CS536: Homework 9

Keith Funkhouser

December 1st, 2015

1

Generate the MIPS code for the following function:

```
int addArgs(int a, int b){
    return a + b;
}
```

Make sure that your function saves the return address and control link on the stack. You may choose to compute intermediate values on the stack (as discussed in class) or you may choose to use registers. List any conventions that you are assuming in your function (i.e. what is the offset or register where parameters are passed? What register or slot do you place return values?)

We are assuming:

- Parameters have been pushed onto the stack by the caller, as described in the notes
- `$V0` will be used for returning an int from a call to `f`
- Parameters are pushed onto the stack in the same way as illustrated in the lecture 20 slides, i.e. the first parameter is furthest towards the top of the stack, and the last parameter is furthest towards the bottom
- We have `$fp` pointing just above the parameters (as shown in the slides), compared to the notes which have `$fp` pointing just below them

```
.text
_f:
  # prologue
    sw   $ra, 0($sp)    # (1) push return addr
    subu $sp, $sp, 4
    sw   $fp, 0($sp)    # (2) push control link
    subu $sp, $sp, 4
    addu $fp, $sp, 8    # (3) set the FP to "bottom" of AR
  #call codegen for a + b, afterwards pushing the result onto the stack
    lw   $t0, 4($fp)    # load arg a into $t0
    sw   $t0, 0($sp)    # push a onto stack
    subu $sp, $sp, 4
    lw   $t0, 8($fp)    # load arg b into $t0
    sw   $t0, 0($sp)    # push b onto stack
    subu $sp, $sp, 4
    lw   $t1, 4($sp)    # pop b into $t1
    addu $sp, $sp, 4
    lw   $t0, 4($sp)    # pop a into $t0
    addu $sp, $sp, 4
    add  $t0, $t0, $t1  # add t1 to t0 in t0
```

```
    sw   $t0, 0($sp)    # push t0 onto stack
    subu $sp, $sp, 4
  #pop value from stack into $V0
    lw   $v0, 4($sp)    # pop value from stack into $v0
    addu $sp, $sp, 4
  #epilogue
    lw   $ra, 0($fp)    # load return address
    move $t0, $fp       # save control link
    lw   $fp, -4($fp)   # restore FP
    move $sp, $t0       # restore SP
    jr   $ra            # return
```

2

In class, we talked about how a control-flow graph represents control transfer in terms of edges on a graph. In this question, you will draw out the control flow graph corresponding to the following block of code:

```
a = 0;
b = 1;
if (a < c){
    if (b < c){
  }
}
for (c < 10){
    c++;
}
```

