EE161 - Homework 2

Kaushik Vada

Question 1: Encoding branches & immediates

4

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?

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

a) You need to store the following:
- Opcode – tells the CPU it's a "branch-equal-immediate."
- Register (R2) – the register being compared.
- Immediate value (7) – the constant to compare with.
- Branch target (loop) – the offset or address to jump to if true.

So, `opcode + rs + immediate + branch offset` are the pieces that must fit into 32 bits.

b)
- R-format expects 3 registers, not an immediate or a branch address.
- I-format has room for one immediate OR offset, but not *two separate immediates* (the 7 and the branch address).
- J-format only holds a jump target, no register or immediate comparison fields.

So there's simply no existing format that can fit both a comparison immediate and a branch address

c) Basically, you'd trade encoding space (fewer bits per field) for instruction flexibility (more complex behavior).

Question 2

5

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

Opcode: 6 bits

Register Field: 5 bits

32 - 11 = 21 bits remaining

21 bit constant

Question 3: Branch offset

6

Which of these will move 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

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

Answer: beq R1, R1, 4

Question 4: Branch offset

7

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 ?
```

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

a) addi $t0, $t0, 1

b) Constant (offset) = -3

Question 5: Which are callers/callees?

8

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
```

a) Main → Caller (since it calls jal manipulate)

b) manipulate → Both caller/callee (calls Main, calls double)

c) double → Callee (called by manipulate & returns to its caller using jr $ra)

Question 6

9

- 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.
- **Note**: ignore $fp, $sp, and $ra registers

7 words

main - B → $t0, $t1 (2 words)

inside B → $s3, $s4 (2 words) (3 words)
$t4 = 1 word

inside C → uses $s0, $s5 (2 words)

Question 7: Procedure calls

10

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).

Note: the question is asking about register convention only in "argument" and "results".

Note: Explain your answer.

```
double:         # inputs: q and r

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

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

        jr   $ra
```

double does not need to save any registers.

$a0 and $a1 hold arguments and are caller-saved; $v0 and $v1 hold results and are overwritten freely by the callee. Since double doesn't touch any $s registers, it doesn't need to preserve anything.