Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.035, Fall 2000 Handout 29 – Codegen Quiz Solutions Monday, November 1

Name:

---

1. *(5 points)* The following four bits of MIPS assembly code is the prologue for the procedure foo. One of the four choices is a possible correct compilation; the others have one or more errors. Select the correct choice.

```
boolean foo(int a, int b) {
    ....
}
```

Answer: A

A:
```
_L4foo:
        sw $fp,-4($sp)
        move $fp,$sp
        sw $ra,-8($sp)
        sw $gp,-12($sp)
        sub $sp,$sp,16
        sw $a1,4($fp)
        sw $a2,8($fp)
    # code for foo
        ....
```

B:
```
_L4foo:
        sw $fp,4($sp)
        move $fp,$sp
        sw $ra,8($sp)
        sw $gp,12($sp)
        sub $sp,$sp,16
        sw $a1,4($fp)
        sw $a2,8($fp)
    # code for foo
        ....
```

C:
```
_L4foo:
        sw $fp,4($sp)
        move $fp,$sp
        sw $ra,8($sp)
        sw $gp,12($sp)
        sub $sp,$sp,12
        sw $a1,4($fp)
        sw $a2,8($fp)
    # code for foo
        ....
```

D:
```
_L4foo:
        sw $fp,-4($sp)
        move $fp,$sp
        sw $ra,-8($sp)
        sw $gp,-12($sp)
        sub $sp,$sp,12
        sw $a1,4($fp)
        sw $a2,8($fp)
    # code for foo
        ....
```

Q1	Q2	Q3	Q4	Q5	Q6	tot

Name:

---

2. *(5 points)* The following four bits of MIPS assembly code attempt to perform a short circuit execution of the code segment below, where `$a0` contains the value `a` and `$a1` contains the value `b`. One of the four choices is a possible correct compilation; the others have one or more errors. Select the correct choice.

```
if ((a > 10) && (b < 20))
     { return true; }
else
     { return false; }
```

Answer: A

A:
```
_L4foo_start:
        move $v0,$a0
        li $v1,10
        bgt $v0,$v1,_L9_rhs_eval
        b _L3Lblock
_L9_rhs_eval:
        move $v0,$a1
        li $v1,20
        blt $v0,$v1,_L2Lblock
        b _L3Lblock
_L2Lblock:
        li $v0,1
        b _L4foo_epilogue
_L3Lblock:
        li $v0,0
        b _L4foo_epilogue
_L8_if_out:
```

B:
```
_L4foo_start:
        move $v0,$a0
        li $v1,10
        sgt $t1, $v0,$v1
        move $v0,$a1
        li $v1,20
        slt $t2, $v0,$v1
        and $v0, $t1, $t2
        bne $v0, $zero, _L3Lblock
        li $v0,1
        b _L4foo_epilogue
_L3Lblock:
        li $v0,0
        b _L4foo_epilogue
_L8_if_out:
```

C:
```
_L4foo_start:
        move $v0,$a0
        li $v1,10
        bgt $v0,$v1,_L3Lblock
        move $v0,$a1
        li $v1,20
        blt $v0,$v1,_L2Lblock
        b _L3Lblock
_L2Lblock:
        li $v0,1
        b _L4foo_epilogue
        b _L8_if_out
_L3Lblock:
        li $v0,0
        b _L4foo_epilogue
_L8_if_out:
```

D:
```
_L4foo_start:
        move $v0,$a0
        li $v1,10
        bgt $v0,$v1,_L2block
        b _L3Lblock
        move $v0,$a1
        li $v1,20
        blt $v0,$v1,_L2Lblock
        b _L3Lblock
_L2Lblock:
        li $v0,1
        b _L4foo_epilogue
        b _L8_if_out
_L3Lblock:
        li $v0,0
        b _L4foo_epilogue
_L8_if_out:
```

Name:

---

3. *(5 points)* For an access to the Espresso array `A` of integers, let `$a0` contain the base address of `A` and let `$v0` be the index `i`. The following four bits of MIPS assembly code attempts to store the value of `A[i]` in `$s3`? One of the four choices is a possible correct compilation; the others have one or more errors. Select the correct choice.

Answer: A

```
A:      mulo $v0, $v0, 4
        add $v0, $v0, $a0
        lw $s3, 4($v0)
```

```
B:      mulo $v0, $v0, 4
        add $v0, $v0, 1
        add $v0, $v0, $a0
        lw $s3, 0($v0)
```

```
C:      add $v0, $v0, 1
        mulo $v0, $v0, 4
        lw $s3, $a0($v0)
```

```
D:      mulo $v0, $v0, 4
        add $v0, $v0, 1
        add $v0, $v0, $a0
        la $s3, -4($v0)
```

Name:

