```
; FindMax(Square, N, M)
; Input Parameters
;   $a0: Square is the address of first element of 2D matrix
;   $a1: N is the number of rows in Square
;   $a2: M is the number of columns in Square
; Return Value:
;   $v0: value of maximum element in Square
;
0x1FFF FFB0 FindMax:    li      $v0, -1              # max <-- -1
0x1FFF FFB4             move    $t0, $zero           # i <-- 0
0x1FFF FFB8 NextRow:    slt     $t7, $a1, $t0        # if N<i then $t7 <-- 1
0x1FFF FFBC             bne     $t7, $zero, Return   # if i>=N Return
0x1FFF FFC0             move    $t5, $a0             # p <-- Square
0x1FFF FFC4             move    $t1, $zero           # j <-- 0
0x1FFF FFC8 NextColumn: slt     $t7, $a2, $t1        # if M<j then $t7 <-- 1
0x1FFF FFCC             bne     $t7, $zero, RowDone  # if j>=M RowDone
0x1FFF FFD0             mul     $t3, $t0, $a1        # $t3 <-- i*N
0x1FFF FFD4             add     $t4, $t3, $t1        # $t4 <-- i*N+j
0x1FFF FFD8             sll     $t5, $t4, 2          # $t5 <-- 4*(i*N+j)
0x1FFF FFDC             lw      $t6, 0($t5)          # $t6 <-- Square[i][j]
0x1FFF FFE0             slt     $t7, $v0, $t6        # if(max < Square[i][j]) then $t7 <-- 1
0x1FFF FFE4             beq     $t7, $zero NoChange
0x1FFF FFE8             move    $v0, $t6             # max <-- Square[i][j]
0x1FFF FFEC NoChange:   addi    $t1, $t1, 1          # j <-- j+1
0x1FFF FFF0             j NextColumn
0x1FFF FFF4 RowDone:    addi    $t0, $t0, 1          # i <-- i+1
0x1FFF FFF8             j NextRow                    # if i != N goto NextRow
0x1FFF FFFC Return:     jr      $ra
```

Figure 1: MIPS Assembly code for `FindMax` procedure.

In this part of the exam we will study the MIPS assembly code for the `FindMax` procedure shown in Figure 1.

Question 1 (20 points): In this question we will explore the binary representation of instructions that appear in the code for the `FindMax` procedure shown in Figure 1. Here is some review of relevant information that we know about branches and jump instructions in MIPS:

Branches The binary representation of the Opcode for a `bne` instruction in MIPS is `000101`, `$t7` is register 15, and `$zero` is register 0. The binary format of a branch instruction in MIPS, from the most-significative to the least-significative bit, starts with the Opcode, followed by the specification of the two registers that are compared by the instruction in the same order that they appear in the assembly instruction, followed by a 16-bit address field. To compute the address of the target instruction, the MIPS branch instruction shifts this 16-bit address field to the left by two bits, sign extends it to 32 bits, and then adds to the value of `PC+4`, where `PC` is the memory address of the branch instruction.

Jumps The binary representation of the Opcode for a jump instruction in MIPS is `000010`. The binary format of a jump instruction in MIPS, from the most-significative to the least-significative bit, starts with the Opcode, followed by a 26-bit address field. To compute the address of the target of the jump, the MIPS jump instruction concatenates the four most-significant bits of `PC+4`, where `PC` is the memory address of the jump instruction, with the 26-bit address field of the instruction, and then shifts the result to the left by two.

a. (**10 points**) What is the **hexadecimal representation** of the branch instruction that appears at address `0x1FFF FFCC` in the code of the `FindMax` procedure shown in Figure 1?

The 16 most significant bits are given by the opcode and the number of the registers:

```
000101 01111 00000 = 0001 0101 1110 0000 = 0x15E0
```

The 16 least significant bits must be calculated based on the address of the branch instruction and on the address of the target instruction. The 6 most-significant hexadecimal digits of the branch instruction and the target instruction are identical. Therefore we can work only with the two least-significant hexadecimal digits:

```
  0x F4   =   1111 0100
- 0x D0   =   1101 0000
```

The easiest way to subtract in binary is to compute the negative of the second operand and then convert the subtraction to an addition. This is equivalent to adding the complement of the second operand and then adding 1:

```
   1111 0100
+  0010 1111
+          1
-------------
   0010 0100
```

Now we need to shift this value to the right by two and represent it in 16 bits:

```
0000 0000 0000 1001 = 0x0009
```

Therefore the hexadecimal representation of that branch instruction is:

`0x 15E0 0009`

Another, much simpler, way to find out the value that goes in the offset field of the branch instruction is to count the number of instructions between `PC+4` and the target and figure out that we need to add 9×4 to `PS+4`.

b. (**10 points**) What is the **hexadecimal representation** of the jump instruction that appears at address `0x1FFF FFF0` in the code of the `FindMax` procedure shown in Figure 1?

All we have to do is to take the target address:
`0x1FFF FFC8 = 0001 1111 1111 1111 1111 1111 1100 1000`
Shift it right by two:
`0000 0111 1111 1111 1111 1111 1111 0010`
Then concatenate the Opcode of the jump instruction with the 26 lowest significant bits of the number above:

`0000 1011 1111 1111 1111 1111 1111 0010` = 0x 0BFF FFF2