Aidan Lopez

1

Assignment: Instruction Set Architecture (part 2)

Instructor: Elaheh Sadredini

elaheh@cs.ucr.edu

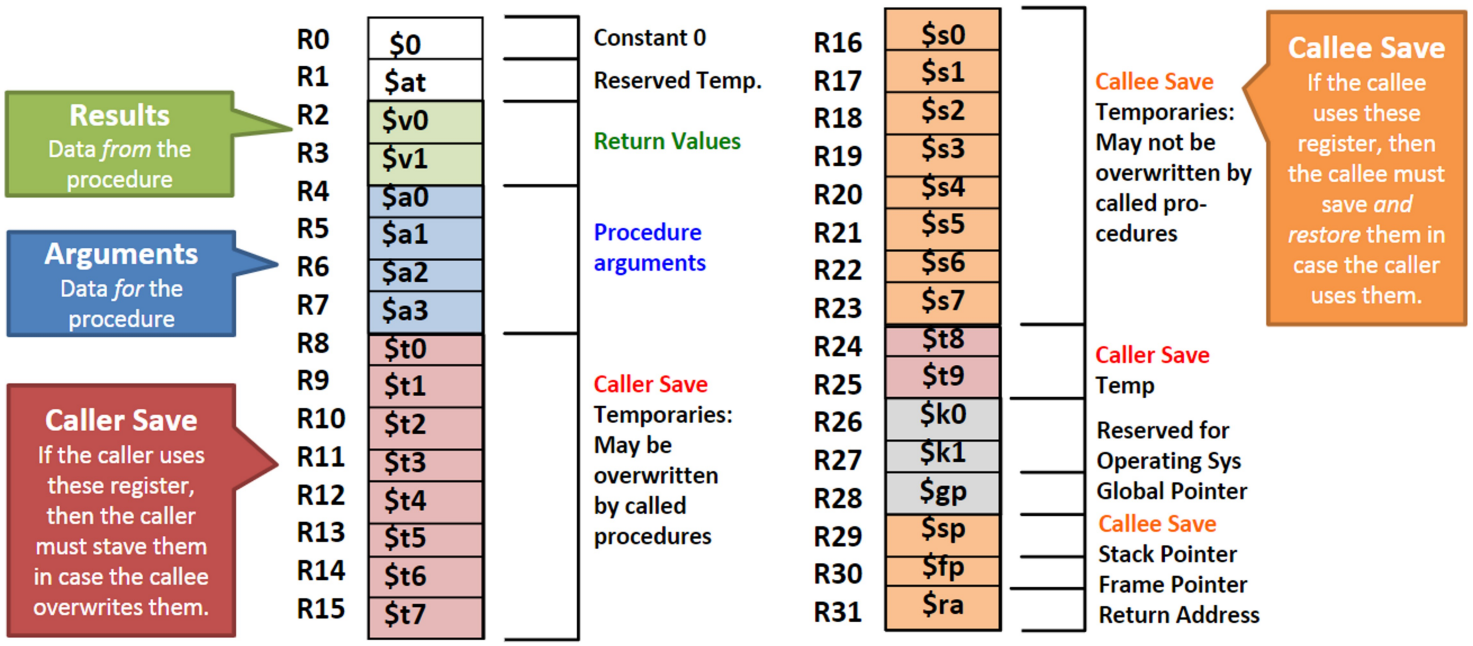
University of California, Riverside

2

Grading (total 4.2 points)

- Question 1: 0.75 point
- Question 2: 0.5 point
- Question 3: 0.5 point
- Question 4: 0.5 point
- Question 5: 0.75 point
- Question 6: 0.6 point
- Question 7: 0.6 point

Review: MIPS register names and conventions



Question 1: Encoding branches & immediates

We want to design a new K-format instruction for a branch-equal-immediate.

A: What information would need to be encoded in a **beq R2, 7, loop** instruction?

B: Why can't we have a **beq R2, 7, loop** instruction? and C: How could we fix this?

Hint: think about the instruction encodings and what we need to encode such an instruction.

A: beq R2, 7, loop needs to encode:

Name	Bit Fields					
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R-Format	op	rs	rt	rd	shmt	funct
I-format	op	rs	rt	address/immediate (16)		
J-format	op	target address (26)				
K-format						

B: Why can't we have this with the R/I/J formats?

C: What tradeoff could we make to implement a K-format?

