# Geração de Código Intermediário
## Expressões Relacionais

### Portugol

```
...
cont <= 5
...
```

### BIP

```
LD      cont
STO     temp1
LDI     5
STO     temp2
LD      temp1
SUB     temp2
BGT     rotulo
```

```
#7
oprel = token.getlexeme();
gera_cod ("STO", TEMP_ESQ);

#8
gera_cod ("STO", TEMP_DIR);
gera_cod ("LD", TEMP_ESQ);
gera_cod ("SUB", TEMP_DIR);
```

### Expressões

```
<exp1> ::= <exp1> '>' #7 <exp2> #8
         | <exp1> '<' #7 <exp2> #8
         | <exp1> '>=' #7 <exp2> #8
         | <exp1> '<=' #7 <exp2> #8
         | <exp1> '!=' #7 <exp2> #8
         | <exp1> '==' #7 <exp2> #8
         | <exp2>
<exp2> ::= <exp2> '+' #6 <exp3>
         | <exp2> '-' #6 <exp3>
         | <exp3>
<exp3> ::= ID #4
         | NUM_INT  #5
```

# Geração de Código Intermediário
## Expressões Relacionais

### Portugol

```
...
cont <= 5
...
```

### BIP

```
LD      cont
STO     1000
LDI     5
STO     1001
LD      1000
SUB     1001
BGT     rotulo
```

```
#7
oprel = token.getlexeme();
gera_cod ("STO", 1000);

#8
gera_cod ("STO", 1001);
gera_cod ("LD", 1000);
gera_cod ("SUB", 1001);
```

### Expressões

```
<exp1> ::= <exp1> '>' #7 <exp2> #8
         | <exp1> '<' #7 <exp2> #8
         | <exp1> '>=' #7 <exp2> #8
         | <exp1> '<=' #7 <exp2> #8
         | <exp1> '!=' #7 <exp2> #8
         | <exp1> '==' #7 <exp2> #8
         | <exp2>
<exp2> ::= <exp2> '+' #6 <exp3>
         | <exp2> '-' #6 <exp3>
         | <exp3>
<exp3> ::= ID #4
         | NUM_INT  #5
```

# Geração de Código Intermediário
## Desvio Condicional Simples

### Código Portugol

```
se (v1 > v2) entao
    v2 ← v1
fimse
```

### Lógica Assembly

código para **exp**

se falso salta r1

código para **lista_cmd**

r1:

### Código BIP

```
        LD      v1
        STO     temp1
        LD      v2
        STO     temp2
        LD      temp1
        SUB     temp2
        BLE     R1
        LD      v1
        STO     v2
R1:
```

### Esquema de Tradução para Geração de Código

**<cmd>:== se <exp> entao #9<lista_cmd> <r_senao> fimse #10**
**<r_senao> ::= senao #11 <lista_cmd>**

```
#9
rotIf = newRotulo();
push (rotIf);
if (oprel == ">")
  gera_cod ("BLE", rotIf);
if (oprel == "<")
  gera_cod ("BGE", rotIf);
     ...
// idem para BEQ, BNE, BGT, BGE, BLT, BLE
#10
rotFim = pop ();
gera_cod ("ROT", rotFim);
```

# Geração de Código Intermediário
## Desvio Condicional Composto

### Código Portugol

```
se (v1 > v2) entao
     v2 ← v1
senao
     v1 ← 0
fimse
```

### Lógica Assembly

código para **exp**

se falso salta r1

código para **lista_cmd**

JMP r2

r1:

código para **lista_cmd**

r2:

### Código BIP

```
        LD      v1
        STO     temp1
        LD      v2
        STO     temp2
        LD      temp1
        SUB     temp2
        BLE     R1
        LD      v1
        STO     v2
        JMP     R2
R1:
        LDI     0
        STO     v1
R2:
```

