

Assembly language

Problem 1

Consider this C struct definition:

```
struct foo {
    int *p;
    int a[3];
    struct foo *sf;
} baz;
```

Suppose that register \$16 contains the address of baz.

For each of the following C statements, indicate which of the MIPS assembly language code fragments below (A-H) could be the result of compiling it.

```
codeA: lw    $8, 0($16)
        sw    $8, 4($16)

codeB: lw    $8, 0($16)
        lw    $9, 0($8)
        sw    $9, 4($16)

codeC: lw    $8, 4($16)
        sw    $8, 0($16)

codeD: sw    $16, 16($16)

codeE: lw    $17, 6($16)

codeF: lw    $17, 12($16)

codeG: lw    $8, 0($16)
        sw    $8, 16($16)

codeH: addi  $8, $16, 4
        sw    $8, 0($16)

___ number = baz.a[2];
___ baz.p = baz.a;
___ baz.a[0] = *baz.p;
___ baz.sf = &baz;
```

Problem 2

Translate the following C procedure to MIPS assembly language. Assume that arguments are passed in registers.

```
int garply (int a, int *b) {
    int c;
    c = subt(a >> 6);
    *b = a + *b;
    if (a < 0) || c < 0)
        return c;
    else
        return c | a;
}
```

Problem 3

Consider the following fragment of a C program.

```
int v[10], s;
int *p;
s = 17;
for (p = &v[3]; *p != 0; p++)
    s = s + *p;
```

Here is a buggy translation in MIPS assembly language, assuming s is in \$16 and p is in \$19.

```
                or    $16, $0, $0
                lw    $19, v+12
loop:           bne   $8, finish
                add   $16, $19, $16
                addi  $19, 1
                j     loop
finish:
```

There are six errors, including one missing instruction, in this translation. Find and fix them.

Problem 4

Consider the following MIPS assembly language routine. (The numbers on the left are just line numbers to help in your answer.) foo takes two integer arguments. The caller of foo and its callee bar follow the MIPS procedure call conventions. Assume var1 has been declared in the .data section with the .word directive.

```
1 foo: addi $sp, $sp, -20
2      sw   $s0, 16($sp)
3      sw   $s1, 12($sp)
4      la   $t0, var1
5      lw   $t0, 0($t0)
6      add  $t1, $a1, $a0
7      addi $s0, $t1, 10
8      add  $s2, $s0, $t1
9      add  $a0, $0, $s2
10     jal  bar
11     add  $t2, $t1, $v0
12     add  $s1, $t2, $a1
13     add  $v0, $0, $s1
14     lw   $s1, 12($sp)
15     lw   $s0, 16($sp)
16     addi $sp, $sp, 20
17     jr   $ra
```

- Describe four bugs that are present in the code.
- For each of these bugs, explain in one sentence either (i) why it will definitely cause the program not to work or (ii) under what condition will the program work correctly, in spite of the bug.

Problem 5

Compile the following C code into MIPS.

```
struct Node {
    int data;
    struct Node *next;
};

int sumList (struct Node *nptr) {
    if (nptr == NULL) return 0;
    else return (nptr->data + sumList (nptr->next));
}
```

Your code must contain meaningful comments and adhere to the MIPS calling convention and register usage conventions. You are allowed to use pseudoinstructions to make it more readable. It should be clean and well structured. It needs to be right, not optimal, but your answer cannot be longer than 20 instructions.

Problem 6

Translate the C function printDownUp to MIPS assembly language, retaining its recursive structure, passing its argument in the appropriate register, and following the usual register conventions. Translate putchar into a putc pseudoinstruction whose register argument contains the character to print.

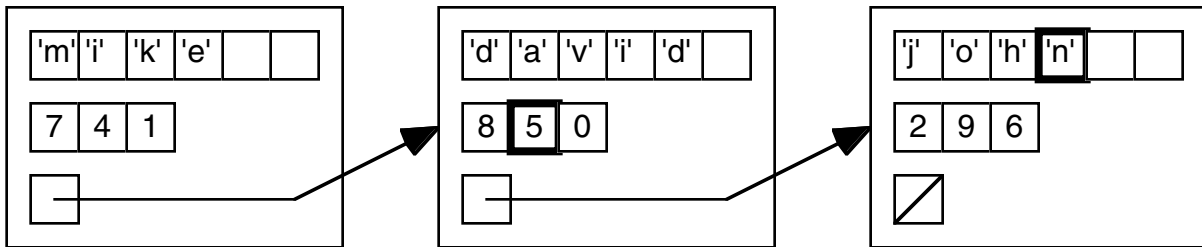
```
void printDownUp (char c) {
    if (c == 'a') {
        putchar (c);
    } else {
        putchar (c);
        printDownUp (c-1);
        putchar (c);
    }
}
```

Problem 7

Consider a list with nodes defined in C as follows.

```
struct ListNode {
    char name[6];
    int code[3];
    struct ListNode* next;
};
```

The diagram below, not drawn to scale, gives an example of such a list.



