addi    $4, $0,1        15 ------------------|----------------|---------------
        lw      $4, zero        16 --------X---------|----------------|---------------
        addi    $4, 4(abc5)     17 ------------------|-------X--------|---------------
        add     $4, abc5+3(0)   18 ------------------|-------X--------|---------------
        lw      $4, word5($0)   19 ---------X--------|-------X--------|---------------
        sw      $4, word5       20 ---------X--------|----------------|---------------
        lb      $4, abc5        21 --------X---------|----------------|---------------
        lb      $4, abc5+3      22 --------X---------|----------------|------X--------
        subi    $4,$0,-5        23 ------------------|----------------|------X--------
        done
```

Translations of MAL instructions

line number	instructions
4	lui $1 ,5 # make sure you know why NO OTHER INSTRUCTION # WORKS HERE!!!! ori $4, $1, 5
5	lui $1,high-part-of-address-of word5 # ditto comment above ori $4,$1,low-part-of-address-of word5
6	lui $1, high-part-of-address-of-word5 lw $4, low-part-of-address-of-word5($1) *could be lui, ori to get full address of word5 into $1, then lw $4,0($1)
11	lui $1, high-part-of-address-of-main # see * lw $4, 4($1)
16	lui $1, high-part-of-address-of-zero lw $4, 8 ($1)

……..the follow explanations are somewhat repetitive…

19	I thought this was illegal, since word5 is not a 16-bit displacement; however MAL made it into lui $1,high-part of address of word5 lw $4, 0($1) #ignore $0
	lui $1, high-part-of-address-of-word5
20	sw $4, 0($1)
	lui $1, high-part-of-address-of-abc5

```
21          lb $4, low-part-of-address-of-abc5($1) # probably 4

            lui $1, high-part-of-address-of-abc5
22          lb $4, lowpart-of-address-of-abc5+3($1) #probably 7
```

Grading: We thought of a scale like this, but we didn't use it but it would have wiped out too many of you….
column 1: 5 points for first 5 correct checks. Checks where they don't belong: -1.
column 2: 2 points for either 2 or 3 checks in right places. Checks where they don't belong, -1
column 3 : 5 points for 5 checks. Checks where they don't belong, -1
explanations: up to 5 points for the first 5 correct.

INSTEAD, this is what we did. We looked in specific locations only, and gave points based on what was there. We ignored all other places. We tried to pick the simplest cases that illustrated important features of MAL and MIPS. There were too many tricky things done by MAL that we, in retrospect, thought it was unfair for you to be tested on (and too many of you would have gotten them wrong!) – in particular MAL figures out a lot of commands that might seem illegal. And we (the instructor &TAs) were fooled on some of these as well!

So here is what we did. We looked in column c items 2,6,8,10,13,22,23. You got 7 points if you had the entries all right (note: 8 should be blank, the others are checked). We looked in column a, entries 1 2 3 4 5, as well as line 7 abc. You got 6 points for getting these all right (all blank except 4,5). Finally, you had to rewrite correctly the lines 4 and 5 (2 points each). All other entries were ignored. Including all entries other than 7 in column b.

17 points total

Question 2 (5 points). Here is a 2-instruction TAL program that is intended to load into register $4 the address in memory of its first instruction, the one at the location "here".

```
here: ????
step2:  addi $4, $31,-4
```

What instruction would you put in place of ???? __*JAL step2*__
If you put JAL you got 2 points. If JAL somewhere_not_step2 1 point.

Question 3 (6 points). Write down the number 2048_{10} in hexadecimal and binary.
In binary: 1000 0000 0000, in hex: 800 (1,1 pts)
Assuming you have a 2's complement representation, what is -2048 as a 16-bit binary number?
1111 0111 1111 1111 is one's complement. Add one to get
1111 1000 0000 0000 (2 pts)

Now write that number as a positive hexadecimal quantity. *0xf800 (2 pts)*

Question 4 (6 points). What do the following expressions evaluate to in C? (Show your work)

((1 & 4) | 3) & (~5) *= ((0..1&0..100)|0..11) & (1…1010) = (0|0..11)&1…1010 = 2 (3 pts)*

((1 && 4) || 3) && (!5) *= ((true and true) or true) and false = false (0) (3 pts)*

Question 5 (5 points). As we have discussed in lecture, one strategy in setting up register usage is to have one register, a global pointer, GP point to the beginning of the data segment of your program. That way it is easy for all programs to refer to data like error message strings with simple offsets from that base register.

Instead of setting GP to the beginning(numerically lowest) address in the `.data` segment, say `0x10000000`, it may be useful to set it somewhere above the lowest address in use, say to `0x10008000`. Why?
This way it is possible to use positive and negative offsets from the global pointer, and address twice as many words of data space. If you said something about addressing more locations but failed to mention negative offsets, you got between 0 and 2 points. I don't know why some people who said something like "it is faster searching from the middle" but this is so totally wrong, you should make sure you are cured of whatever notion suggested that answer to you. (5 points)

Question 6 (5 points). On the MIPS, architecture, the only addresses that one can branch to are on full-word boundaries (have 2 0s at the end), and therefore all branch instructions always use the 16-bit immediate as an 18 bit signed field. Since one can only load words from a full-word location, why aren't ALL the offsets in all the load and store instructions also multiplied by 4?

The load and store byte operations have to operate at any byte, not just word boundaries, so that requires full addressing. The load and store word instructions can also use odd offsets because the contents of registers may be odd (actually, not a multiple of 4). It is expected in these circumstances that that the sum of the two will be divisible by 4. It is not possible for the PC to be other than at a word boundary, and so the offset in the branch instructions must also be divisible by 4 to be sensible.

Question 7 (15 points). Here is a simple program to compute a function of one argument. It is based on a program you have seen in the lab, but this time it is not recursive.

a. See if you can make it smaller by eliminating (crossing out) unnecessary lines of assembler.

