Name:_____ Login:_____

University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

CS61c Woojin Yu
Summer 2001

Midterm Exam

This is a *closed-book* exam. No calculators please. You have 1 hour. Each question is marked with its number of points.

This exam booklet should have 8 printed pages. Check to make sure that you have all the pages. Put your name and login neatly on each page.

Show your answers in the space provided for them. Write neatly and be well organized. If you need extra space to work out your answers, you may use the back of previous questions. However, only the answers appearing in the proper answer space will be graded.

Good luck!

Problem	maximum	score
1	35pts	
2	22pts	
3	23pts	
4	15pts	
Total	95pts	

Name:_____ Login:_____

MIPS instructions

Important – please note: The MIPS instructions shown in this table are the ones that you must use on the entire exam. **Do not use any instructions that are not in this table. If you use any instructions not listed below, you will lose points.**

The columns under "format" show the bit fields of the instructions. The number in the parentheses following each name is the number of bits in that field.

In the table, PC refers to the *program counter*.

You may carefully tear this page from your exam booklet for easy reference.

name	Format						Syntax	meaning
	Op(6)	rs(5)	rt(5)	rd(5)	Shamt(5)	func(6)		
add	0				0	32	add rd,rs,rt	rd = rs + rt
sub	0				0	34	sub rd,rs,rt	rd = rs - rt
and	0				0	36	and rd,rs,rt	rd = rs AND rt
or	0				0	37	or rd,rs,rt	rd = rs OR rt
sll	0	0				0	sll rd,rt,shamt	rd = logical shift rt left shamt bits
srl	0	0				2	srl rd,rt,shamt	rd = logical shift rt right shamt bits
slt	0				0	42	slt rd,rs,rt	if rs<rt set rd=1 else rd=0
jr	0		0	0	0	8	jr rs	PC=rs
	Op(6)	rs(5)	rt(5)	immediate(16)				
addi	8						addi rt,rs,immed	rt = rs + immed
andi	12						andi rt,rs,immed	rt = rs AND immed
ori	13						ori rt,rs,immed	rt = rs OR immed
lw	35						lw rt,immed(rd)	rt = MEMORY[rd+immed]
sw	43						sw rt,immed(rd)	MEMORY[rd+immed] = rt
lui	15						Lui rt,immed	rt = immed shifted left 16 bits
beq	4						beq rs,rt,label	branch if equal
bne	5						Bne rs,rt,label	branch if not equal
	Op(6)	target address(26)						
j	2						j label	jump
jal	3						Jal label	jump and link

Name:_____ Login:_____

1. Rewrite the following C source code using the MIPS Assembly instructions. Please follow the register conventions mentioned in class. ONLY USE THE INSTRUCTIONS ON PAGE 2.

```
int func1(int *a)
{
    int *temp;
    if (*a)
    {
       temp=a;
       a++;
       return(*temp+func1(a));
    }
    else
       return 0;
}
```

Answer

The following solution keeps closely to the C code:

```
func1:   # $s0 is a, $s1 is temp
     addi $sp, $sp, -16   # allocate stack
     sw   $s0, 0($sp)     # save some registers
     sw   $s1, 4($sp)
     sw   $ra, 8($sp)     # save the return address
     add  $s0, $0, $a0    # make s0 hold the value of a
     lw   $t0, 0($s0)     # derefrence a, t0 = *a
     beq  $t0, $0, else   # if (*a == 0) then do the else
     addi $s1, $0, $s0    # temp = a
     addi $s0, $s0, 4     # a++, a is an int pointer
     lw   $t1, 0($s1)     # dereference temp
     sw   $t1, 12($sp)    # save (*temp) so it survives the jal
     add  $a0, $s0, $0    # put a as arg to func1
     jal  func1           # call func1(a)
     lw   $t1, 12($sp)    # bring back the value of (*temp)
     add  $v0, $t1, $v0   # return = (*temp)+func(a)
     j    fin
else:
     add  $v0, $0, $0     # return 0
fin:
     lw   $s0, 0($sp)     # save some registers
     lw   $s1, 4($sp)
     lw   $ra, 8($sp)     # save the return address
     addi $sp, $sp, 16    # allocate stack
     jr   $ra
```

Grading

3 points for each of the following (partial credit was given):

1) allocating space on the stack and storing `$ra` and other registers you needed
2) dereferencing `a` correctly
3) implementing the branch correctly
4) implementing the pointer arithmetic correctly
5) setting `$a0` correctly
6) restoring `temp` if needed
7) dereferencing `temp`, if necessary
8) correct resulting value in `$v0`
9) storing 0 in `$v0` for the else case
10) restore stack, `$ra`, other registers

Name:_____ Login:_____

2. In the following, some of the statements are incorrect or illegal; cross out any such bad statements. Show in the spaces provided what the remaining print statements will print when the program is executed. Briefly state why the illegal statements are wrong.

```
int main()
{
    char a = 'A', ur[] = "ORDINALS", b = 'C';
    char *alpha = &a, *beta = alpha, *gamma = ur;
    char **aleph = &alpha, **beth = &beta
```

`printf("%c\n", *gamma);`	The letter 'O'.
`printf("%s\n", b);`	Incorrect, b isn't a char*.
`printf("%c\n", *beta);`	The letter 'A'.
`printf("%c\n", alpha);`	Incorrect, alpha isn't a char.
`alpha = ur + 1;`	Correct, but nothing printed.
`printf("%c\n", *(ur + 3));`	The letter 'I'.
`printf("%s\n", &ur[1]);`	The string "RDINALS".
`Ur = alpha;`	Illegal, can't assign to an array.
`printf("%c\n", *beta);`	The letter 'A'.
`beth = &(alpha);`	Correct, but nothing printed.
`If( (*aleph)[1] == (*beth)[1] )` `printf("CH is true.\n");`	This case is printed.
`Else` `printf("CH is false.\n");`	

```
    return 0;
}
```

Grading

Each part was worth 2 points. We tried not to let wrong answers cascade, but the way we did this is a little complicated to explain.