Part a

Assume that register \$a1 contains a pointer to the first node of the list. Write MIPS assembly language code that loads \$s2 with the second integer in the second node in the list (with the list pictured above, this will load a 5 into \$s2).

Part b

Again assume that register \$a1 contains a pointer to the first node of the list. Write MIPS assembly language code that loads \$s2 with the fourth character in the third node in the list (with the list pictured above, this will load 'n' into \$s2).

Problem 8

Consider the following C functions that check if one string contains another as a substring. The terms “string 1” and “string 2” are used in the comments to mean the strings represented by `s1` and `s2` respectively.

```
int containsAsSubstring (char *s1, char *s2) {
    if (*s2 == '\0') {                                /* if string 2 has run out, */
        return 1;                                    /* it's a substring of string 1. */
    } else if (*s1 == '\0') {                        /* if string 1 has run out, */
        return 0;                                    /* string 2 isn't a substring of string 1. */
    } else if (startsWith (s1, s2)) {
        return 1;
    } else {
        return containsAsSubstring (s1+1, s2);
    }
}

int startsWith (char *s1, char *s2) {
    if (*s2 == '\0') {                                /* any string starts with the empty string */
        return 1;
    } else if (*s1 == '\0') {                        /* if string 1 has run out, */
        return 0;                                    /* it doesn't start with string 2. */
    } else if (*s1 != *s2) {
        return 0;
    } else {
        return startsWith (s1+1, s2+1);
    }
}
```

Some examples of how `containsAsSubstring` behaves are listed below.

string 1	string 2	result of <code>containsAsSubstring</code>
"abcde"	"abc"	1
"xyabc"	"abc"	1
"axbc"	"ab"	0
"xy"	"abc"	0

Fill in the missing code in the MIPS assembly language implementation of `containsAsSubstring` below. (Don't worry about `startsWith`.) Your code should perform as described in the accompanying comments, and should follow conventions described in class and in lab and homework assignment 6 for passing arguments and managing registers and the system stack. You may assume that neither argument pointer is null.

```

containsAsSubstring:
    # save registers on the stack

    # check base cases

    beqz $t1,returnTrue
    beqz $t0,returnFalse

    move $s0,$a0    # does string 1 start with string 2?
    move $s1,$a1
    jal  startsWith
    bnez $v0,returnTrue

    add  $a0,$s0,1  # no match; make recursive call
    move $a1,$s1
    jal  containsAsSubstring
    j    return
returnTrue:
    # prepare to return 1

    j    return
returnFalse:
    # prepare to return 0

return:
    # restore registers and return

```

Problem 9

Here is the `pwdHelper` function from project 1. The declaration of `struct entryNode` appears on the last page of this exam.

```
void pwdHelper (struct entryNode * wd) {
    if (strcmp (wd->name, "/") != 0) {
        pwdHelper (wd->parent);
        printf ("/%s", wd->name);
    }
}
```

Write an assembly language version of `pwdHelper` that retains the recursive structure and follows all conventions for register use and stack management. You may use pseudoinstructions. Assume that functions named `strcmp` and `printf` are accessible from your function, and that they also follow all conventions for register use and stack management.

`.text`

`.data`

Problem 10

Part a

Given the following definition,

```
struct node {
    char name[12];
    int value;
};
```

what is sizeof (struct node)? _____

Assume that the sizes of chars and ints are the same as on the 271 Soda computers.

Part b

Translate the following code to assembly language in the space that follows. Your solution should adhere to conventions described in P&H. Comments in your code will help us understand your solution approach, and may earn you partial credit for an incorrect solution.

```
void exam1 (struct node **to) {
    exam2 (*to);
    (*(to-1))--;
}
```

prolog: save information on stack if necessary

exam1:

call exam2

compute (*(to-1))--

epilog: restore necessary things and return

Problem 11

Suppose that the label `names` marks the beginning of an array of strings. In MIPS assembly language, this might appear as follows:

```
names:  .word    starting address of first string
        .word    starting address of second string
        ...
```

Give a MIPS assembly language program segment that loads the fourth character of the second string into register `$t0`. For example, if the array contains the strings "mike", "clancy", "dave", and "patterson", this character would be the 'n' in "clancy". Assume that there are at least two strings in the array and at least four characters in the second string.

