CAO HW2

Remy Jaspers - 4499336

February 25, 2017

1

To translate the for loop to MIPS, we first store 0 in a variable, say $s3:

add $s3, $s3, $zero
FOR_LOOP: slti $t0, $s3, 50
beq $t0, $zero, EXIT
sll $t1, $s0, 2
add $t2, $s0, $t1
lw $t3, 0($t2)
sll $t4, $s1, 2
add $t5, $s1, $t1
lw $t6, -4($t5)
add $t7, $t3, $t6
sw $t3, 0($t5)
addi $s3, $s3, 1
j FOR_LOOP
EXIT:

2a

For the first instruction

not $t1, $t2

First we convert $t2, containing 0x00FF05A4 to bits:

0000 0000 1111 1111 0000 0101 1010 0100

And invert each bit to produce the following sequence, which is stored in $t1:

1111 1111 0000 0000 1111 1010 0101 1011

For the second instruction:

orn \$t1, \$t2, \$t3

We convert \$t3 containing 0xFFFF003D, to bits and invert it:

!(1111 1111 1111 1111 0000 0000 0011 1101) =
0000 0000 0000 0000 1111 1111 1100 0010

Or \$t3 with \$t2 resulting in \$t1:

0000 0000 1111 1111 0000 0101 1010 0100 |
0000 0000 0000 0000 1111 1111 1100 0010 =
0000 0000 1111 1111 1111 1111 1110 0110

2b

The not instruction can be rewritten using a NOR (\$2.6 in the book), where one operand is the value which we want to invert, and the second operand is the \$zero register. The orn has to be done in two steps, first inverting the \$t3 and then or the result. The final mips code will then be:

$$\textbf{nor \$t1, \$t2, \$zero} \tag{1}$$
$$\textbf{nor \$t3, \$t3, \$zero} \tag{2}$$
$$\textbf{or \$t1, \$t2, \$t3} \tag{3}$$

2c

All three instructions are in the R-format. We have the following decimal values in the fields, and below the word representation is shown:

op | rs | rt | rd | shamt | funct
1)
0 | 10 | 0 | 9 | 0 | 39
000000 | 01010 | 00000 | 01001 | 00000 | 100111
2)
0 | 11 | 0 | 11 | 0 | 39
000000 | 01011 | 00000 | 01011 | 00000 | 100111
3)
0 | 10 | 11 | 9 | 0 | 37
000000 | 01010 | 01011 | 01001 | 00000 | 100101

2d

The MIPS instruction set already contains the minimal set of logical primitives AND, OR, NOR, XOR. We can implement all sorts of logic using these primitives. There's no need to create an OR NOT function in MIPS, this would go against the principle of keeping the hardware to implement these instructions simple.