Hints for Project 3

Note: Suggest to read both Project 3 **documentation** and hints.

Rules for Array

```
VarDecl: IDENTIFIER LBRACKET INTCON RBRACKET{
          //This is an array declaration.
          // Allocate the memory space for the array by assign the offset
          // Keep its offset in the symbol table
          // Pay attention to the offset computation/update
 }

Variable: IDENTIFIER LBRACKET Expr RBRACKET{
          //Generate instructions to load the array element to the register
          //How to calculate its address? (base + offset)
          // base comes from IDENTIFIER, offset comes from offset

}
```

Translation of if statement

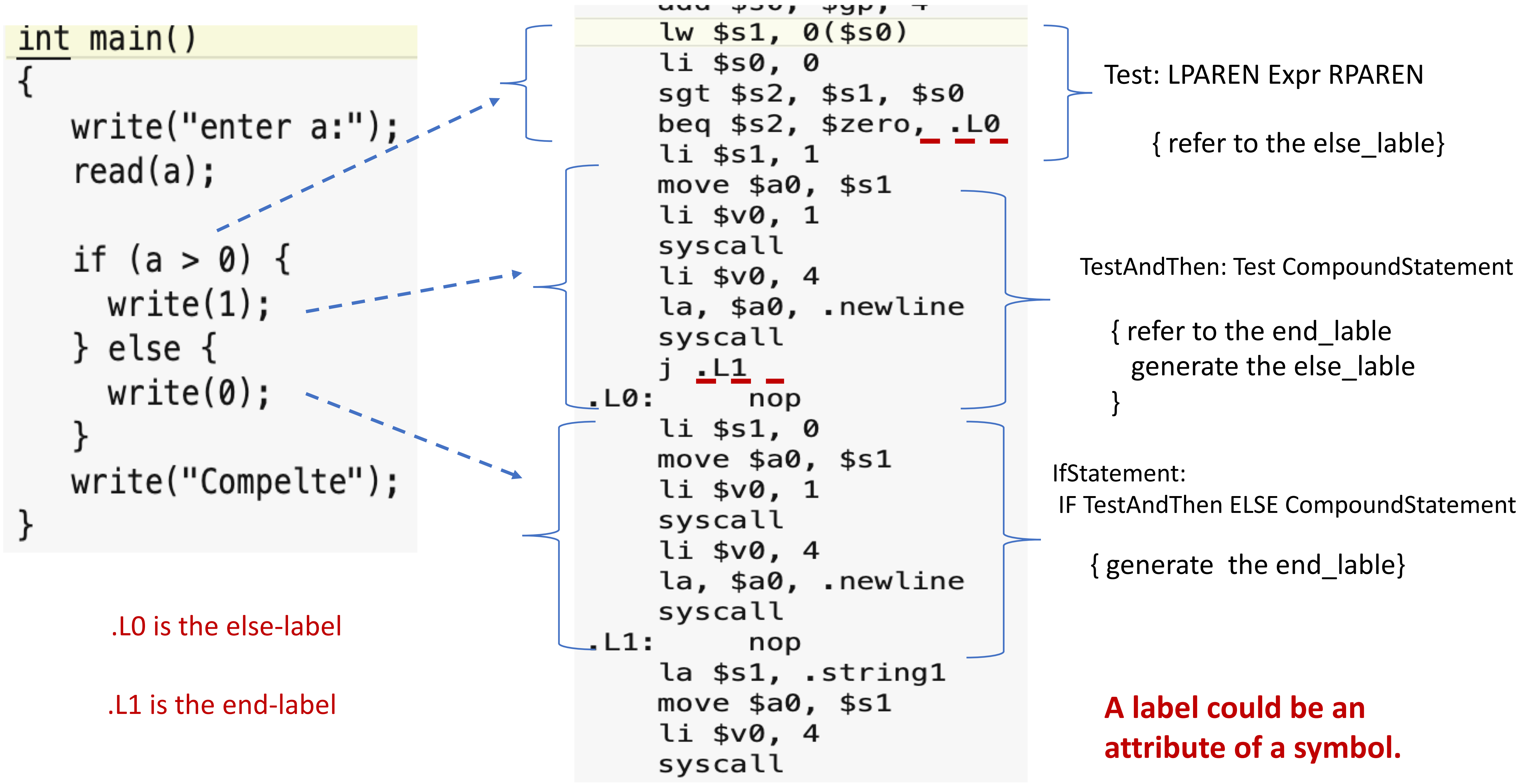
```c
int a;

int main() {

  write("enter a:");
  read(a);
  if (a != 0) {
    write(1);
  }
  write("complete!");
}
```

```
        lw $s1, 0($s0)
        li $s0, 0
        sne $s2, $s1, $s0
        beq $s2, $zero, .L0
        li $s1, 1
        move $a0, $s1
        li $v0, 1
        syscall
        li $v0, 4
        la, $a0, .newline
        syscall
        j .L1
.L0:        nop
.L1:        nop
        la $s1, .string1
        move $a0, $s1
        li $v0, 4
        syscall
```