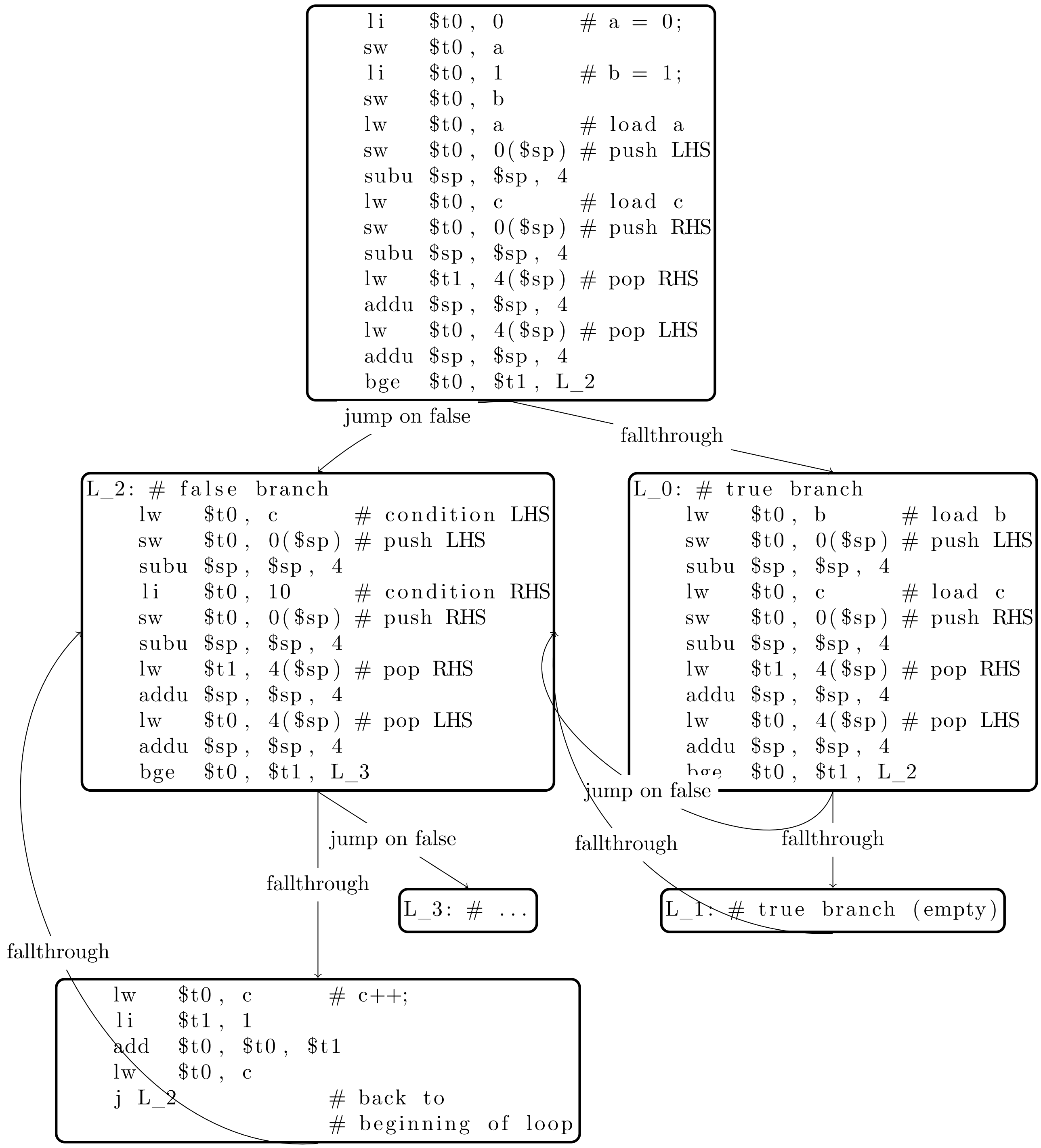
assume that all of the variables are declared globally.

Consider the following MIPS instructions for the code above:

```
    li   $t0, 0      # a = 0;
    sw   $t0, a
    li   $t0, 1      # b = 1;
    sw   $t0, b
    lw   $t0, a      # load a
    sw   $t0, 0($sp) # push LHS
    subu $sp, $sp, 4
    lw   $t0, c      # load c
    sw   $t0, 0($sp) # push RHS
    subu $sp, $sp, 4
    lw   $t1, 4($sp) # pop RHS
    addu $sp, $sp, 4
    lw   $t0, 4($sp) # pop LHS
    addu $sp, $sp, 4
    bge  $t0, $t1, L_2
L_0: # true branch
    lw   $t0, b      # load b
    sw   $t0, 0($sp) # push LHS
    subu $sp, $sp, 4
    lw   $t0, c      # load c
    sw   $t0, 0($sp) # push RHS
    subu $sp, $sp, 4
    lw   $t1, 4($sp) # pop RHS
    addu $sp, $sp, 4
```

```
    lw   $t0, 4($sp) # pop LHS
    addu $sp, $sp, 4
    bge  $t0, $t1, L_2
L_1: # true branch (empty)
L_2: # false branch
    lw   $t0, c      # condition LHS
    sw   $t0, 0($sp) # push LHS
    subu $sp, $sp, 4
    li   $t0, 10     # condition RHS
    sw   $t0, 0($sp) # push RHS
    subu $sp, $sp, 4
    lw   $t1, 4($sp) # pop RHS
    addu $sp, $sp, 4
    lw   $t0, 4($sp) # pop LHS
    addu $sp, $sp, 4
    bge  $t0, $t1, L_3
    lw   $t0, c      # c++;
    li   $t1, 1
    add  $t0, $t0, $t1
    lw   $t0, c
    j L_2            # back to
                     # beginning of loop
L_3: # ...
```

The control flow graph is then as follows:



3

Consider the following block of MIPS code:

```
.text
main:
    li $t0 1
    li $t1 2
```

```
addu $t0 $t1 $t0
sw $t0 ($sp)
sw $t1 4($sp)
li $t2 8
subu $sp $sp $t2
lw $t3 4($sp)
lw $t0 8($sp)
li $ra 0x0
jr $ra
```

List the values in each of the following registers immediately after the `jr` instruction:

- `$t0`: 3
- `$t1`: 2
- `$t2`: 8
- `$t3`: undefined
- `$ra`: 0x0 (hex for 0)
- `PC` (the instruction pointer): 0x0 (

If any value is undefined by the function put **undefined** as the value.