---

4. *(15 points)* The following MIPS assembly comes from the output of an optimizing Espresso compiler:

```
    # method: bar
_L4bar:
        sw $fp,-4($sp)
        move $fp,$sp
        sw $ra,-8($sp)
        sw $gp,-12($sp)
        sub $sp,$sp,24
_L4bar_start:
        bgt $a1,10,_L2Lblock
        b _L3Lblock
_L2Lblock:
        mulo $t9,$a2,2
        move $v0,$t9
        b _L4bar_epilogue
_L3Lblock:
        mulo $t9,$a3,3
        move $v0,$t9
        b _L4bar_epilogue
_L8_if_out:
_L4bar_epilogue:
        move $sp,$fp
        lw $fp,-4($sp)
        lw $ra,-8($sp)
        lw $gp,-12($sp)
        jr $ra
_L4bar_end:
```

   (a) How many arguments does this procedure use? (The answer is not zero!)
   Answer: 3

   (b) For each argument, list where it is located in the machine.
   Answer: arg1 in `$a1`, arg2 in `$a2`, arg3 in `$a3`

   (c) This procedure returns a value. Indicate the line above that tells you that.
   Answer: the lines that set `$v0`

   (d) What does this procedure calculate? Provide an expression in terms of its arguments, which you should call `arg1` through `arg`*n*.
   Answer: `if (arg1 > 10) { return(arg2*2); } else { return(arg3*3); }`

Name:

---

5. *(15 points)* Below is an Espresso procedure compiled into MIPS assembly code.

```
myproc:
        sw $fp,-4($sp)
        move $fp,$sp
        sw $ra,-8($sp)
        sw $gp,-12($sp)
        sub $sp,$sp,16
        sw $a0,0($fp)
        sw $a1,4($fp)
        sw $a2,8($fp)
        sw $a3,12($fp)
        lw $v0,4($fp)
        lw $v1,4($fp)
        mulo $v0,$v0,$v1
        la $v1,-16($fp)
        sw $v0,0($v1)
        lw $v0,-16($fp)
        move $sp,$fp
        lw $fp,-4($sp)
        lw $ra,-8($sp)
        lw $gp,-12($sp)
        jr $ra
```

(a) What does this procedure do (ten words or less)?
Answer: Squares its argument

(b) Indicate in the listing above the prologue, body, and epilogue of the function.
Answer: Prologue: lines 1-9, Body 10-15, Epilogue 16-20

(c) For the body (not the prologue and epilogue), write a possible sequence of low-IR instructions that could result in this assembly code.
Answer: There are many reasonable ones, anything that squares an argument is fine.

6. *(15 points)* Below are a few Decaf procedures.

```
int baz(int a, int b)
{
==>     return (b - a);
}

int bar(int a, int b, int c)
{
-->     return (baz(a, b) + c);
}

int foo(int a, int b, int c, int d, int e, int f)
{
        int i1, i2;

        i1 = bar(a, b, c);
-->     i2 = bar(d, e, f);
        return (i1 + i2);
}
```

Suppose we call `foo` with the arguments `(1, 2, 3, 4, 5, 6)`. Here, `==>` indicates the current execution point, and `-->` indicates a current call in progress. That is, we've frozen execution of `foo()` in progress.

Suppose `$sp` is `0x4000` upon entry to `foo`. Also assume that we're not using any registers `$t0` – `$t9` or `$s0 -- $s7`, and that all variables are kept somewhere on the stack.

In the empty stack on the next page, fill in the contents of each stack slot at the current execution point. We've indicated the slots that `i1` and `i2` are in, but you'll need to fill in their values. For the rest, you need to indicate what is being stored in each slot, along with its value. If the value is unknown, say so. You should use the convention described in handout 23. However, you can ignore `$gp`.

Name:

Answer: There are many variations on this, you might not have the same temporaries, or you might save extra registers, etc etc. Since we don't require you to use o32 on internal calls, you might not have reserved space for 4 arguments at each call site.

`0x4000` `foo`'s saved `$fp` : unknown

`0x3ffc` `foo`'s saved `$ra` : unknown

`0x3ff8` `i1` : 4

`0x3ff4` `i2` : unknown

`0x3ff0` temp storing `i1 + i2` : unknown

`0x3fec` empty (for alignment) : unknown

`0x3fe8` arg4 for `bar` : unknown

`0x3fe4` arg3 for `bar` : unknown

`0x3fe0` arg2 for `bar` : unknown

`0x3fdc` arg1 for `bar` : unknown

`0x3fd8` `bar`'s saved `$fp` : `0x4000`

`0x3fd4` `bar`'s saved `$ra` : unknown

`0x3fd0` temp storing `baz(a + b)` : unknown

`0x3fcc` temp storing `baz(a + b) + c` : unknown

`0x3fc8` arg4 for `baz`

`0x3fc4` arg3 for `baz`

`0x3fc0` arg2 for `baz`

`0x3fbc` arg1 for `baz`

`0x3fb8` `bar`'s saved `$fp` : `0x3fd8`

`0x3fb4` `bar`'s saved `$ra` : unknown

`0x3fb0` temp storing `b - a` : unknown or 1

....