61C Spring 2002, MT2, Clancy

Problem 1 (6 points, 15 minutes)
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.

Statement	will it contribute an entry to the relocation table? (yes or no)
`lui $at, upper half of value`	______________
`lw $t1, lower half of value($at)`	______________
`beq $at,$0,t1greater`	______________
`j gotmax`	______________

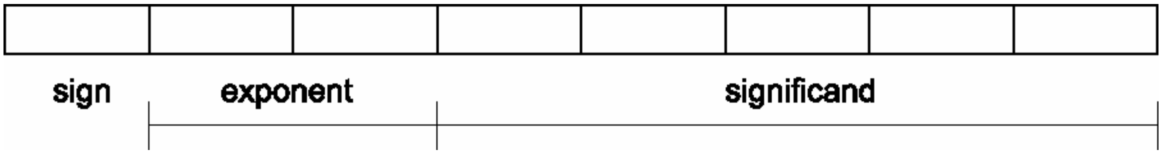
Part b

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 t1greater is the 33th instruction in max.o;s text segment and the word labeled by value is the 7th word in max.o's data segment. Fill in the missing hexadecimal digits.

Show your work.

instruction	corresponding hexadecimal value
lui $at, upper half of value	3C01 ___________
lw $t1, lower half of value($at)	8C29 ___________
slt $at,$t1,$a1	0125 082A
beq $at,$0,t1greater	1020 ___________
add $t0,$0,$a1	0005 4020
j gotmax	_________________
t1greater: add $t0,$0,$t1	0009 4020
gotmax: ...	

Problem 2 (6 points, 15 minutes)
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 the tree bits are allocated to the exponent and four to the significand.



Part a
Express in decimal the value represented by the byte 0xE1. Show your work for full credit. (A list of powers of 2 appears for your reference on the next page.)

Part b
Let a be the value represented by the byte 0xE1. Determine a value for 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.

Powers of 2

n	2^n
-7	0.0078125
-6	0.015625
-5	0.03125
-4	0.0625
-3	0.125
-2	0.25
-1	0.5
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048

Problem 3 (5 points, 14 minutes)
Complete the framework on the next page 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_cddr, cons (L->th_car, soFar));
     }
}
```

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

Framework to be completed

```
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.
```

Problem 4 (2 points, 5 minutes)

Under what conditions will execution of the instruction

```
sw $t0, 3($t0)
```

produce an error? Circle your answer, and briefly explain.

never sometimes always

Explanation: