Compiler Technology of Programming Languages

Code Generation
Control Structures and Loops

Prof. Farn Wang

Simple If

```
if ( a )
{
    b = 1;
}
```

```
    ldr w9, [x29, #-4]
    cmp w9, #0
    beq     Lexit1
    mov     w10,    #1
    str     w10,    b
Lexit1:
    ....
```

What if a is an expression? e.g. if (a&b), (a+b)

```
cmp w9, #0      where w9 = expr.place
beq Lexit1
```

How to generate code for (a>b) ?

```
cmp w9, w10
ble  Lexit1
```

If-else

```
if ( a )
{
    b = 1;
}
 else
{
    b = 0;
}
```

```
        ldr w9, [x29, #-4]
        cmp w9, #0
        beq     Lelse1
        mov     w10,    #1
        str     w10,    b
        b       Lexit1
Lelse1:
        mov     w11,    #0
        str     w11,    b
Lexit1:
        ….
```

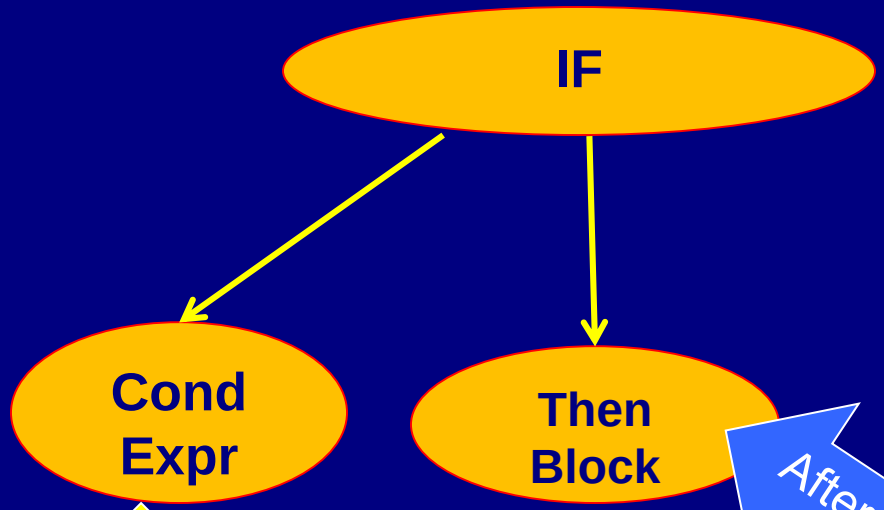
Nested If-else

```
if ( a )
{    b = 1;}
 else
{    if ( j )
        {b = 0;}
        else
        {c = 0;}
        a = 0;
}
```

```
        ldr     w9,     a
        cmp     w9, #0
        beq     Lelse1
        mov     w10,    #1
        str     w10,    b
        b       Lexit1
Lelse1:
        ldr     w11,    j
        cmp     w11,#0
        beq     Lelse2
        mov     w12,#0
        str     w12,    b
        b       Lexit2
Lelse2:
        mov     w13,#0
        str     w13,    c
Lexit2:
        mov     w9,#0
        str     w9,     a
Lexit1:
```

Code Generation for If Structures

- A test that branches to the else part (if-else) or the exit part (simple if) if the condition is false.
- Unique labels such as Lelse1, Lexit2
- Need to handle nested structures, so label number should be maintained on a stack.
- Since the code gen walks AST via recursive calls, label number can be maintained in a local variable.
- Jump to Lexit (skip over the else part) at the end of the *then* block.

