Modos

Lucas

27 de agosto de 2015

Exercise 2.25 In this exercise, you will explore 32-bit constants in MIPS. For the following problems, you will be using the binary data in the table below.

```
a. 0010 0000 0000 0001 0100 1001 0010 0100 two
b. 0000 1111 1011 1110 0100 0000 0000 0000 two
```

2.25.1 [10] 2.10 Write the MIPS assembly code that creates the 32-bit constants listed above and stores that value to register $t1 .

```
a. lui $at, 0x2001
   ori $t1, $at, 0x4924

b. lui $at, 0x0fbe
   ori $t1, $at, 0x4000
```

2.25.2 [5] 2.6, 2.10 If the current value of the PC is 0x00000000, can you use a single jump instruction to get to the PC address as shown in the table above?

```
a. 0010 0000 0000 0001 0100 1001 0010 0100 two = 0x20014924

max(pc) -> pc+4[31-28] == 0000 :: 1111 1111 1111 1111 1111 1111 11 :: 00

           pc atual +4 == 0000    0000 0000 0000 0000 0000 0000 01    00
```

```
min(pc) -> pc+4[31-28] == 0000 :: 0000 0000 0000 0000 0000 0000 00 :: 00

ffffffc < 20014924, portanto não alcança.

b. 0000 1111 1011 1110 0100 0000 0000 0000 two = 0xFBE4000

max(pc) -> pc+4[31-28] == 0000 :: 1111 1111 1111 1111 1111 1111 11 :: 00

           pc atual +4 == 0000    0000 0000 0000 0000 0000 0000 01    00

min(pc) -> pc+4[31-28] == 0000 :: 0000 0000 0000 0000 0000 0000 00 :: 00

ffffffc > fbe4000, portanto alcança.
```

2.25.3 [5] 2.6, 2.10 If the current value of the PC is 0x00000600, can you use a single branch instruction to get to the PC address as shown in the table above?

```
a. 0010 0000 0000 0001 0100 1001 0010 0100 two = 0x20014924

max(pc) -> pc+4[31-28] == 0000 :: 1111 1111 1111 1111 1111 1111 11 :: 00

           pc atual +4 == 0000    0000 0000 0000 0000 0100 0000 01    00

min(pc) -> pc+4[31-28] == 0000 :: 0000 0000 0000 0000 0000 0000 00 :: 00

ffffffc < 20014924, portanto não alcança.

b. 0000 1111 1011 1110 0100 0000 0000 0000 two = 0x0fbe4000

max(pc) -> pc+4[31-28] == 0000 :: 1111 1111 1111 1111 1111 1111 11 :: 00

           pc atual +4 == 0000    0000 0000 0000 0000 0100 0000 01    00

min(pc) -> pc+4[31-28] == 0000 :: 0000 0000 0000 0000 0000 0000 00 :: 00

ffffffc > fbe4000, portanto alcança.
```

2.25.4 [5] 2.6, 2.10 If the current value of the PC is 0x1FFFf000, can you use a single branch instruction to get to the PC address as shown in the table above?

```
a. 0010 0000 0000 0001 0100 1001 0010 0100 two = 0x20014924

max(pc) -> pc+4[31-28] == 0001 :: 1111 1111 1111 1111 1111 1111 11 :: 00

           pc atual +4 == 0001    ffff ffff ffff ffff 0000 0000 01    00

min(pc) -> pc+4[31-28] == 0001 :: 0000 0000 0000 0000 0000 0000 00 :: 00

ffffffc < 20014924, portanto não alcança, por poucos bytes, precisaria de mais
um jump para completar.

b. 0000 1111 1011 1110 0100 0000 0000 0000 two = 0x0fbe4000

max(pc) -> pc+4[31-28] == 0000 :: 1111 1111 1111 1111 1111 1111 11 :: 00

           pc atual +4 == 0000    0000 0000 0000 0000 0100 0000 01    00

min(pc) -> pc+4[31-28] == 0000 :: 0000 0000 0000 0000 0000 0000 00 :: 00

ffffffc > fbe4000, portanto alcança.
```

2.25.5 [**10**] **2.10** If the immediate field of an MIPS instruction was only 8 bits wide, write the MIPS code that creates the 32-bit constants listed above and stores that value to register $t1.

```
a. 0010 0000 0000 0001 0100 1001 0010 0100 two = 0x20014924

lui $t1, 0x20
ori $t1, $t1, 0x01  # 0x20010000

lui $t2, 0x49
ori $t2, $t2, 0x24  # 0x49240000

srl $t2, $t2, 16    # 0x00004924

add $t1, $t1, $t2   # 0x20014924

b. 0000 1111 1011 1110 0100 0000 0000 0000 two = 0x0fbe4000

lui $t1, 0x0f
ori $t1, $t1, 0xbe  # 0x0fbe0000
```

```
lui $t2, 0x40
ori $t2, $t2, 0x00  # 0x40000000

srl $t2, $t2, 16    # 0x00004000

add $t1, $t1, $t2   # 0x0fbe4000
```

2.27.5 [10] 2.10 By reducing the size of the immediate fields of the I-type and J-type instructions, we can save on the number of bits needed to represent these types of instructions. If the immediate field of I-type instructions were 8 bits and the immediate field of J-type instructions were 18 bits, rewrite the MIPS code above to reflect this change. Avoid using the lui instruction.