

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

CS 61c

J. Wawrzynek

Spring 2006

Machine Structures  
Midterm I

Your Name: \_\_\_\_\_

ID Number: \_\_\_\_\_ Your Seat Number: \_\_\_\_\_

Left Neighbor ID: \_\_\_\_\_ Right Neighbor ID: \_\_\_\_\_

This is an open-book exam. You are allowed to use any books and notes that you wish. No calculators or electronic devices of any kind, please. You have 2 hours. Each question is marked with its number of points.

This exam booklet should have 10 printed pages, plus 4 blank pages at the end. Check to make sure that you have all the pages. Put your student ID 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 or the blank sheets attached to the back of your exam booklet. However, only the answers appearing in the proper answer space will be graded.

Good luck!

problem	maximum	score
1	8pts	
2	5pts	
3	5pts	
4	10pts	
5	10pts	
6	10pts	
7	12pts	
total	60pts	

1. [8 points]

Which of these unsigned numbers is largest:  $\text{FadedAbe}_{\text{hex}}$ ,  $\text{DeadBeef}_{\text{hex}}$  or  $\text{FeedFace}_{\text{hex}}$ ?

What is  $\text{BOD}_{\text{hex}}$  in decimal?

What is  $337_{\text{ten}}$  in hexadecimal?

What is the decimal equivalent of the 6-bit two's complement number  $101010_{\text{two}}$ ?

Put a T (true) or F (false) in each table cell:

	unsigned	sign & magnitude	1's complement	2's complement
Can represent positive numbers				
Can represent negative numbers				
Has more than one representation for 0				
Uses the same addition process as unsigned	T			

2. [5 points] The following program is compiled and run on a MIPS computer.

```
1 int main() {
2     int    i;
3     int four_ints[4];
4     char*  c;
5
6     for(i=0; i<4; i++) four_ints[i] = 2;
7
8     c = (char*)four_ints;
9     for(i=0; i<4; i++) c[i] = 1;
10
11    printf("%x\n", four_ints[2]);
12 }
```

What does it print out? (The “%x” in `printf` is used to print out a word in hexadecimal format without any leading 0.)

If we change the 2 on line 11 to a 0, then recompile and run, what would be printed?

3. [5 points] The program below is written using the MIPS instruction set. It is loaded into memory at address 0xF000000C (all instruction memory addresses are shown below).

```
F000000C  loop:  addi  $1, $1, -1  #
F0000010          beq   $1, $0, done #
F0000014          j     loop      #
F0000018  done:
```

For each instruction in the program, write down the values (in decimal) of each field in the machine language version of that instruction using the following notation:

[ value | value | ... ]

For example, using this notation, the MIPS instruction `add $3,$2,$1` would be described as:

[ 0 | 2 | 1 | 3 | 0 | 32 ]

Put your answers to the right of the “#”s.

## 4. [10 points]

a) The following function should allocate space for a new string, copy the string from the passed argument into the new string, and convert every lower-case character in the new string into an upper-case character. Fill in the blanks and the body of the `for()` loop. You may *not* declare any new variable.

```
char* upcase(char* str) {
    char* p;
    char* result;

    result = (_____) malloc(_____);

    strcpy(_____, _____);

    for(_____ ) {

    }

    return result;
}
```

b) Consider the code below. The `upcase_name()` function should convert the  $i^{\text{th}}$  name to upper case by calling `upcase_by_ref`, which should in turn call `upcase()`. Complete the implementation of `upcase_by_ref`. You may *not* change any part of `upcase_name`.

```
void upcase_by_ref(_____ ) {

}

void upcase_name(char* names[], int i) {
    upcase_by_ref( &(names[i]) );
}
```

*Continued on next page ...*

Below is table for the ASCII character codes, that you might need for part b). The numbers along the left and top indicate the first and second hex digits of the codes, respectively.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	“	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

5. [10 points] The original MIPS processor did not support multiplication; compilers were expected to break down multiplication and division into simpler operations. Even on newer MIPS processors (that have the MUL instruction), compilers sometimes still do this to improve performance.