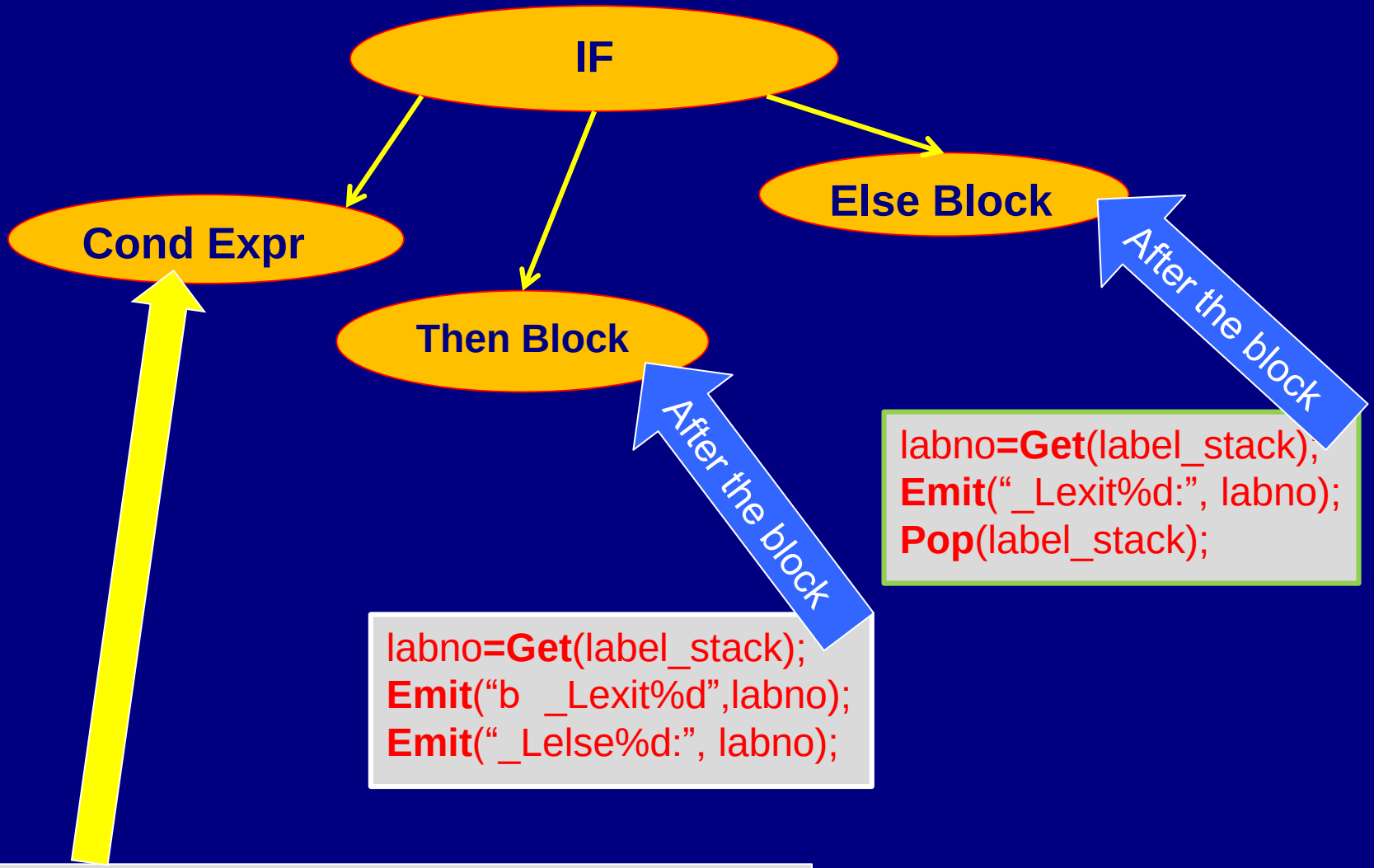


After the block

Emit(“beqz expr->place, _Lexit%d”, ++label_no);
Push(label_stack, label_no);

labno=**Get**(label_stack);
Emit(“_Lexit%d:”, labno);
Pop(label_stack);

Using cmp and b <con>


IF
Cond Expr
Then Block
Else Block
After the block
After the block
labno=Get(label_stack);
Emit("_Lexit%d:", labno);
Pop(label_stack);
labno=Get(label_stack);
Emit("b _Lexit%d",labno);
Emit("_Lelse%d:", labno);
Emit("beqz expr->place, _Lelse%d", ++label_no);
Push(label_stack, label_no);