```
__start:
        li      $a0,4  #f(4)
        jal     f
        done

f:
        sub     $sp,$sp,8       # adjust the stack for 2 items
        sw      $ra,4($sp)      # save the return address
        sw      $a0,0($sp)      # save the argument n
        addi    $v0,$zero,1     # initialize answer
loop:   slt     $t0,$a0,2       # test for n < 2
        beq     $t0,$zero,L1    # if n>1, go to L1

        addi    $sp,$sp,8       # pop off 2 items off stack
        jr      $ra             #return to after jal

L1:     mul     $v0,$a0,$v0
        sub     $a0,$a0,1       # n >=1;  argument gets (n-1)
        j       loop
```

(4 points for realizing that there is no need to save and restore things that don't change. Most people got this all right. If you ADDED lines, you didn't read the question.)

b. There is no `subi` instruction on MIPS. Make believe you are the MAL assembler and convert `subi $a0,$a0,1` into one or more actual machine instructions (express these in TAL):
addi $a0,$a0,-1
or possibly something longer like
addi $at, $0,1
sub $a0,$a0,at

If you used other registers than $at , or you used non TAL instructions, then you were wrong and got 0.
(3 points)

c. Assume that the location of the instruction `beq $t0,$zero,L1` is `0x00400028`.
That instruction includes a field with a 16-bit displacement representing the distance *K* to branch to get to L1. (In hexadecimal), what is the value of *K*? (explain) . *The displacement is relative to the next instruction, since the PC will be set to PC+4 while executing the beq.*
The label L1 is 2 words (8 bytes) further along. Therefore the displacement is 8, but the actual field K will be 2, since it is always shifted to the left by 2 bits for use.
*Although the answer was **2** , partial credit was given for 8, 3, 12, for various excuses (4 points)*

e. Finally, what does f(4) return? *This is a version of factorial; it returns 24 in $v0 . Virtually everyone got this right. Though it computes f(0) wrong.*
(4 points)

Question 8 (20 points). Translate the following C procedure into MIPS assembler. Try to keep it brief! Use the usual conventions which assume that the arguments to procedures are kept in $a0-$a4 (regs 4-7) , temporary variables (preserved across call) are kept in $s0-$s7 (regs 16-23) . These can be used for internal computation but must be preserved by the callee across procedure calls. The return value is $v0 (reg 2). *[Points: for structure of subroutine/return; 5; successful return value setup: 5. Branching, testing: 5, getting a[x] into $a0 for call: 5; Excess number (>2) random errors, -2.]*

/* Question 8 */

```
int foo(int x, int *a){
  int z;
  if (x < 0) z = 0;
  else z = bar (a[x]);
return z;
}
```

```
foo:    addi    $sp,$sp,-4
        sw      $31, 0($sp)     #save return address
        bgez    $a0,else        # x>=0 goto else; use MAL
        addi    $v0,$0,0        #z=0  leave it in  return value!
        b       return
else:   sll     $a0,2           #multiply x by 4
        add     $a0,$a0,$a1     #get  address of a[x] into register $a0
        lw      $a0,0($a0)      #get value a[x] into $a0
        jal     bar             #call bar
return:
        lw      $31, 0($sp)     #put return address back in $31
        addi    $sp,$sp,4       #pop stack
        jr      $31
```

Notes:. You should put in comments, generally. It is not so clever to do checking before saving stuff on the stack, even if it is possibly faster. Reason: you stand a substantially higher chance of making a programming error. People who pushed stuff on the stack in the middle of the program did so erroneously in about 30% of the cases, and so lost points by being clever.

Many students seemed to think there was an instruction format like lw $a0, $a0($a1) which there is definitely NOT in MIPS. Some people left out the lw. Be sure you understand the difference between the address and the value of a quantity! Many people left out the label "foo". A few people save registers they did not use (no points off, but quite unnecessary). Some people seemed to use elaborate contortions to multiply an integer (x) by 4. The simplest is probably sll; the P&H book uses two adds to double. Using mul is slower and more code.