What is the purpose of the label .L0?

What is the purpose of the label .L1?

Translation of if-then-else



```
int main()
{
   write("enter a:");
   read(a);

   if (a > 0) {
     write(1);
   } else {
     write(0);
   }
   write("Compelte");
}
```

```
      add $s0, $gp, 4
      lw $s1, 0($s0)
      li $s0, 0
      sgt $s2, $s1, $s0
      beq $s2, $zero, .L0
      li $s1, 1
      move $a0, $s1
      li $v0, 1
      syscall
      li $v0, 4
      la, $a0, .newline
      syscall
      j .L1
.L0:       nop
      li $s1, 0
      move $a0, $s1
      li $v0, 1
      syscall
      li $v0, 4
      la, $a0, .newline
      syscall
.L1:       nop
      la $s1, .string1
      move $a0, $s1
      li $v0, 4
      syscall
```

Test: LPAREN Expr RPAREN

{ refer to the else_lable}

TestAndThen: Test CompoundStatement

{ refer to the end_lable
  generate the else_lable
}

IfStatement:
IF TestAndThen ELSE CompoundStatement

{ generate the end_lable}

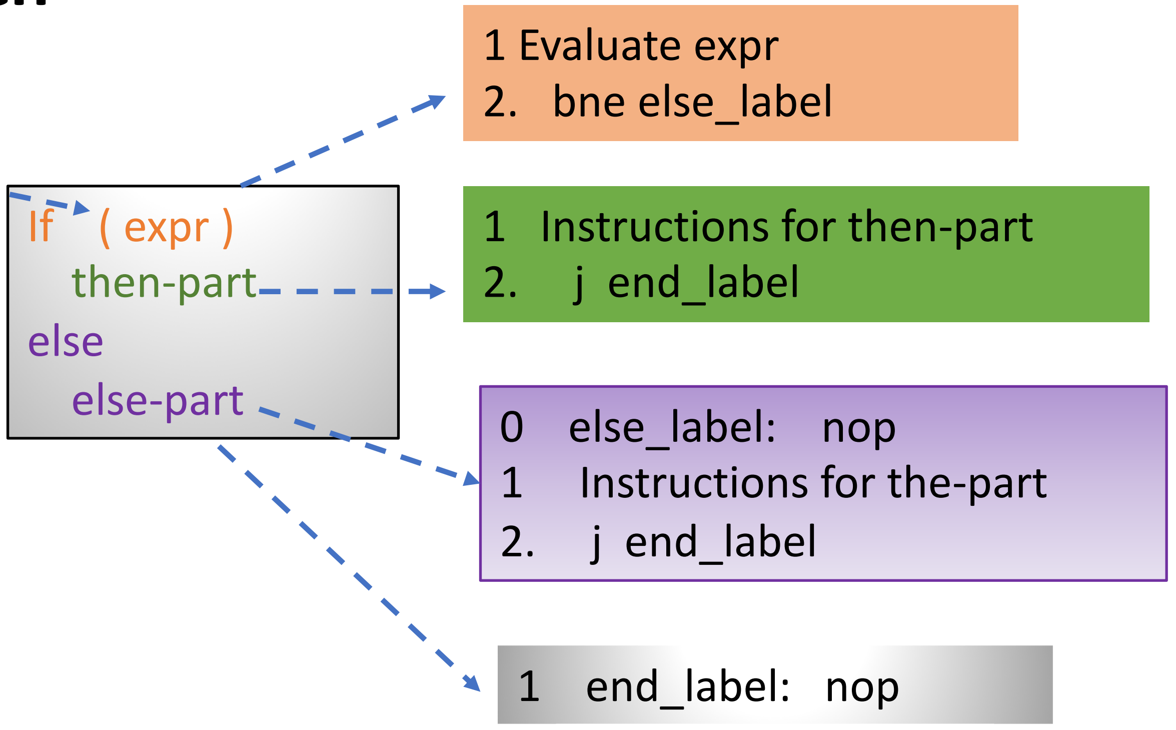
.L0 is the else-label

.L1 is the end-label

A label could be an attribute of a symbol.

Rules for Branch

```
Test: LPAREN Expr RPAREN{
            //What is  $$?
            // Hint: What if $2 is false?
}


TestAndThen       : Test CompoundStatement {
        // hint: generate  the "end"  branch
}
```



```
If   ( expr )
   then-part
else
   else-part
```

```
1 Evaluate expr
2.   bne else_label
```

```
1   Instructions for then-part
2.      j  end_label
```

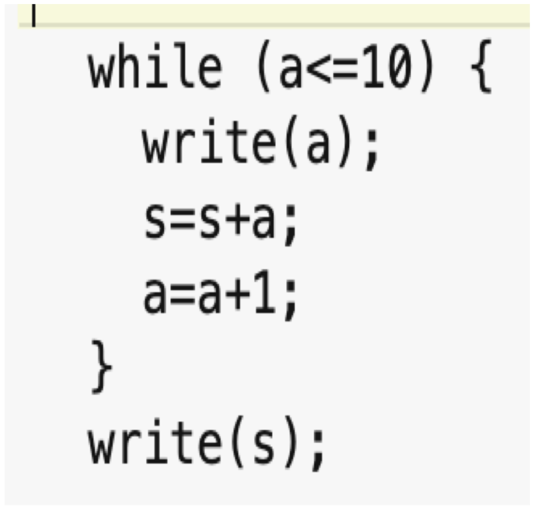
```
0    else_label:    nop
1     Instructions for the-part
2.      j  end_label
```

```
1    end_label:   nop
```