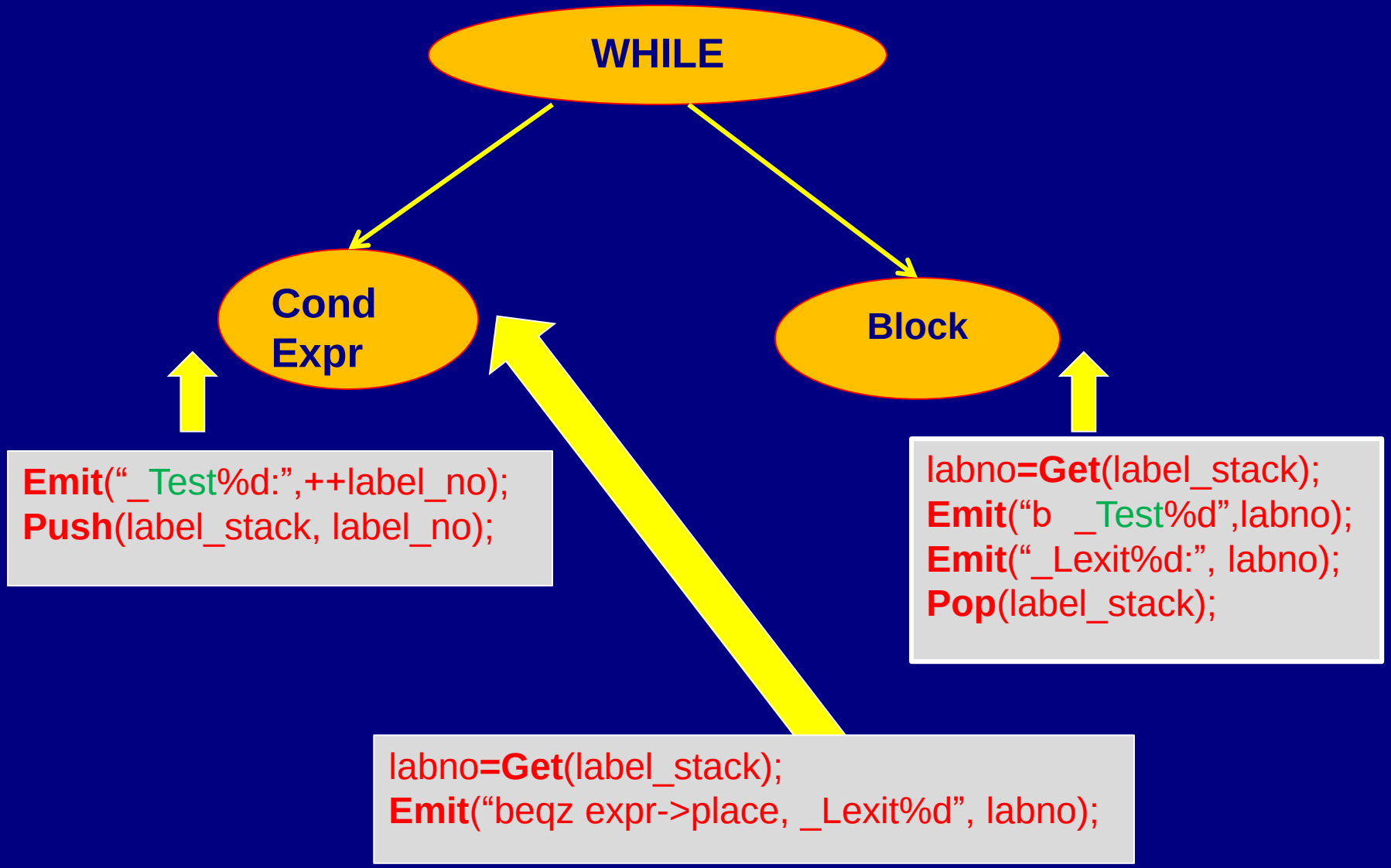
Avoid using the label stack

```
Procedure visit(AST IF n) {
    int local_label_number;
    local_label_number= ++label_no; /* keeps a local
    copy of the current label_no. If there are nested
    structures, this local label will be saved on the calling
    stack automatically.  */
     …..
     CodeGen(…);
     …
     gen_labels();
}
```

