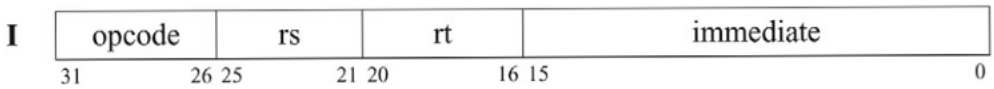
Question 1 (14 points):

```
25 # lumiptr:
26 # parameters:
27 #    $a0: screen address
28 #    $a1: R (number of rows)
29 #    $a2: C (number of columns)
30 lumiptr:
31     mul  $t1, $a1, $a2    # $t1 <- R*C
32     add  $t1, $a0, $t1    # $t1 <- screen + R*C
33     add  $v0, $0, $0      # luminosity <- 0
34 next_p:
35     lbu  $t2, 0($a0)      # $t2 <- pixel
36     add  $v0, $v0, $t2    # lumens <- lumens + pixel
37     addi $a0,$a0, 1       # p++
38     bne  $a0, $t1, next_p
39     jr   $ra
```

Figure 1: The code for `lumiptr`.

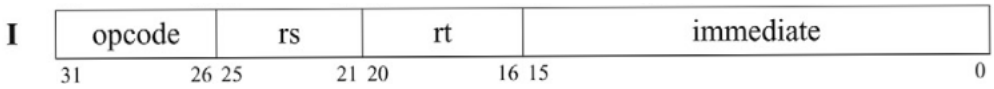
An instruction `bne rs, rt, label` uses the I-format:



1. (**4 points**) Assume that the register `$s1` contains the memory address of the first instruction of the function `lumiptr` shown in Figure 1 (MIPS programs are stored in memory in their binary representation). Write the shortest sequence of MIPS instructions that loads the binary representation of the instruction `bne` that appears in line 38 into `$a0`.

```
lw $a0, 24($s1)
```

2. (**5 points**) Assume that register `$a0` contains the binary representation of an `lbu` instruction. Write the minimum sequence of MIPS instructions that produces a value in `$v0` such that only the bit corresponding to the number of the register `rt` of the `lbu` instruction is 1. All other bits of `$v0` must be zero. An instruction `lbu rt, offset(rs)` uses the I-format:




```
sll   $a0, $a0, 11
srl   $a0, $a0, 27    # $a0 <-- rt
addi  $v0, $v0, 1     # $v1 <-- 1
sllv  $v0, $v0, $a0   # $v0 <-- bit for rt is 1
```

3. (**5 points**) The `bne rs, rt, label` instruction also follows the I-type format. Assume that register `$a0` contains the binary representation of a `bne` instruction and that register `$a1`

contains the value that was in the PC when that instruction was fetched. Write the shortest sequence of MIPS instructions that places in $v0 the address of the branch target. The branch target is the instruction that is executed when the branch is taken.

```
addi  $v0, $a1, 4     # $v0 <-- PC + 4
sll   $t1, $a0, 16    # $t1 <-- offset << 16
sra   $t1, $t1, 14    # $t1 <-- sign-extended(offset << 2)
addi  $v0, $v0, $t1   # $v0 < PC + 4 + sign-extended(offset << 2)
```