A three-line solution is sufficient. You may use any registers you want.

Problem 12

Complete the given framework to produce an assembly language function named `reverse` that implements the following (equivalent) Scheme and C functions:

Scheme

```
(define (reverse L soFar)
  (if (null? L) soFar
      (reverse (cdr L) (cons (car L) soFar) ) ) )
```

Equivalent C version

```
struct Thing {
  ... (as in project 1)
}
typedef struct thing *ThingPtr;
ThingPtr reverse (ThingPtr L, ThingPtr soFar) {
  if (L == NIL) {
    return soFar;
  } else {
    return reverse (L->th_cdr, cons (L->th_car, soFar));
  }
}
```

The code you supply should match the associated comments. Don't worry about memory allocation; the `cons` function will deal with that.

`reverse:`

```
# Save relevant registers on stack.
```

```
# Check base case.
```

`recursive:`

```
# Prepare for call to cons.
```

```
jal cons
```

```
# Prepare for recursive call to reverse.
```

```
jal reverse
```

`return:`

```
# Pop stack, restore relevant registers, and return the desired result.
```

Shifting and bitwise operations

Problem 13

Write a sequence of no more than six MIPS instructions that extracts bits 17:11 of register \$s0 and inserts them into bits 8:2 of register \$s1, leaving all the remaining bits of \$s1 unchanged. You may use \$t registers as temporaries.

Problem 14

Consider a function `isolateFloatFields` that isolates components of a normalized positive floating point value in IEEE 32-bit format. Given such a value, `isolateFloatFields` should return

- the exponent, and
- the integer that results from omitting the binary point from the fraction represented by the significand.

For example, if the value 2.875 base 10 (which is $1.0111 \cdot 2^1$) is passed to `isolateFloatFields`, it should return the integer 1 for the exponent and the integer whose binary representation is 10111 followed by nineteen zeroes for the significand.

Complete the assignment statements in the C version of the function `isolateFloatFields` below.

The `theBits` function returns an unsigned integer whose bits are the same as those of its float argument. It's needed since bitwise operators in C may not be applied to float values.

```
void isolateFloatFields (float x, int *exponent, int *fractBits) {
    unsigned int bits = theBits (x);
    *exponent = _____ ;
    *fractBits = _____ ;
}
```

Problem 15

Assume that \$t0 contains an I-format MIPS instruction. In both parts of this problem, you are to write an assembly language segment that puts the *sign-extended immediate field* of the instruction into \$t1. For example, if the instruction in \$t0 were the machine language encoding of `addi $a0,$a0,-17`, you would store -17 in \$t1. You may use pseudoinstructions and other temporary registers in your solution.

Part a

Give an assembly language program segment that copies the sign-extended immediate field of the machine code instruction in \$t0 into \$t1, that consists *only* of shift instructions.

Part b

Give an assembly language program segment that copies the sign-extended immediate field of the instruction in \$t0 into \$t1, that does not contain any shift instructions.

Problem 16

In lab, you wrote a function that returned the contents of the various fields of a MIPS I-format instruction. In this problem, we consider a similar task for the Prune 100 computer. The Prune, like the MIPS, has 32-bit instructions. The Prune has only 16 registers. In an I-format Prune instruction, the meaning of the bits is as follows.

- The first 8 bits are the op code.
- The next 4 bits are the register to be modified by the instruction.
- The last 20 bits are the immediate operand, in 1's complement.

Thus the equivalent to the MIPS assembly language instruction `addi $10,-2` might appear in hexadecimal as

```
94 af ff fd
```

if the op code for the `addi` instruction were 94 base 16.

On the next page, write a MIPS assembly language function `splitIFormat` that returns the contents of the register and immediate fields of a Prune 100 I-format instruction. If written in C, its prototype would be

```
void splitIFormat (int instr, int *register, int *immediate);
```

Follow the conventions described in class and in lab for passing arguments and managing registers and the system stack. Provide comments sufficient for the graders to understand your work.

Machine language; architecture; the assembly process

Problem 17

What is the result of interpreting 0x82988000 as a *MIPS instruction*? Give your answer as an assembly language instruction, use numeric register names, and show intermediate steps.

Problem 18

Which of the following is true of the *ori* instruction? Briefly explain your answer.

- ori* is always translated by the assembler into a single native MIPS instruction.
- ori* is always translated by the assembler into a sequence of two or more native MIPS instructions.
- ori* is sometimes translated by the assembler into a single native MIPS instruction and sometimes into a sequence of two or more native MIPS instructions.

Problem 19

Why did the MIPS designers use PC-relative branch addressing (One sentence is enough!)