While Loop

```
while ( a )
{
    stmt;
}
```

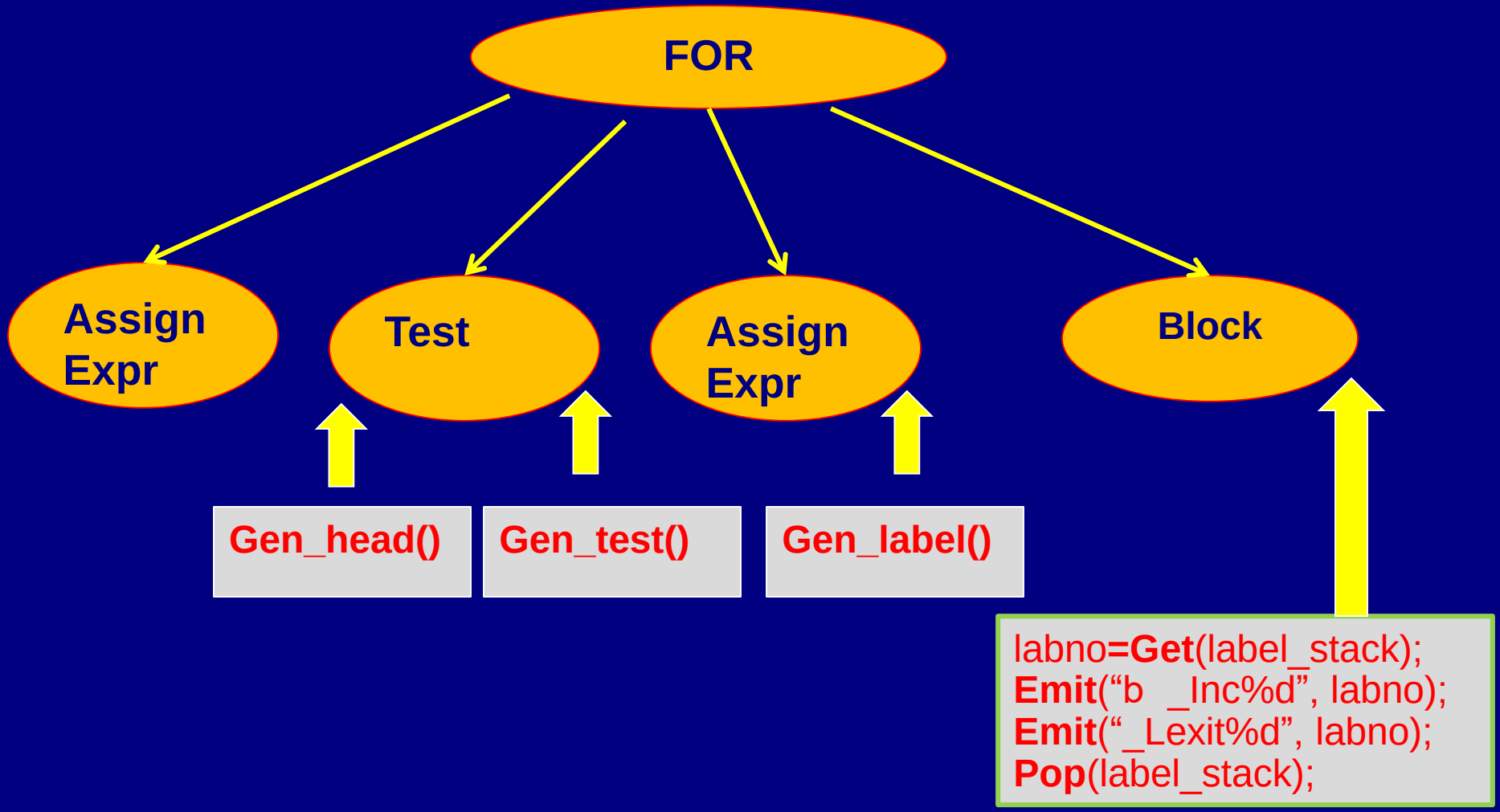
```
_Test1:
        ldr     w9,     a
        cmp     w9,     #0
        beq     _Lexit1
        …. Code for stmt …
        b       _Test1
_Lexit1:
        ….
```

