Spring 2008 MT

Question 5: Three's a Crowd… (15 pts, 36 min)

Breaking news! We have just developed hardware that has 3-states: {`false=0`, `true=1`, and `maybe=2`}! Now we can store all our numbers in base 3. The race is on to develop a good encoding scheme for integer values.

Decimal	Ternary
5	12_{three}
26	
	1000_{three}

a) To warm up, first do some simple conversions between decimal and unsigned ternary. We've done one for you.

b) Suppose we have N ternary digits (*tets*, for short). What is the largest unsigned integer that can be stored? ____________________

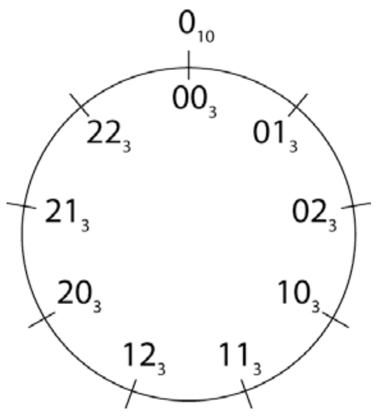
Ok, now that we've got unsigned numbers nailed down, let's tackle the negatives. We'll look to binary representations for inspiration.

c) Name two disadvantages of a *sign and magnitude* approach in ternary. Suppose a leading 0 means positive, and a leading 1 means negative, similar to what we did in the binary days.

____

____

d) Maybe *three's complement* will be more promising. To make sure we understand what that means, let's begin with a very small example – say a 2-tet number. Fill in the following number ring of tet-patterns with the values we'd like them to represent (just as in two's complement, we want all zeros to be zero, and want a balanced number of positive and negative values).



e) Recall that for an N-bit *two's* complement number, the bit-pattern of the largest positive number looks like 011…11. For an N-tet *three's* complement number, what does the tet-pattern of the largest positive number look like?

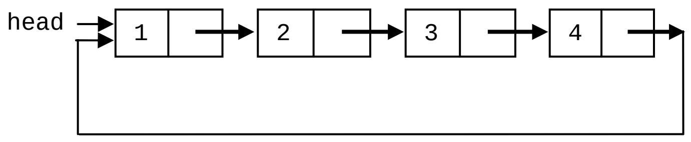
____

f) Provide (in pseudocode) an algorithm for *negating* an N-tet three's complement number.

Fall 2004 MT

Question 1: C and Circular Lists (22 points – 30 min.)

We're writing a circular linked list to keep numbers. The idea is very similar to a single-linked list, but the last element points to the first. Our circular linked list is made up of elements of type `pair` (a data type from CS61A and project 1). Assume when the list is empty we initialize the *global variable* `head` to `NULL`. Here's an example on the left, with the `pair` definition on the right:



The `pair` structure is defined as follows:

```
struct pair {
    int car;           // "number"
    struct pair *cdr; // next "pair"
} *head;
```

In the figure above, we can see 4 elements linked. When we insert an element, it goes *after the first element.* E.g., if we represent the distinct elements list in the example above *before* insertion as `{1 2 3 4}`, then *after* a call to `insert(5)` it would be `{1 5 2 3 4}`.

a) Help us to write the `insert` function by adding only 3 statements.

```
void insert(int d) {
      /* create the new node */

      tmp->car = d;

      /* Insert the new node in the right place */
      if ( head == NULL ) {
            /* The struct was empty… link the itself and we're done */
            tmp->cdr  = tmp;
            head      = tmp;
      } else {
            /* There were already elements in the linked list.
                Link the new node after the first element. */


      }
}
```

b) Instead we'd like you to link it after the "second" element. (If we only have 1 element, do the same as before. Can it be done by only modifying the 2nd and 3rd statements? If so, do it below. If not, explain why.

c)

```
/* Link the new node after the "second" element */
```

Or

Question 1 (continued): C and Circular Lists (22 points – 30 min.)

