Solutions for Exercise 2.23, 2.24, 2.26,2.27, 2.31, 2.39, 2.40, 2.41, 2.42

> Disclaimer:
> 1. The solution is just for your reference. They may contain some mistakes. DO TRY to solve the problems by yourself. Please also pay attentions to the course website for the updates.
>
> 2. Try not to use pseudoinstructions for any exercises that ask you to produce MIPS code. Your goal should be to learn the real MIPS instruction set, and if you are asked to count instructions, your count should reflect the actual instructions that will be executed and not the pseudoinstructions.

2.23 $t2 = 3

2.24

For jump instruction, it obtains bit 31-28 from PC (which is 0010 in this example). Therefore, it is impossible to jump to 0x40000000.

For more details, please refer https://chortle.ccsu.edu/AssemblyTutorial/Chapter-17/ass17_5.html

For beq instruction, the address range is just 16-bit. Therefore, it is also impossible.

2.26.1 20

2.26.2

```
while (i>0)
{
    i = i - 1;
    B += 2;
}
```

2.26.3

For any value of N > 0, (N full loops + 2 lines) of MIPS instructions must be executed; each full loop contains 5 MIPS instructions. Therefore, if the register $t1 is initialized to the value N, a total of (5*N)+2 MIPS instructions are executed. The additional 2 instruction is because of the need to exit the loop

2.27

```
    add  $t0, $0, $0        # i = 0
L1: slt  $t2, $t0, $s0      # i < a
    beq  $t2, $0, Exit      # $t2 == 0, go to Exit
    add  $t1, $0, $0        # j = 0
```

```
L2:  slt   $t2, $t1, $s1        # j < b
     beq   $t2, $0, L3          # if $t2 == 0, go to L3
     add   $t2, $t0, $t1        # i+j
     sll   $t4, $t1, 4          # $t4 = 4*j    (sll was written instead of mul.)
     add   $t3, $t4, $s2        # $t3 = &D[4*j]
     sw    $t2, 0($t3)          # D[4*j] = i+j
     addi  $t1, $t1, 1          # j = j+1
     j     L2
L3:  addi  $t0, $t0, 1          # i = i+1
     j     L1
Exit:

fib:
     addi  $sp, $sp, -12        # allocate stack frame of 12 bytes
     sw    $a0, 8($sp)          # save n
     sw    $ra, 4($sp)          # save return address
     sw    $s0, 0($sp)          # save $s0

     slti  $t0, $a0, 2          # fib(i) = i for i = 0, 1
     beq   $t0, $0, else
     add   $v0, $a0, $0         # $v0 = 0 or 1
     j     exit            # go to exit
else:
     addi  $a0, $a0, -1         # fib(n-1)
     jal   fib             # recursive call
     add   $s0, $v0, $0
     addi  $a0, $a0, -1         # fib(n-2)
     jal   fib             # recursive call
     add   $v0, $v0, $s0
exit:
     lw    $a0, 8($sp)          # restore $a0
     lw    $ra, 4($sp)          # restore return address
     lw    $s0, 0($sp)          # restore $s0
     addi  $sp, $sp, 12         # free stack frame
     jr    $ra             # return to caller
```

2.39 Generally, all solutions are similar:

lui $t1, top_16_bits

ori $t1, $t1, bottom_16_bits

Therefore, the answer is

lui $t1, 0x2001

ori $t1, $t1, 0x4924

2.40

Address in Exercise 2.39 is 20014924_{16}

No, jump can go up to 0x0FFFFFFC.

2.41

Address in Exercise 2.39 is 20014924_{16}

No, range is 0x604 + 0x1FFFC = 0x0002 0600 to 0x604 – 0x20000
= 0×FFFE 0604.

2.42

Address in Exercise 2.39 is 20014924_{16}

Yes, range is 0x1FFFF004 + 0x1FFFC = 0x2001F000 to 0x1FFFF004 - 0x20000 = 1FFDF004