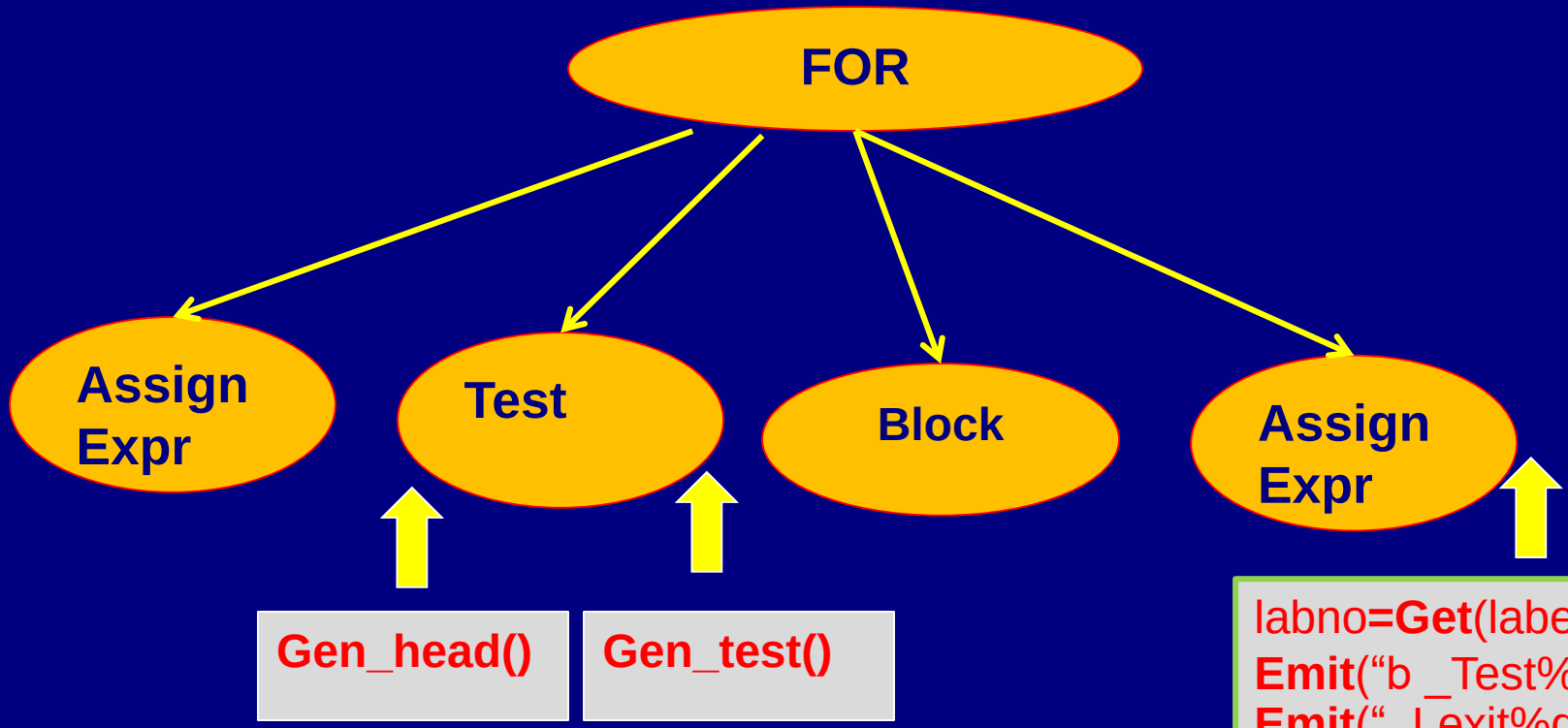

WHILE
Cond Expr
Block
Emit("_Test%d:",++label_no);
Push(label_stack, label_no);
labno=Get(label_stack);
Emit("b _Test%d",labno);
Emit("_Lexit%d:", labno);
Pop(label_stack);
labno=Get(label_stack);
Emit("beqz expr->place, _Lexit%d", labno);

For loop

```
for (j=0; j < n; j=j+1)
    stmt;
```

