Problem M1.2: CISC, RISC, and Stack: Comparing ISAs

Problem M1.2.A — CISC

How many bytes is the program? 19

How many bytes of instructions need to be fetched if b = 10?

(2+2) + 10*(13) + (6+2+2) = 144

Assuming 32-bit data values, how many bytes of data memory need to be fetched? Stored?

Fetched: the compare instruction accesses memory, and brings in a 4 byte word b+1 times: 4 * 11 = 44
Stored: 0

Problem M1.2.B — RISC

Many translations will be appropriate, here's one. We ignore MIPS32's branch-delay slot in this solution since it hadn't been discussed in lecture. Remember that you need to construct a 32-bit address from 16-bit immediate values.

x86 instruction	label	MIPS32 instruction sequence
`xor %edx,%edx`		*`xor r4, r4, r4`*
`xor %ecx,%ecx`		*`xor r3, r3, r3`*
`cmp 0x8047580,%ecx`	*`loop`*	*`lui r6, 0x0804`* *`lw r1, 0x7580 (r6)`* *`slt r5, r3, r1`*
`jl L1`		*`bnez r5, L1`*
`jmp done`		*`j done`*
`add %eax,%edx`	*`L1`*	*`add r4, r4, r2`*
`inc %ecx`		*`addi r3, r3, #1`*
`jmp loop`		*`j loop`*
`...`	*`done:`*	`...`

How many bytes is the MIPS32 program using your direct translation?

10*4 = 40

How many bytes of MIPS32 instructions need to be fetched for b = 10 using your direct translation.

There are 2 instructions in the prelude and 7 that are part of the loop (we don't need to fetch the 'j done' until the 11th iteration). There are 5 instructions in the 11th iteration. All instructions are 4 bytes. 4(2+10*7+5) = 308.

Note: You can also place the label '`loop`' in two other locations assuming `r6` and `r1` hold the same values for the remaining of the program after being loaded. One location is in front of the `lw` instruction, and we reduce the number of fetched byte to 268. The other is in front of the `slt` instruction, and we further decrease the number of fetched bytes to 228.

How many bytes of data memory need to be fetched? Stored?

Fetched: 11 * 4 = 44 (or 4 if you place the label '`loop`' in front of the `slt` instruction)
Stored: 0

Problem M1.2.C Optimization

There are two ideas that we have for optimization.

1) We count down to zero instead of up for the number of iterations. By doing this, we can eliminate the slt instruction prior to the branch instruction.

2) Hold b value in a register if you haven't done it already.

```
            xor r4, r4, r4
            lui r6, 0x0804
            lw r1, 0x9580(r6)
            jmp dec
loop:       add r4, r4, r2
dec:        addiu r1, r1, #-1
            bgez r1, loop
done:
```

This modification brings the dynamic code size down to 144 bytes, the static code size down to 28 and memory traffic down to 4 bytes.

Problem M1.3: Addressing Modes on MIPS ISA

Problem M1.3.A **Displacement addressing mode**

The answer is yes.

```
LW R1, 16(R2)      ➔    ADDI R3, R2, #16
                        LW R1, 0(R3)
```

(R3 is a temporary register.)

Problem M1.3.B **Register indirect addressing**

The answer is yes once again.

```
LW R1, 16(R2)      ➔

lw_template:   LW   R1, 0       ; it is placed in data region
               ...
  LW_start:    LW   R3, lw_template
               ADDI R4, R2, #16
               ADD  R3, R3, R4  ; R3 <- "LW R1, addr"
               SW   R3, _L1     ; write the LW instruction
       _L1:    NOP             ; to be replaced by "LW .."
```

(R3 and R4 are temporary registers.)

Yes, you can rewrite the code as follows.

```
Subroutine: lw   R6, ret_inst ; r6 = "j 0"
            add  R6, R6, R31  ; R6 = "j return_addr"
            sw   R6, return   ; replacing nop with "j return_addr"

            xor  R4, R4, R4   ; result = 0
            xor  R3, R3, R3   ; i = 0
loop:       slt  R5, R3, R1
            bnez R5, L1       ; if (i < b) goto L1
return:     nop               ; will be replaced by "j return_addr"
L1:         add  R4, R4, R2   ; result += a
            addi R3, R3, #1   ; i++
            j    loop
ret_inst:   j    0            ; jump instruction template
```