Problem 20

Assemble the following MIPS instructions into executable binary. Show the position of each field by drawing a box around the corresponding bit positions.

address	assembly language instruction	machine language instruction
0x400000	addi \$a0, \$a0, -4	
0x400004	L0: bne \$s1, \$t2, L1	
0x400008	lw \$s2, 128(\$sp)	
0x40000c	j L0	
0x400010	L1: subu \$v0, \$a0, \$s0	

Problem 21

Decode the following binary numbers as MIPS instructions and give the equivalent MIPS assembly language statements.

address	value
0x40	10001100101101110000000000100100
0x44	00000010111001001011000000100011
0x48	0001111011000000111111111110000

Problem 22

Part a

Translate the following program segment to native MIPS instructions. You may use either names or numbers for the registers.

```
        li    $t1,-5
loop:   sub   $t1,$t1,3
        bgt  $t1,$a1,loop
```

Equivalent native MIPS segment:

Part b

Your answer to part a should include a branch instruction. Translate this branch instruction to machine language by filling in the boxes below with 0's and 1's.

Floating-point computation

Problem 23

Part a

Convert 6.25 to IEEE single precision. Show your work, and give your answer in binary.

Part b

Show all the steps involved in computing the single-precision floating-point sum of 0x43D55555 and 0x41ADDEB7. Give the result in hexadecimal. (Don't convert anything to decimal.)

Part c

What is the result of interpreting 0x82988000 as a *single precision IEEE floating-point value*? Give your answer as a sum of powers of 2, and show intermediate steps.

Problem 24

Encode the value 17.2510 according to the single precision IEEE floating-point standard and show its representation in hexadecimal.

Problem 25

Given below is a MIPS assembly language program segment that computes $(x+1.0)^2$ by adding x^2 to $2x$, then adding 1 to that sum.

```
        .data
x:      .float
answer: .float
one:    .float 1.0
        .text
__start:
        l.s    $f4,x
        l.s    $f6,one
        mul.s  $f8,$f4,$f4      # x2
        add.s  $f8,$f8,$f4      # + 2*x
        add.s  $f8,$f8,$f4
        add.s  $f8,$f8,$f6      # + 1.0
        s.s    $f8,answer
```

Part a

Consider the case where x is 2.0^{12} . What is the difference between the value stored in answer and the actual value of $(2.0^{12} + 1.0)^2$? (If the answer is computed correctly, the difference will be 0.) Show your work.

Part b

Does the sequence in which the terms are added affect the correctness of the answer? Briefly explain.

Problem 26

Consider the following C program segment.

```
int k, saved_k;
float x;
...
saved_k = k;
x = (float) k;
k = (int) x;
if (k == saved_k) {
    printf ("no change after conversion to float\n");
} else {
    printf ("change after conversion to float\n");
}
```

Recall that a cast converts the casted value to the given type. Thus if *k* contains the integer 3, the assignment

```
x = (float) k;
```

results in *x* containing the floating point value 3.0.

Assume for the following questions that an *int* and a *float* each use 4 bytes of memory, that a *double* uses 8 bytes of memory, and that a *float* and a *double* are stored using IEEE floating-point representation.

Part a

Find an *int* value *k* for which the above program segment produces the output

```
change after conversion to float
```

and give its hexadecimal (not decimal) representation.

Part b

Suppose that *x* in the above program segment was declared as *double*, with *k* being correspondingly cast to *double*. Would the output still be the same, using your answer to part a? Briefly explain.

Part c

Return now to the original program segment, and give the *largest* (signed) hexadecimal integer value that *k* could contain and still produce the output

```
no change after conversion to float
```

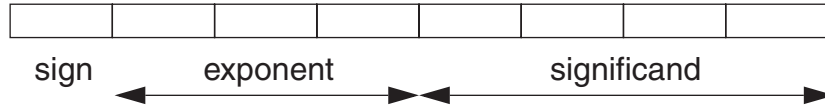
Briefly explain your answer.

Part d

Give the 4-byte (single precision) IEEE floating-point representation (in hexadecimal) of your answer to part c. Show how you got your answer.

Problem 27

Consider a representation (diagrammed below) for storing 8-bit floating point values that's exactly the same as the IEEE floating point representation except that three bits are allocated to the exponent and four to the significand.



Part a

Express in decimal the value represented by the byte 0xC1. Show your work for full credit.

Part b

Let a be the value represented by the byte 0xC1. Determine a value b that, when added to a using the byte counterpart of IEEE floating point addition, produces a result that's not equal to the algebraic sum of a and b . Express this value in hexadecimal, and verify the mismatch of the computed and the algebraic sum.