Name:_____ Login:_____

3. Convert this MIPS machine code into MAL (MIPS Assembly Language) instructions. Your final answers should use the register names, not the numbers (i.e. $t0, not $8)
Also, values which represent addresses (if any) should be converted into the full 32 bit address. Some space is left below each binary encoding to let you write down your work. However, you will only receive credit for what you write in the answer box.

Address:	Instructions:					
0x10001A00	001000	11101	11101	11111	11111	111100
0x10001A04	101011	11101	11111	00000	00000	000000
0X10001A08	000011	00000	10000	01000	00000	000111
0X10001A0C	000000	00010	00010	00010	00000	100000
0X10001A10	100011	11101	11111	00000	00000	000000
0X10001A14	001000	11101	11101	00000	00000	000100
0X10001A18	000000	11111	00000	00000	00000	001000

Answer Box:

Address:	Instructions:
0x10001A00	`addi $sp, $sp, -4`
0x10001A04	`sw $ra, 0($sp)`
0X10001A08	`jal 0x1041001C`
0X10001A0C	`add $v0, $v0, $v0`
0X10001A10	`lw $ra, 0($sp)`
0X10001A14	`addi $sp, $sp, 4`
0X10001A18	`jr $ra`

Grading
Each of the instructions were worth 3 points, except for the `jal`, which was worth 5. You lost 1 point for each mistake you made on an instruction, until you lost all the points associated with that instruction.

Name:_____ Login:_____

4. Short Answer Questions (5 Parts)
a. Assume an 8-bit two's complement machine on which all operations are performed on 8-bit registers. Answer the results of the following operations in hexadecimal. Assume that subtraction is done with SUBU and addition is done with ADDU.

(i)
```
  43 (hex)
- 4A (hex)
----------
  F9 (hex)
```

(ii)
```
  82 (hex)
+ AB (hex)
----------
  2D (hex)
```

Grading
Each part was worth 2 points. 12D for part (ii) got you 1 point, unless you said something about a trap not occurring on ADDU, or overflow not being detected, or something like that. For both parts, we also accepted correct answers in binary or decimal.

b. List the two values that can change on execution of the `jal` instruction.

Answer
The values we were looking for were `$ra` and `PC`. Some people put down `$v0`, `$a0`, or something like that; these might be changed in preparation for a `jal` or before returning from a `jal`, but not directly by the `jal` itself.

Grading
These were 1 point each, no exceptions.

c. Describe how the calculation of the target address for the `beq` instruction is different from that of the `j` instruction.

Answer
We accepted answers mentioning either that

1) The `beq` instruction is `PC`-relative, while the `j` instruction is absolute (well, it is a little bit PC-relative because it takes the top 4 bits of the `PC`, but that's beside the point).
2) The `beq` takes the 16-bit immediate and adds it (after multiplying by 4) to the PC, whereas the `j` takes the 26 bits in the instruction, multiplies by 4, and takes the top 4 bits of the `PC` for its own top 4 bits.

Grading
We were generous with partial credit on this question; if you had some idea, we gave you a point, but you had to be fairly detailed to get full credit. Yes, it was a little subjective.

d. What output would typically be seen from running the following (correct) program on a 32-bit machine, such as the MIPS machine we are studying? The `sizeof` operator determines (at compile-time) the size (in bytes) of the type yielded by its argument.

```
#include <stdio.h>

int main (void)
{
    char a[] = "foobar", b[15] = "baz", *c = "garply";
    int d[5] = { 1, 2, 4, 8, 16 };
    void bar (char*, int[]);

    printf("%d\n", sizeof(a));
    printf("%d\n", sizeof(b));

    bar(c, d);

    return 0;
}

void bar (char *c, int d[])
{
    printf("%d\n", sizeof(c));
    printf("%d\n", sizeof(d));
}
```

List, in order from first to last, the four values that appear.

Answer

1) 7. Since `a` is an array initialized without specific bounds, the initializer determines the array size. 6 for the letters in "foobar", plus 1 for the null-terminating character.
2) 15. `b` is a char array, with 15 elements, so it takes 15 bytes of storage.
3) 4. `c` is a pointer, and the typical size of a pointer on a 32-bit machine is 32 bits, or 4 bytes.
4) 4. `d` is actually a pointer to char, not an array, since arrays cannot be passed by value in C.

Grading

This was worth 4 points, 1 point for each correct printed value.

e. Write an equivalent, one-line statement in C. Do not use any bit operators or the comma operator (no &, |, ^, ~, <<, >>, or commas). Assume the C variable `unsigned int x` is stored in register $t0.

```
andi $t0, $t0, 63
```

Answer
We expected you to write something like `x = x % 64;` or similar. This is because we are basically masking out all bits past the 6th lowest (which represents 2^6). But since these bits all add on values that are multiples of 64, they would have left a remainder of 0 when divided by 64. A **lot** of people wrote something like `x = x + 63`. Perhaps you misread `andi` as `addi` (no comment), or you had an OR operation in mind, although that wouldn't quite have worked either.

Grading
This was worth 2 points. If you mentioned mod or modulo (but you didn't write the %), or if you had a % but the wrong modulus, we gave you a point if you were close; e.g., if you said `x = x % 63`.

```
Relaxation Question:

      What is your favorite movie?
```

Answer
The correct answer was *Forrest Gump*, but since this question wasn't worth any points, it didn't much matter what you put.