d) Now, we want to be able to delete the full structure. Assume that the OS immediately fills any freed space with garbage, so you cannot access freed heap contents. Finish the recursive `delete_recursively` function. We want the tightest, cleanest code possible (measured by the number of statements which terminate in semicolons). If you use only *2 semicolons*, full credit. If you use 3, you'll lose 1 point. If you use more, you'll lose 2 points.

```
void delete_full_structure()
{
      if (head == NULL )
            return;

      delete_recursively(head);
}
```

You saw how inserting a fifth element numbered 5 into our list messed up our numbering. We'd like to write `reset_numbers` that *clobbers* the node numbers to restore the nice `1, 2, 3,`… numbering. Note: A pointer to a `struct` stored in memory is just a pointer to memory we treat as broken up into the fields.

```
      void reset_numbers(pair* p, int i)
      {
            if (p != NULL ) {
                  p->car = i;
                  reset_numbers(->cdr, i++);
            }
      }
```

e) Convert `reset_numbers` to MIPS **keeping its structure recursive**. I.e., don't hand-optimize.

prologue
body
epilogue

f) In one sentence, what happens *on an actual MIPS machine* if we call `reset_numbers(head, 1)` on the lists as described in this problem? (Assume our list is not empty).

MIPS

Consider `visit_in_order`, then translate into MIPS.

```
typedef struct node{
     int value;
     struct node* left;
     struct node* right;
} node;

void visit_in_order(node *root, void (*visit)(node*)) {
     if (root) {
          visit_in_order(root->left, visit);
          visit(root);
          visit_in_order(root->right, visit);
     }
}
```

Fall 2005 MT

Question 2: Bit fields and bit fields of wheat…(7 pts, 15 min.)

You believe the encoding of MIPS opcodes, functions and bit field widths should be modified. What we currently use is on the left and what you propose is on the right – note you have not changed the code for the R and I cases, just reordered them. We've left out the code (replacing it with "??") for determining *which* R and *which* I instruction it is, assume that can be worked out later. How many R-type, I-type and J-type instructions did we have and will we have and what is one big pro and two big cons of this proposal? Consider how this subtle change could change the bit fields for the instructions for better or worse. Yes, we already know we'd probably have to reprint things like the green sheet. When counting instructions, only count the number of different operators. E.g., you should count `jr $v0` and `jr $a1` as the same instruction (same `jr` operator). We've filled one in for you already.

Current

```
if ((bit[31]…bit[26] == 0) {

    // Look elsewhere for which R it is…
    inst_type = Rl inst = ??;

} else if ((bit[31]…bit[26] == 2) {

    inst_type = J; inst = jump;

} else if ((bit[31]…bit[26] == 3) {

    inst_type = Jl inst = jal;

} else {

    // Remaining ops are I
    // Ignore Floating Pt formats FR,FI
    inst_type = I; inst = ??;
}
```

Proposed (changes in **bold**)

```
if (bit[31] == 1) {// Most-signif. Bit

    inst_type = J
    if (bit[30] == 1)
        inst = jump;
    else
        inst = jal;

} else if ((bit[31]…bit[26] == 0) {

    // Look elsewhere for which R it is…
    inst_type = Rl inst = ??;

} else {

    // Remaining ops are I
    // Ignore Floating Pt formats FR,FI
    inst_type = I; inst = ??;
}
```

Current			Proposed (changes in **bold**)		
# of R-type	# of I-type	# of J-type	# of R-type	# of I-type	# of J-type
		2			

The pro is…	1.
The cons are…	1. 2.