Linking

Problem 28

For each of the following utilities, specify what it takes as input and what it produces as output. Describe one key function it performs in this translation.

Compiler

Assembler

Linker

Loader

Problem 29

Given below are two assembly language program segments that are to be linked together with library code containing the `getchar` and `malloc` functions. On each line, specify *how many entries* in the relocation table would be produced by the assembler for the code on that line. (Put 0 for each line that doesn't generate a relocation table entry.)

Note that neither of these files is the result of compilation from C.

In the file main.s

In the file node.s

.text	_____	.text	_____
start:	_____	getNode:	_____
li \$t0,0	_____	addi \$sp,\$sp,-8	_____
sw \$t0,head(\$0)	_____	sw \$ra,0(\$sp)	_____
loop:	_____	sw \$s0,4(\$sp)	_____
jal getNode	_____	jal getchar	_____
beqz \$v0,gotAll	_____	beqz \$v0,return	_____
lw \$t0,head(\$0)	_____	ori \$v0,0x20	_____
sw \$0,4(\$v0)	_____	move \$s0,\$v0	_____
sw \$v0,head(\$0)	_____	li \$a0,8	_____
j loop	_____	jal malloc	_____
gotAll:	_____	sw \$s0,0(\$v0)	_____
...	_____	return:	_____
.data	_____	lw \$ra,0(\$sp)	_____
head:	_____	lw \$s0,4(\$sp)	_____
.word 0	_____	addi \$sp,\$sp,8	_____
	_____	jr \$ra	_____

Problem 30

Consider the following three machine instructions, which appear in memory starting at the address 0x00400000.

<i>address (in hex)</i>	<i>contents (in hex)</i>
00400000	12080002
00400004	3C11FFFF
00400008	08100004

Part a

“Disassemble” the instructions; that is, give an assembly language program segment that would be translated into the given machine language. You may use numeric rather than symbolic register names. A list of op codes (Figure A.19 from P&H) appears at the end of this exam.

Handle branches and jumps specially; where you would normally have a label, provide instead a hexadecimal byte address. For example, you should list a jump to the first instruction as

```
j 0x00400000
```

and represent a branch to the first instruction, say bltz, similarly as

```
bltz $9,0x00400000
```

Part b

For each of the instructions, indicate whether (a) it *must have contributed* an entry to the relocation table, (b) it *may have contributed* an entry to the relocation table, or (c) it *could not have contributed* an entry to the relocation table. Briefly explain your answers.

<i>address (in hex)</i>	<i>contents (in hex)</i>	<i>explanation of why this instruction must have, may have, or could not contribute relocation entry</i>
00400000	12080002	
00400004	3C11FFFF	
00400008	08100004	

Problem 31

Consider the following assembly language program segment, which loads \$t0 with the larger of \$a1 and an integer labeled by value.

```
        lui $at, upper half of value
        lw  $t1, lower half of value($at)
        slt $at, $t1, $a1
        beq $at, $0, t1greater
        add $t0, $0, $a1
        j   gotmax
t1greater:
        add $t0, $0, $t1
gotmax:
        ...
```

Part a

The table below lists some of the statements in the program segment. Indicate which of the statements listed below will be represented by an entry in the relocation table.

<i>statement</i>	<i>will it contribute an entry to the relocation table? (yes or no)</i>
lui \$at, upper half of value	
lw \$t1, lower half of value(\$at)	_____
beq \$at,\$0,t1greater	_____
j gotmax	_____

Part b

Given below is the part of the text segment of max.o that's the assembled version of the assembly language segment above. Assume that when the code is included in a program that is assembled into a file named max.o, the instruction labeled by tlgreater is the 25th instruction in max.o's text segment and the word labeled by value is the third word in max.o's data segment. Fill in the missing hexadecimal digits. Show your work.

<i>instruction</i>	<i>corresponding hexadecimal value</i>
lui \$at, upper half of value	3C01 _____
lw \$t1, lower half of value(\$at)	8C29 _____
slt \$at,\$t1,\$a1	0125 082A
beq \$at,\$0,tlgreater	1020 _____
add \$t0,\$0,\$a1	0005 4020
j gotmax	_____
tlgreater: add \$t0,\$0,\$t1	0009 4020
gotmax: ...	

Circuits and boolean algebra

Problem 32

Consider a logic circuit that, given inputs x_0 , x_1 , and x_2 , produces a binary encoding in outputs q_1 and q_0 of how many of the x_k are 1. A truth table relating q_1 and q_0 to the x_k appears below.

x_0	x_1	x_2	q_1	q_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Using and, or, not, and xor, design Boolean equations to represent the circuit. Your equations should be simplified where possible; show your work.