Answer

- Answer A, B, and C. Your answer should be brief (1-3 sentences for each part)

A. For this we would first need the op code for beq(6 bits), then the first source register bits(5 bits), then the immediate(16bits), and the address for loop (16 bits).
B. This doesn't work with the R/I/J format because none of the formats utilize only 1 register and 2 immediates. Since immediates can be as large as 16bit values, the two would require 32 bits along with the 11 bits for the opcode and first source register.
C. To implement a K-format, we could impose a requirement on the compared immediate, limiting it's value to being a max of 5 bits in order to maintain a total of 32 bits.

Question 2

- If we added a new instruction format to MIPS that only specified one register, how large a constant could it hold? Briefly Explain why.

Answer

If a new instruction format that only specified one register was added, the amount of bits dedicated to the constant could be 21, holding a constant as large as -(2^(21-1)) to (2^(21-1)-1). Since the opcode is 6 bits and one register requires 5, 32 - 11 = 21 bits, thus the remaining 21 bits could be used to hold a constant

Question 3: Branch offset

Which of these will jump forward 4 instructions? Briefly explain the reason.

1. beq R1, R1, 16
2. beq R1, R1, 12
3. beq R1, R1, 4
4. beq R1, R1, 3
5. Can't tell: Conditional branches depend on the values in the registers

Answer

- Select the correct answer(s). Add a brief explanation.

beq R1, R1, 3 jumps forward 4 instructions. This is because the branch operation will take the current address and add the immediate multiplied by 4 + 4. Doing the math, 3*4 + 4 = 16, and since the program counter increments by 4 per instruction, 16/4 = 4, thus moving 4 instructions forward.

Question 4: Branch offset

A: Identify the destination for the jump

B: Fill in the constant needed to jump to that point.

(This code increments i inside the loop until it reaches 10, and then exits.)

```
add  $t0, $zero, $zero      # i = 0
addi $t1, $zero, 10      # j = 10
addi $t0, $t0, 1            # i++
slt  $t2, $t0, $t1       # t2=1 if (t0<t1) else 0
bne  $t2, $zero, _____      #  jump to ?
```

Answer

- Answer Part A and B. Add a brief explanation for each part.

A. Since the code is meant to increment i until i = 10, we need to jump to the address that contains the addi $t0, $t0, 1 instruction, since this increments i and the following instructions do the required checks.

B. The constant needed to jump to that point would be -8. since bne and beq are relative offsets (add to current PC). Adding since the PC increments by 4 for each instruction, and the desired instruction is 2 instructions back, adding -8 will cause the PC to call the instruction 2 instructions behind the current.

Question 5: Which are callers/callees?

Is main a caller/callee?

Is manipulate a caller/callee?

Is double a caller/callee?

```
main:
        addi $t4, $zero, 8        # a = 8
        addi $t5, $zero, 4        # b = 4
        jal  manipulate           # $s0 = manipulate(a, b)
        beq   $t4, $s0, main      # loop based on results
        j    Exit
manipulate:       # inputs: a and b
        add  $t2, $t0, $t1        # c = a + b
        sub  $t3, $t0, $t1        # d = a – b
        jal  double               # $t4, $t5 = double(c, d)
        add  $t4, $t4, $t2        # e = 2c + c
        sub  $t5, $t5, $t4        # result = 2d – e
        jr   $ra
double:           # inputs: q and r
        add  $v0, $a0, $a0              # result1 = q + q = 2q
        add  $v1, $a1, $a1              # result2 = r + r = 2r
        jr   $ra
Exit:                             # exit instructions
```

Answer

- Select the correct answer. No explanation is required.

Is main a caller/callee?

The main is a caller

Is manipulate a caller/callee?

Manipulate is a caller and callee

Is double a caller/callee?

Double is a callee

Question 6

- main() uses t0, t1, s0, s1, B() uses t4, s3, s4, C() uses t1, t2, t3, t4, s0, s5. How many words on the stack are needed when main() calls B() and B() calls C()? Explain why.

Answer

3 words are needed on the stack. C() uses t1, t4, S0 which are used in main() and B() respectively. Because of this,

3 words are needed on the stack. C() uses t1, t4, S0 which are used in main() and B() respectively. Because of this, C() could overwrite them, thus raising the need for their initial values to be saved on the stack.

Question 7: Procedure calls

Follow the register conventions for **arguments** and **results.** Figure out which registers "double" needs to save since it is the callee (see the code in Question5).

```
double:          # inputs: q and r

    add  $v0, $a0, $a0        # result1 = q + q = 2q

    add  $v1, $a1, $a1        # result2 = r + r = 2r

    jr   $ra
```

Answer

- Explain your answer.

  Double does not need to save anything. As convention states, the callee only has to save $s0-$s7, $sp, $fp, $ra. The return values are stored in $v0-$v1.