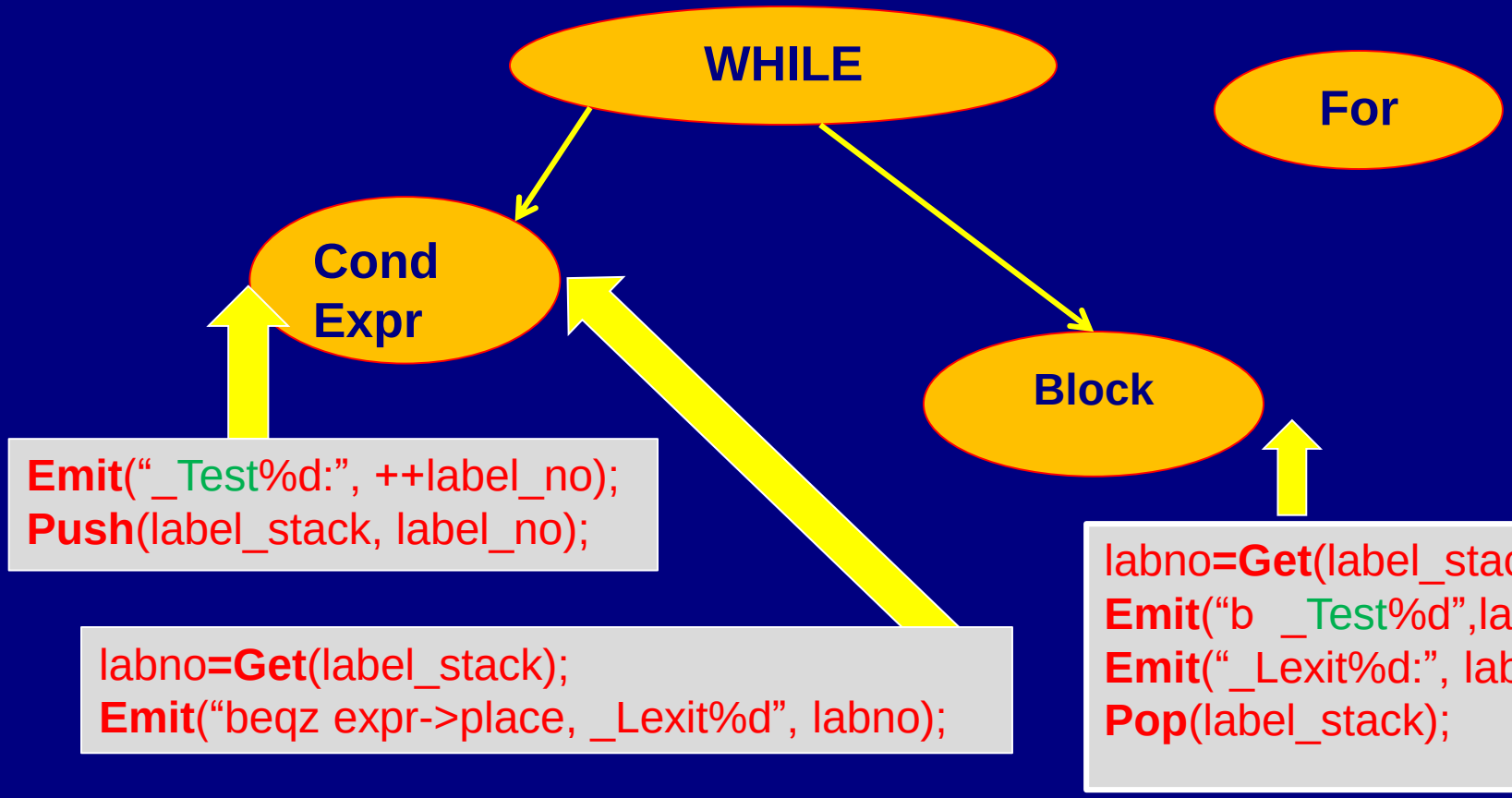
```
        mov     w9,#0
        str w9, j
_Test1:
        eval expr j<n in w10
        cmp w10,#0
        beq     _Lexit1
        b       _Body1
_Inc1:
        ldr w11,  j
        add w11,w11,#1
        str w11, j
        b       _Test1
_Body1:
        code for stmt;
        b       _Inc1
_Lexit1:
        ....
```


