1 Actividade 0x0b - Geração de Código

Para termos um compilador completo, resta-nos implementar o gerador de código. Nesta parte temos várias opções:

- geração de código para máquina de pilha;
- geração directa de código máquina;
- geração de código numa linguagem "compilável" (e.g., **C**, **Go**, **erlang**, etc.)

Cabe ao grupo de trabalho decidir com qual das abordagens acima se sente mais confortável.

Para efeitos desta actividade prática, usaremos a abordagem utilizada na aula teórica: geração de código para máquina de pilha. Os alunos podem, no entanto, implementar o gerador de código usando outra abordagem a seu gosto.

2 Código para máquina de pilha

A nossa máquina de pilha será implementada em `mips` e goza das seguintes propriedades:

- A única zona de memória é a pilha;
- As instruções assumem que os seus argumentos estão na pilha e que o valor de retorno deve ser colocado na pilha;
- A pilha cresce para baixo;
- Usamos o registo `$t0` para guardar o que está no topo da pilha
- O topo da pilha (na memória) está no endereço `$sp+4`
- (resumindo, o topo da pilha está em `$t0`, o segundo elemento da pilha está em `$sp+4`, o terceiro elemento da pilha está em `$sp+8`, etc.)

2.1 Exemplos

2.1.1 Geração de código para um literal inteiro

```
void codegen_intlit(int i) {
  printf("li $t0, %d\n", i);
}
```

2.1.2 Geração de código para um *binop*

```
void codegen_binop(char op, t_exp e1, t_exp e2) {
  codegen_exp(e1);
  printf("sw $t0, 0($sp)\n");
  printf("addiu $sp, $sp, -4\n");  /* push */
  codegen_exp(e2);
  printf("lw $t1, 4($sp)\n");
  switch (op) {
    case '+':
      printf("add ");
      break;
    case '-':
      printf("sub ");
      break;
    case '*':
      printf("mul ");
      break;
    /* . . . */
  }
  printf("$t0, $t1, $t0\n");
  printf("addiu $sp, $sp, 4\n");   /* pop */
}
```

2.1.3 Geração de código para um condicional

```
void codegen_stm_cond_equals(t_exp e1, t_exp e2,
                             t_stms stms_true, t_stms stms_false)
{
  LABEL lbl_iftrue, lbl_iffalse, lbl_endif;

  lbl_iftrue = GEN_LABEL(. . .);
  lbl_iffalse = GEN_LABEL(. . .);
  lbl_endif = GEN_LABEL(. . .);

  codegen(e1);
  printf("sw $t0, 0($sp)\n");
  printf("addiu $sp, $sp, -4\n");
  codegen(e2);
  printf("lw $t1, 4($sp)\n");
  printf("addiu $sp, $sp, 4\n");
  printf("beq $t0, $t1, %s\n", LABEL2STR(lbl_iftrue));
  printf("%s:\n", LABEL2STR(lbl_iffalse));
  codegen(stms_false);
  printf("j %s\n", LABEL2STR(lbl_endif));
  printf("%s:\n", LABEL2STR(lbl_iftrue));
  codegen(stms_true);
  printf("%s:\n", LABEL2STR(lbl_endif));
}
```

1. (Ou podemos gerar o código da condição do *if* e testar se o valor é maior que zero, por exemplo.)
2. (Assumimos a existência das funções GEN_LABEL() e LABEL2STR(), que tratam da gestão de *labels*.)

2.1.4 Geração de código para uma chamada de função

```
void codegen_exp_funcall(t_exp_funcall fc)
{
  printf("sw $fp, 0($sp)\n");     /* salvaguardar o FP */
  printf("addiu $sp, $sp, -4\n");
  codegen(fc->arglist[n-1]);
  printf("sw $t0, 0($sp)\n");     /* colocar o resultado do arg_n
    na frame */
  printf("addiu $sp, $sp, -4\n");
  /* . . .  processar restantes argumentos . . . */
  codegen(fc->arglist[0]);
  printf("sw $t0, 0($sp)\n");
  printf("addiu $sp, $sp, -4\n");
  printf("jal %s\n", fc->label);
}
```

2.1.5 Geração de código para uma declaração de função

```
void codegen_decl_func(t_decl_func fd)
{
  printf("%s:\n", fd->label)
  printf("move $fp, $sp\n");
  printf("sw $ra, 0($sp)\n");
  printf("addiu $sp, $sp, -4\n");
  codegen(fd->bodystms);   /* executa o corpo, return value está
   em $t0 */
  printf("%s_exit:\n", fd->label);
  printf("lw $ra, 4($sp)\n");
  printf("addiu $sp, $sp, %d\n", fd->ra_size);
  printf("lw $fp, 0($sp)\n");
  printf("jr $ra\n");
}
```

3 Exercício

1. Implementar as funções para a geração de código para cada tipo de nó da APT;
   - Não esquecer o tratamento de `floats` (com registos `$f`) e `strings`.
   - Procurar implementar um sistema de geração de *labels* únicos, de forma a não existirem conflitos.
2. Chamar a função na regra inicial, executando a geração de código de todo o programa.

3. (opcional) Trocar os `printf()` por `fprintf()`, de forma a poder fazer o output do código para um ficheiro.