Fall 2005 Final

F2) "*Cache*, money. Dollar bills, y'all." (18 pts, 24 min)

Given the following code for a MIPS machine with 8 byte blocks, an empty 128-entry *fully-associative* LRU L1 cache, 4 MiB `ARRAY_SIZE`, and `char A[]` starting at a block boundary (byte 0 of a block):

```
for (i = 0 ; i < (ARRAY_SIZE/STRETCH) ; i++) {                    /* # of STRETCHes    */
   for (j = 0 ; j < STRETCH ; j++)  sum     += A[i*STRETCH + j]; /* for each STRETCH */
   for (j = 0 ; j < STRETCH ; j++)  product *= A[i*STRETCH + j]; /* for each STRETCH */
}
```

a) What is the `T:I:O` bit breakup (assuming byte addressing)? ______:______:______

b) *Cache size* (data only, not tag and extra bits) in bytes? (Use IEC) ____________________

c) What is the largest `STRETCH` that *minimizes cache misses*? (Use IEC) ____________________

d) Given the `STRETCH` size from (c), what is the *# of cache misses*? (Use IEC) __________________

e) Given the `STRETCH` size from (c), if `A` ***does not** start* at a block boundary, *roughly* what is the ratio of the # of *cache misses* for this case to the number you calculated in question (d) above. (e.g., 8x, 1/16th) ____________________

f) Can a 32-bit MIPS machine make use of more than 4 GiB of physical memory? Why or why not?

____

g) Would the performance of such a machine be any better than one with 4 GiB of physical memory? Why or why not?

Spring 2011 MT

4. AMAT

Suppose a MIPS program executes on a machine with a single data cache and a single instruction cache, and

- 20% of the executed instructions are loads or stores
- the data cache hit rate is 95%
- the instruction cache hit rate is 99.9%
- the instruction and data cache miss penalty is 10 cycles
- the instruction and data cache hit time is 1 cycle

a) How many mempry references are there per executed instruction?

b) How many data cache misses are there per executed instruction?

c) How many instruction cache misses are there per executed instruction?

d) Assume that if there were no cache misses, the CPU would be 1. What is the CPI of the program given the cache miss rate above?

e) What is the average memory access time of the program?

Spring 2005 MT

Question 4: Float, float on… (10 pts, 25 min)

The staff of CS61C is constantly investigating new approaches and solutions to old problems. (Either that or we just don't know when to leave well enough alone!) Well, we've come up with a new floating point format that **obeys the rules of the IEEE 754 Floating Point Standard** (denorms, NaNs, ±∞) but is *only 13 bits long*. It has 1 sign bit, 3 exponent bits, and 9 bits in the fraction (significand):

S	E	E	E	F	F	F	F	F	F	F	F	F

Answer the following questions about this new float format. As a sanity-check, if you calculate the bias of this format, you should get 3.

a) What is the largest non-infinite positive number that can be represented? Leave your answer as a base 10 mixed fraction (i.e. K ± (N/D)) where either K or N can be 0.

b) What is the smallest positive number that can be stored in this format?

Convert the following floating point values in the format above (expressed in hexadecimal) to their numerical (base 10) equivalents, if appropriate.

c) `0x1000`

d) `0x1F00`

What is the representation for the following floating point numbers? (Write your answer in hex).

e) `-∞`

f) `-4.5`

Spring 2012 MT

4. Compiling Linking Interpreting etc.

a) In one word each, name the most common producer and consumer of the following items:

Choose from: linker loader compiler assembler programmer

(item)	This is the output of:	This is the input to:
bne $t0, $s0, done		
char *s = “hello world”		
app.o string.o		
firefox		

b) Circle ***all*** the advantages of interpretation over compilation:

- Interpreted programs often have a higher CPI, resulting in better resource use.
- Interpreted programs do not require compilation.
- An interpreted interpreter can interpret itself, eliminating any need for compilers.
- An interpreted program can run on many different platforms without modification.

c) Circle ***all*** the reasons why many programs are compiled instead of interpreted:

- Compilers produce programs that often execute faster.
- It is easier to write code when using a compiler than when using an interpreter.
- Interpreted programs cannot be scaled to many machines.
- It’s harder to reverse engineer a compiled program than an interpreted one.