FOR
Assign Expr
Test
Assign Expr
Block
Gen_head()
Gen_test()
Gen_label()
labno=Get(label_stack);
Emit("b _lnc%d", labno);
Emit("_Lexit%d", labno);
Pop(label_stack);


FOR
Assign Expr
Test
Block
Assign Expr
Gen_head()
Gen_test()
labno=Get(label_stack);
Emit("b _Test%d", labno);
Emit("_Lexit%d", labno);
Pop(label_stack);

CodeGen Routines

```
gen_head
    emit("_Test%d:", ++label_no);
    push(label_stack, label_no);
gen_test
    labno=get(label_stack);
    if expr->type is integer but not integer const
          emit("beqz     expr->place, _Lexit%d", labno);
    emit("b       _Body%d", label_no);
    emit("_Inc%d:", label_no);
gen_label
    labno=get(label_stack);
    emit("b       _Test%d:", labno);
    emit("_Body%d:", labno);
```



Be careful about the label number handling since the label number is needed before the "***cond***" is evaluated.

Boolean Expressions

- (**I** && **J** && **K**)
  assuming I,J, and K are in register w9,w10,w11
  the generated code may look like:

```
    beqz    w9, F1
    beqz    w10, F1
    beqz    w11, F1
T1:
    mov     w12, #1
    b       E1
F1:
    mov     r7, #0
E1:
```

Boolean Expressions in Control Structures

- Using Jump Code Sequence

If (**I** && **J** && **K**) Then… Else …

```
    beqz        w9, Else
    beqz        w10, Else
    beqz        w11, Else
Then ….
    ….
Else
    ….
```

Boolean Expressions in Control Structures

- The expression could be an assignment expression

If (a = (**I** && **J** && **K**)) Then… Else …

For assignment expressions, we do need to generate a boolean value.

- && could mixed with ||

(a < b) && (a < 10) | | (b > 5) | | (c ==0)

Jump Code for relop_expr

if ((a < b) && (a < 10) | | (b > 5)) then X else Y

assuming a, and b are in register w9, w10

```
    bge         w9, w10, F1 // cmp and bge are folded
    bge         w9, 10, F1
    b           Then   // jump directly instead of mov w11 #0
F1:     // (a<b) && (a<10) is False, need to eval (b>5)
    ble         w10,5          ELSE
Then:
    X
ELSE:
    Y
```

Function Call

- save_registers();
- passing_param();
- gen_proc_call(name);
- gen_after_return();

Function Call

- save_registers();

  Optional – depends on whether caller save registers are used

- passing_param();

  Not required in Part I

- gen_proc_call(name);

  generate “bl name” instruction

- gen_after_return();

  generate a copy inst that moves x0 to x7 to a target reg

  remove stack space allocated for arguments

  restore saved registers if any