Consider the following C function:

```
int foo(int x) {
    return x*257;
}
```

Write the corresponding MIPS assembly code below. You may not use any form of MUL. Your answer must not be more than *two* instructions long (not including the jr \$ra).

```
foo:                                # $a0 contains x

                                     # return value should be in $v0
    jr $ra
```

Multiplication is more difficult when neither argument is known at compile time. The general procedure for achieving multiplication of two unsigned numbers is to use a series of shift and add operations (think about how long-hand multiplication works). The following assembly code multiplies two unsigned numbers, \$a0 and \$a1, leaving the result in \$v0. Assume that the result is sufficiently small that it fits in a single register.

Fill in the missing lines.

```
    addx $v0, $zero, $zero    # clear $v0

loop:  beq  $a1, $zero, done   # if $a1==0, we are done
       andi $t0, $a1, 1      # check bottom bit of $a1...
       beq  $t0, $zero, skip  # ...if it is 0, skip over
                                     # the next instruction

       -----, -----, ----- # fill me in!

skip:  srl  $a1, $a1, 1       # shift $a1 to the right

       sll  -----, -----, ----- # fill me in!
       j    loop             # repeat

done:  jr   $ra
```

6. [10 points] Consider the design of a new type of 16-bit processor, with the following characteristics:

- One machine word equals 16 bits.
- 16 16-bit registers.
- Byte-addressed memory of  $2^{16}$  memory locations.
- 16 different instruction opcodes (some defined below).
- Single word instruction format..

The table below lists a subset of instructions for this machine. Your job is to devise the machine language (instruction encodings). Each instruction will begin with a 4-bit opcode field on the far left of the instruction word, followed by whatever other fields are needed to encode the instruction.

To the extent possible you should model your instruction encodings after the MIPS. Obviously, there will be differences between the two because of the smaller instruction size and other differences between the two machines, but also many similarities. For instance, both machines use *PC-relative addressing* for branches, and *absolute addressing* for jumps. Also, as with the MIPS your encoding should allow the machine to branch (or jump) to the furthest instruction possible, given the constraints in machine encoding.

For each instruction in the table, divide the instruction word into fields and for each field specify the contents and the width in bits. Use the following notation:

[ name:width | name:width | ... ].

(With this notation, the MIPS r-format could be described as:

[ opcode:6 | rs:5 | rt:5 | rd:5 | shamt:5 | funct:6 ].)

*Continued on next page ...*



a) Fill in the right-most column with your instruction encoding for each instruction.

instruction	opcode	meaning	encoding
add ra,rb,rc	0	ra=rb+rc	
addi ra,immediate	5	ra=ra+immediate	
lw ra,offset(rb)	8	ra=memory[offset+rb]	
sw ra,offset(rb)	9	memory[offset+rb]=ra	
sll ra,rb,shamt	10	ra=rb>>shamt	
srl ra,rb,shamt	11	ra=rb<<shamt	
bez ra,label	12	if ra==0 go to label	
jmp label	15	go to label	

b) In the space below, describe the behavior of your branch instruction (**bez**) in the style of the following description for the MIPS **beq** instruction:

IF (rs==rt) PC = PC + 4 + (SignExtend(immediate) << 2) ELSE PC = PC + 4

c) Similarly, describe the behavior of your **jmp** instruction in the style of the following description for the MIPS **j** instruction:

PC = { PC[31..28], immediate, 00 }, where { , , } means concatenation.

7. [12 points] Below is a recursive version of the function `BitCount`. This function counts the number of bits that are set to 1 in an integer.

Your task is to translate this function into MIPS assembly code. The parameter `x` is passed to your function in register `$a0`. Your function should place the return value in register `$v0`. It is important to clearly comment your code to facilitate our grading process.

```
int BitCount(unsigned x) {
    int bit;
    if (x == 0) return 0;
    bit = x & 0x1;
    return bit + BitCount(x >> 1);
}
```

MIPS assembly code translation:

```
#####
## BitCount
## $a0 = x, $v0 = return value
#####
BitCount:
```

Scrap.

Scrap.

Scrap.

Scrap.