```
main:
 li $a0, 2000                  # a=2000
 li $a1, 4                     # b=4
 li $a2, 5                     # c=5
 li $a3, 6                     # d=6
 jal evalY
 move $a0, $v0                   # put the number to print ($v0) in $a0
 li $v0, 1                     # syscall 1 (print_int)
 syscall
 jr $ra

evalX:                           # x = 16c * ((16b ^ d) + 8b)/16
 sll $t0, $a1, 4             # $t0 = 16b
 xor $t0, $t0, $a3           # $t0 = 16b^d
 sll $t1, $a1, 3             # $t1 = 8b
 add $t0, $t0, $t1            # $t0 = (16b ^ d) + 8b
 sll $t1, $a2, 4             # $t1 = 16c
 mult $t1, $t0                 # do multiply 16c * ((16b ^ d) + 8b)
 mflo $t0                        # put the result back in $t0 = 16c * ((16b ^ d) + 8b)
 srl $v0, $t0, 4             # $v0 = 16c * ((16b ^ d) + 8b) / 16
 jr $ra                          # Return the result in $v0

evalY:                           # y = (14615 + -b) / ((a / 1233) * a)
 li $t1, 14615                 # $t1 = 14615
```

```
  sub $t0, $zero, $a1       # $t0 = -b

  add $t1, $t1, $t0         # $t1 = 14615 + -b

  li $t2, 1233              # $t2 = 1233

  div $a0, $t2              # do divide a / 1233

  mflo $t0                  # put the result back in $t0 = a / 1233

  mult $t0, $a0             # do multiply ((a / 1233) * a)

  mflo $t0                  # put the result back in $t0 = ((a / 1233) * a)

  div $t1, $t0              # do divide (14615 + -b) / ((a / 1233) * a)

  mflo $v0                  # $v0 = (14615 + -b) / ((a / 1233) * a)

  jr $ra                    # Return the result in $v0


# the format of block should be x = 16c * ((16b ^ d) + 8b)/16, y = (14615 + -b) / ((a /
1233) * a)

#                                                 c         b    d       b
b        a              a

proc:

  sub $sp, $sp, 4           # preserve value of $s0... $s5, $ra in stack

  sw $s0, 0($sp)

  sub $sp, $sp, 4

  sw $s1, 0($sp)

  sub $sp, $sp, 4

  sw $s2, 0($sp)

  sub $sp, $sp, 4

  sw $s3, 0($sp)

  sub $sp, $sp, 4
  sw $s4, 0($sp)

  sub $sp, $sp, 4

  sw $s5, 0($sp)

  sub $sp, $sp, 4

  sw $ra, 0($sp)

  move $s0, $a0                # $a0: number of groups to process => move to $s0

  move $s1, $a1                # $a1: Starting address of the input data => move to $s1

  move $s2, $a2                # $a2: Starting address for the output data    => move to $s2

doloop:

  lw $a2 0($s1)               # first one is c

  addiu $s1, $s1, 4

  lw $a1 0($s1)               # second one is b

  addiu $s1, $s1, 4

  lw $a3 0($s1)               # third one is d

  jal evalX                   # call evalX and get the result X in $v0

  move $s3, $v0                 # store X in variable $s3

  addiu $s1, $s1, 8

  lw $a1 0($s1)               # fifth one is b

  addiu $s1, $s1, 4

  lw $a0 0($s1)               # sixth one is a

  jal evalY                   # call evalY and get the result Y in $v0

  move $s4, $v0                 # store Y in variable $s4

  move $a0, $s3                 # $a0=X
  move $a1, $s4                # $a1=Y
  jal numb1                    # call numb1 and get result in $v0
  move $s5, $v0                # store result of numb1 in $s5
  move $a0, $s3                # $a0=X
  move $a1, $s4                # $a1=Y
  jal numb2                    # call numb2 and get result in $v0
  add $s5, $s5, $v0          # add result of numb2 to $s5 (note that the result is only
valid in lowest byte)
  sb $s5, 0($s2)             # store lowest byte in address specified in $s2
  addiu $s1, $s1, 8          # skip seventh and go to start of next group
  addiu $s2, $s2, 1          # go to next byte to store
  addiu $s0, $s0, -1         # decrease remaining group count
  bne $zero, $s0, doloop
  lw $ra, 0($sp)             # get $s0… $s5, $ra value back from stack
  addiu $sp, $sp, 4
  lw $s5, 0($sp)
  addiu $sp, $sp, 4
  lw $s4, 0($sp)
  addiu $sp, $sp, 4
  lw $s3, 0($sp)
  addiu $sp, $sp, 4
  lw $s2, 0($sp)
  addiu $sp, $sp, 4
  lw $s1, 0($sp)
```

```
addiu $sp, $sp, 4

lw $s0, 0($sp)

addiu $sp, $sp, 4

jr $ra
```