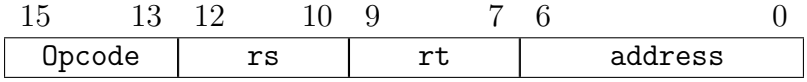
►Solution◄

Question 1: (20 points)

You are participating in the Computing Science Industrial Internship Program and your placement is with *Tiny Inc.*, a company that produces *TinyProc*— a new processor developed for the automobile industry. All instructions in *TinyProc* have 16 bits. *TinyProc* also works with 16-bit addresses. The format of a branch instruction in *TinyProc* is as shown below:



Where `rs` and `rt` specify the source and target registers for the branch instruction, respectively. The address of the target of a branch instruction is computed using the same mechanism used in the MIPS processor, but the increment of the PC and the shift left have to be adjusted for a 16-bit address machine: first the Program Counter (PC) is incremented by **two**, then the bitfield `address` of the branch instruction is shifted left **by one**, sign-extended to sixteen bits, and added to the incremented PC.

There are two branch instructions in the Instruction Set Architecture of *TinyProc*. The opcode for `beq` is `010` and the opcode for `blt` is `011`. When writing the MIPS assembly code below, you cannot use pseudo-instructions that use constants that are larger than 16 bits.

a. (10 points) Write, in MIPS assembly, a subroutine called `IsBranch` that receives in `$a0` a memory address. If the *TinyProc* instruction at that address is a branch, then `IsBranch` returns `$v0 = 1`, otherwise `IsBranch` returns `$v0 = 0`. Obey all the MIPS calling conventions.

Solution:

```
# IsBranch receives a memory address in $a0 and determines if the TinyProc
# instruction at that address is a branch.
#
# Parameters:    $a0: memory address of a TinyProc instruction
#
# Return value:  $v0: 1 if the instruction is a branch, 0 otherwise

IsBranch:
        add     $v0, $zero, $zero       # $v0 <-- 0
        lhu     $t0, 0($a0)             # $t0 <-- TinyProc Instruction
        andi    $t1, $t0, 0xE000        # $t1 <-- Opcode of Instruction
        li      $t2, 0x4000             # $t2 <-- 0x4000
        beq     $t1, $t2, BranchTrue    # If Ocpcode == 010 goto BranchTrue
        li      $t3, 0x6000             # $t3 <-- 0x6000
        beq     $t1, $t3, BranchTrue    # If Opcode == 011 goto BranchTrue
        jr      $ra                     # return $v0 = 0
BranchTrue:
        addi    $v0, $v0, 1             # return $v1 = 1
        jr      $ra

# Alternative implementations for IsBranch (thanks to students)

IsBranch:
        add     $v0, $zero, $zero       # $v0 <-- 0
        lhu     $t0, 0($a0)             # $t0 <-- TinyProc Instruction
        srl     $t1, $t0, 13            # $t1 <-- Opcode of Instruction
        li      $t2, 2                  # $t2 <-- 0000 0000 0000 0010
        beq     $t1, $t2, BranchTrue    # If Ocpcode == 010 goto BranchTrue
        li      $t3, 3                  # $t3 <-- 0000 0000 0000 0011
        beq     $t1, $t3, BranchTrue    # If Opcode == 011 goto BranchTrue
        jr      $ra                     # return $v0 = 0
BranchTrue:
        addi    $v0, $v0, 1             # return $v1 = 1
        jr      $ra

IsBranch:
        add     $v0, $zero, $zero       # $v0 <-- 0
        lhu     $t0, 0($a0)             # $t0 <-- TinyProc Instruction
        srl     $t0, $t0, 14            # $t1 <-- Two MSB of Opcode of Instruction
        li      $t2, 1                  # $t2 <-- 0000 0000 0000 0001
        beq     $t1, $t2, BranchTrue    # If two MSB of Opcode == 01 goto BranchTrue
        jr      $ra                     # return $v0 = 0
BranchTrue:
        addi    $v0, $v0, 1             # return $v1 = 1
        jr      $ra
```

b. (10 points) Write, in MIPS assembly, a subroutine called `CountBranches` that receives the address of the first instruction in a *TinyProc* program in `$a0` and returns in `$v0` the number of branches found in the program. The instructions of this *TinyProc* are stored continuously in memory and the end of the program is signalled by a half word containing `0xFFFF`. `CountBranches` must call `IsBranch` to identify if an individual instruction is a branch. It must follow all the MIPS calling conventions.

Solution:

```
# CountBranches receives in $a0 the memory address of the first instruction of
# a TinyProc program and uses IsBranch to count the number of branches found
# in the program. A halfword containing 0xFFFF signals the end of the program.
#
# Parameters:    $a0: memory address of first instruction in a TinyProc program
#
# Return value:  $v0: number of branches in the TinyProc program
#

CountBranches:
        addi    $sp, $sp, -16           # room to save $ra, $s0, $s1, $s2
        sw      $ra, 0($sp)             # save $ra
        sw      $ra, 4($sp)             # save $s0 --- persistent copy of $a0
        sw      $s1, 8($sp)             # save $s1 --- termination flag
        sw      $s2, 12($sp)            # save $s2 --- BranchCounter
        add     $s0, $a0, $zero         # $s0 <-- InstrAddress
        li      $s1, 0xFFFF             # $s1 <-- 0xFFFF
        add     $s2, $zero, $zero       # BranchCounter <-- 0
NextInstr:
        lhu     $t0, 0($s0)             # $t0 <-- TinyProc Instruction
        beq     $t0, $s1, Finished      # if Instruction == 0xFFFF goto Finished
        add     $a0, $s0, $zero         # $a0 <-- instruction address
        jal     IsBranch
        beq     $v0, $zero, NotBranch   # if IsBranch(Instruction) == 0 goto NotBranch
        addi    $s2, $s2, 1             # BranchCounter <-- BranchCounter+1
NotBranch:
        addi    $s0, $s0, 2             # InstrAddress <-- InstrAddress+2
        j       NextInstr
Finished:
        add     $v0, $s2, $s2           # counter <-- $s2
        lw      $ra, 0($sp)             # restore $ra
        lw      $s0, 4($sp)             # restore $s0
        lw      $s1, 8($sp)             # restore $s1
        lw      $s2, 12($sp)            # restore $s2
        addi    $sp, $sp, 16            # move $sp to original position
        jr      $ra
```