### Esquema de Tradução para Geração de Código

**<cmd>:== se <exp> entao #9<lista_cmd> <r_senao> fimse #10**
**<r_senao> ::= senao #11 <lista_cmd>**

```
#11     rotIf = pop ();
        rotFim = newRotulo();
        gera_cod ("JMP", rotFim);
        push (rotFim);
        gera_cod ("ROT", rotIf);
```

# Geração de Código Intermediário
## Laço de Repetição Enquanto

### Código Portugol

```
enquanto (cont <= 5)
    cont <- cont + 1
fimenquanto
```

### Lógica Assembly

r1:

código para **exp**

Se falso salta r2

código para **lista_cmd**

JMP r1

r2:

### Código BIP

```
R1:
        LD      cont
        STO     temp1
        LDI     5
        STO     temp2
        LD      temp1
        SUB     temp2
        BGT     R2
        LD      cont
        ADDI    1
        STO     cont
        JMP     R1
R2:
```

# Geração de Código Intermediário
## Laço de Repetição Enquanto

### Esquema de Tradução

**<cmd>::= enquanto #12<exp> #13 faca <lista_cmd> fim_enquanto #14**

### Lógica Assembly

r1:

código para **exp**

Se falso salta r2

código para **lista_cmd**

JMP r1

r2:

```
#12
rotIni = newRotulo();
push (rotIni);
gera_cod ("ROT", rotIni);

#13
rotFim = newRotulo();
push (rotFim);
if (oprel == ">")
  gera_cod ("BLE", rotFim);
if (oprel == "<")
  gera_cod ("BGE", rotFim);
     ...
// idem para BEQ, BNE, BGT, BGE, BLT, BLE
#14
rotFim = pop();
rotIni = pop();
gera_cod ("JMP", rotIni);
gera_cod ("ROT", rotFim);
```

# Geração de Código Intermediário
## Laço de Repetição Faça Enquanto

### Código Portugol

```
Faca
    cont <- cont + 1
enquanto (cont <= 5)
```

### Lógica Assembly

r1:

código para **lista_cmd**

código para **exp**

se true salta r1

### Código de M. Pilha

```
R1:
        LD      cont
        ADDI    1
        STO     cont
        LD      cont
        STO     temp1
        LDI     5
        STO     temp2
        LD      temp1
        SUB     temp2
        BLE     R1
```

OBS: Aqui o salto (**BRANCH**) para retornar ao início do LOOP obedece a lógica da expressão!

Ex. (cont <= 5) gera BLE

# Geração de Código Intermediário
## Laço de Repetição Faça Enquanto

### Esquema de Tradução

**<cmd>::= faca #15 <lista_cmd> enquanto <exp> #16**

### Lógica Assembly

r1:

código para **lista_cmd**

código para **exp**

se true salta r1

```
#15
rotIni = newRotulo();
push (rotIni);
gera_cod ("ROT", rotIni);

#16
rotIni = pop();
if (oprel == ">")
  gera_cod ("BGT", rotIni);
if (oprel == "<")
  gera_cod ("BLT", rotIni);
     ...
// idem para BEQ, BNE, BGT, BGE, BLT, BLE
```

OBS: Aqui o salto (**BRANCH**) para retornar ao início do LOOP obedece a lógica da expressão!

Ex. (cont <= 5) gera BLE

# Geração de Código Intermediário
## Laço de Repetição Para

### Código Portugol

```
para (i=1; i<10; i++){
   escreva (i)
}fimpara
```

### Lógica Assembly

inicialização
r1:
código para **exp**
se false salta r2
código para **lista_cmd**
**incr/decr**
JMP r1
r2:

### Código de M. Pilha

```
        LDI     1
        STO     i
R1:
        LD      i
        STO     temp1
        LDI     10
        STO     temp2
        LD      temp1
        SUB     temp2
        BGE     R2
        LD      i
        STO     $out_port
        LD      i
        ADDI    1
        STO     i
        JMP     R1
R2:
```