Problem Set 5 (Updated)

Computer Science 240
Fall 2014
Due: Friday, October 10

Relevant Reading. Patterson and Hennessy §2.5, §2.12-2.15

Problem 1. Which of the codes below are pseudoinstructions in MIPS assembly language (that is, they are not found directly in the machine language)?

(i) `addi $t0, $t1, 40000`
(ii) `beq $s0, 10, Exit`
(iii) `sub $t0, $t1, 1`

Problem 2. **THIS PROBLEM IS NOW OPTIONAL.** If any of the following instruction need editing during the *linking phase*, describe which instructions may need editing and why. For those instructions that do not require editing, describe briefly (one sentence) why not.

```
Loop:
     lui  $at, 0xABCD          # a
     ori  $a0, $at, 0xFEDC     # b
     jal  add_link             # c
     bne  $a0, $v0, Loop       # d
```

Problem 3. *(Supporting files are on the course website.)*
A *singly linked list* is a simple recursive data structure. Linked lists are represented by sequences of *nodes* or *elements*, each containing a value and a reference to the *next* node in the list. The first node in the list is called the *head* of the list. The last node in the list uses a *null* reference to indicate that there is no next node in the list. The null reference is traditionally encoded as address 0, the address of a memory location that normal programs are not allowed to use.

We have provided a simple linked list implementation in Java (written in a style that is closer to idiomatic C than idiomatic Java programming style). **LinkedList.java** defines a simple ListNode representation for building linked lists as well as methods to *show* lists, *add* a value to the end of a list, and *insert* a value at a specified index in a list.

We have also provided a partial MIPS translation of the provided Java linked list implementation. **LinkedList.asm** is a translation of the methods of LinkedList.java to MIPS code, using a simple memory layout to represent ListNode objects, described in comments in LinkedList.asm.

Your task is to implement the MIPS *insert* procedure by translating LinkedList.java's *insert* method. As you learn about the ListNode representation and design your *insert* procedure, answer the following questions.

Why must add and insert allocate allocate new *ListNode* objects on the heap? How could the program break if these procedures allocated space for *ListNode*s on the stack?

Problem 4.

We discovered a dump of memory contents stored in a file called "very-important-secrets.dump" in a CS 240 folder on a CS server that stopped working after losing power on October 2. It seems to show the contents of memory just before the server stopped. Clearly we need to investigate. We have disassembled part of the memory dump that appears to encode instructions for the most important procedure in the program that was running (possibly the keys to the future of computer science!). The disassembled code is:

```
Address     Code        Disassembled MIPS

0x003ffffc  0x24100007  addi $16,$0,0x00000007
0x00400000  0x20010003  addi $1,$0,0x00000003
0x00400004  0x0024082a  slt $1,$1,$4
0x00400008  0x14200007  bne $1,$0,0x0000000a
0x0040000c  0x2881fffe  slti $1,$4,0xfffffffe
0x00400010  0x14200005  bne $1,$0,0x00000008
0x00400014  0x00044080  sll $8,$4,0x00000002
0x00400018  0x3c011001  lui $1,0x00001001
0x0040001c  0x34290008  ori $9,$1,0x00000008
0x00400020  0x01094020  add $8,$8,$9
0x00400024  0x8D080000  lw $8, 0x00000000($8)
0x00400028  0x01000008  jr $8
0x0040002c  0x2402ffff  addiu $2,$0,0xffffffff
0x00400030  0x03e00008  jr $31
0x00400034  0x00851020  add $2,$4,$5
0x00400038  0x03e00008  jr $31
0x0040003c  0x02048020  add $16,$16,$4
0x00400040  0x72058002  mul $16,$16,$5
0x00400044  0x02061022  sub $2,$16,$6
0x00400048  0x03e00008  jr $31
0x0040004c  0x70a61002  mul $2,$5,$6
0x00400050  0x03e00008  jr $31
```

We also found what appears to be part of the static data section of memory:

```
Address     Contents of word at that address

0x10010000  0x0040002C
0x10010004  0x00400034
0x10010008  0x00400040
0x1001000C  0x0040003C
0x10010010  0x00400044
0x10010014  0x0040004C
0x10010018  0x0000001C
0x1001001C  0x00000400
```

Your job is to reconstruct a single Java method or C function that was likely compiled to generate this MIPS. Feel free to make up names of variables, etc., as needed.

– End of required problems –

Challenge Problem 1. Create LinkedListChallenge.java by converting LinkedList.java to a more object-oriented (and optionally recursive) style:

- A LinkedListChallenge object should hold an instance variable, *head*, referring to the first *ListNode* in the list.
- Convert methods that manipulate lists to instance methods of LinkedListChallenge, such that we can call, for example, `llc.insert(3,5)` instead of `insert(ll,3,5)`.

Translate your implementation to MIPS using dynamic dispatch.

- Create vtables for LinkedListChallenge and (if needed) ListNode.
- Add a vtable pointer in an object header attached to your in-memory ListNode representation.
- Implement method calls by dynamic dispatch, inspecting an object's header to find its vtable, then indexing in the vtable to lookup the address of the procedure to run.