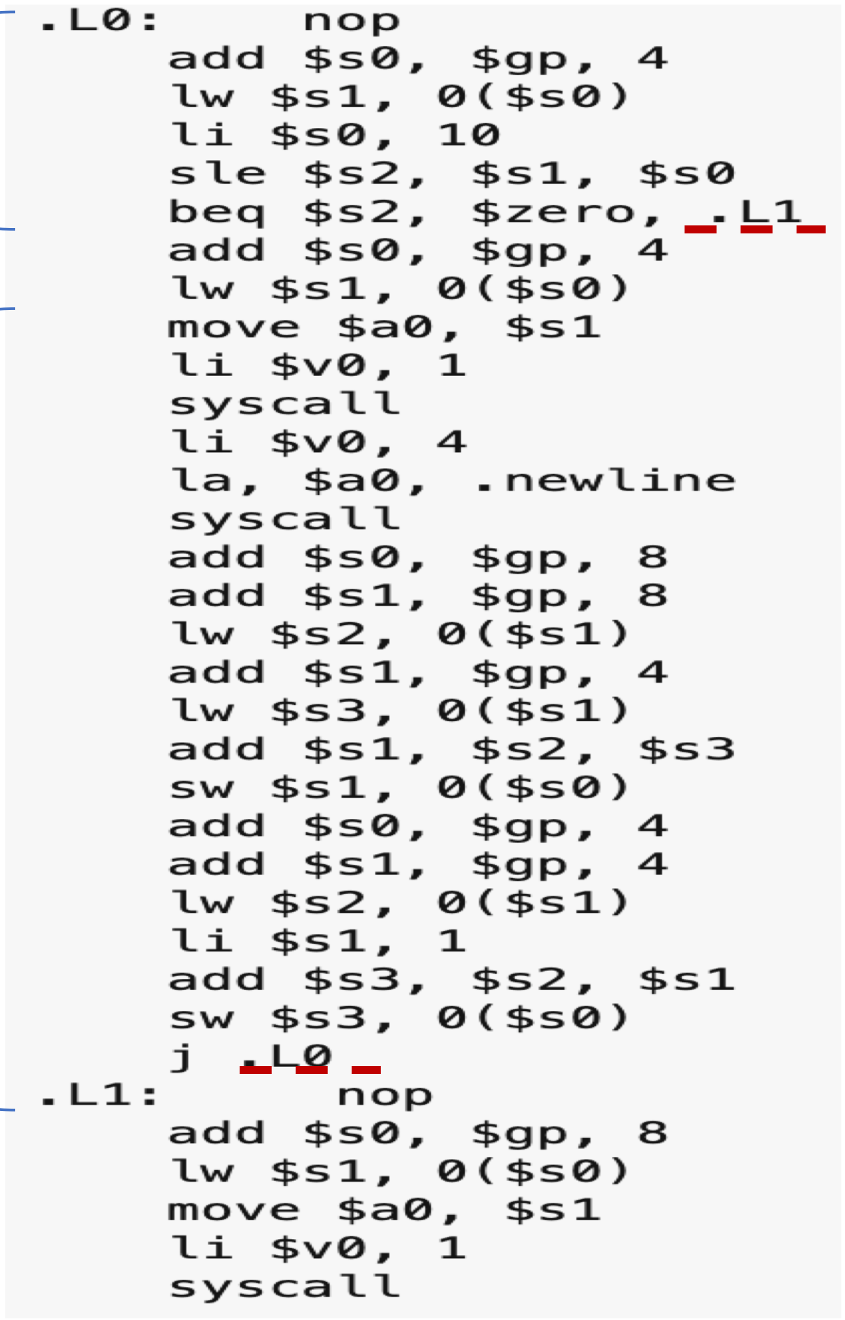
```
IfStatement: IF TestAndThen ELSE CompoundStatement{
    // At this time point, the instructions for TestAndThen and
    // CompoundStatement are generated.
   // hit: generate the "end" target
}
```

- You need to generate branch instructions
- You need to generate labels
- You can use labels before using them

Translation of While Loop



```
while (a<=10) {
  write(a);
  s=s+a;
  a=a+1;
}
write(s);
```



```
.L0:    nop
        add $s0, $gp, 4
        lw $s1, 0($s0)
        li $s0, 10
        sle $s2, $s1, $s0
        beq $s2, $zero, .L1
        add $s0, $gp, 4
        lw $s1, 0($s0)
        move $a0, $s1
        li $v0, 1
        syscall
        li $v0, 4
        la, $a0, .newline
        syscall
        add $s0, $gp, 8
        add $s1, $gp, 8
        lw $s2, 0($s1)
        add $s1, $gp, 4
        lw $s3, 0($s1)
        add $s1, $s2, $s3
        sw $s1, 0($s0)
        add $s0, $gp, 4
        add $s1, $gp, 4
        lw $s2, 0($s1)
        li $s1, 1
        add $s3, $s2, $s1
        sw $s3, 0($s0)
        j .L0
.L1:     nop
        add $s0, $gp, 8
        lw $s1, 0($s0)
        move $a0, $s1
        li $v0, 1
        syscall
```

.L0 is the start-label

Indicate the starting of an iteration

.L1 is the end-label

Indicate the end of the loop

Rules for While loop

```
WhileToken    : WHILE{
   // Indicate the beginning of an iteration
}

WhileExpr       : LPAREN Expr RPAREN{
        //  evaluate  Expr
       //  branch to  the end of the branch if false
}

WhileStatement: WhileToken WhileExpr Statement{
    // Generate the instruction to repeat the iteration
    // Generate the label for the exit/end of the loop
}
```