2.6

2.6.1
```
temp = Array[0];
temp2 = Array[1];
Array[0] = Array[4];
Array[1] = temp;
Array[4] = Array[3];
Array[3] = temp2;
```

2.6.2
```
lw $t0, 0($s6)
lw $t1, 4($s6)
lw $t2, 16($s6)
sw $t2, 0($s6)
sw $t0, 4($s6)
lw $t0, 12($s6)
sw $t0, 16($s6)
sw $t1, 12($s6)
```

\

2.13

2.13.1 $128 + \times > 2^{31}-1$, $x > 2^{31}-129$ and $128 + x < -2^{31}$, $x < -2^{31} - 128$ (impossible)

2.13.2 $128 - x > 2^{31}-1$, $x < -2^{31}+129$ and $128 - x < -2^{31}$, $x > 2^{31} + 128$ (impossible)

2.13.3 $x - 128 < -2^{31}$, $x < -2^{31} + 128$ and $x - 128 > 2^{31} - 1$, $x > 2^{31} + 127$ (impossible)

2.18

2.18.1 opcode would be 8 bits, `rs`, `rt`, `rd` fields would be 7 bits each

2.18.2 opcode would be 8 bits, `rs` and `rt` fields would be 7 bits each

2.18.3 more registers → more bits per instruction → could increase code size

more registers → less register spills → less instructions

more instructions → more appropriate instruction → decrease code size

more instructions → larger opcodes → larger code size

2.20

```
srl $t0, $t0, 11
sll $t0, $t0, 26
ori $t2, $0, 0x03ff
sll $t2, $t2, 16
ori $t2, $t2, 0xffff
and $t1, $t1, $t2
or  $t1, $t1, $t0
```

2.27

```
        addi $t0, $0, 0
        beq  $0,  $0, TEST1
LOOP1:  addi $t1, $0, 0
        beq  $0,  $0, TEST2
LOOP2:  add  $t3, $t0, $t1
        sll  $t2, $t1, 4
        add  $t2, $t2, $s2
        sw   $t3, ($t2)
        addi $t1, $t1, 1
TEST2:  slt  $t2, $t1, $s1
        bne  $t2, $0, LOOP2
        addi $t0, $t0, 1
TEST1:  slt  $t2, $t0, $s0
        bne  $t2, $0, LOOP1
```

2.31

```
fib:   addi $sp, $sp, -12       # make room on stack
       sw   $ra, 8($sp)         # push $ra
       sw   $s0, 4($sp)         # push $s0
       sw   $a0, 0($sp)         # push $a0 (N)
       bgt  $a0, $0, test2      # if n>0, test if n=1
       add  $v0, $0, $0         # else fib(0) = 0
       j rtn                    #
test2: addi $t0, $0, 1          #
       bne  $t0, $a0, gen       # if n>1, gen
       add  $v0, $0, $t0        # else fib(1) = 1
       j rtn
gen:   subi $a0, $a0,1          # n-1
       jal  fib                 # call fib(n-1)
       add  $s0, $v0, $0        # copy fib(n-1)
       sub  $a0, $a0,1          # n-2
       jal  fib                 # call fib(n-2)
       add  $v0, $v0, $s0       # fib(n-1)+fib(n-2)
rtn:   lw   $a0, 0($sp)         # pop $a0
       lw   $s0, 4($sp)         # pop $s0
       lw   $ra, 8($sp)         # pop $ra
       addi $sp, $sp, 12        # restore sp
       jr   $ra

# fib(0) = 12 instructions, fib(1) = 14 instructions,
# fib(N) = 26 + 18N instructions for N >=2
```

2.39 Generally, all solutions are similar:

```
lui $t1, top_16_bits
ori $t1, $t1, bottom_16_bits
```

2.40 No, jump can go up to 0x0FFFFFFC.

2.43

```
trylk: li   $t1,1
       ll   $t0,0($a0)
       bnez $t0,trylk
       sc   $t1,0($a0)
       beqz $t1,trylk
       lw   $t2,0($a1)
       slt  $t3,$t2,$a2
       bnez $t3,skip
       sw   $a2,0($a1)
skip:  sw   $0,0($a0)
```

2.44

```
try:  ll   $t0,0($a1)
      slt  $t1,$t0,$a2
      bnez $t1,skip
      mov  $t0,$a2
      sc   $t0,0($a1)
      beqz $t0,try
skip:
```