Báo cáo Thực hành Xây dựng chương trình dịch
Bài 5: Sinh mã đích cơ bản

Họ và tên: Lê Hải Yến
MSSV: 20225780

I. GIỚI THIỆU TỔNG QUAN

1. Giới thiệu

   Sinh mã đích là giai đoạn cuối cùng trong quá trình biên dịch, có nhiệm vụ chuyển đổi cây cú pháp trừu tượng (AST) đã được phân tích và kiểm tra ngữ nghĩa thành mã máy thực thi được. Trong bài tập này, chúng ta sinh mã cho một máy ảo sử dụng kiến trúc ngăn xếp (Stack Machine).

2. Kiến trúc máy ngăn xếp

   Máy ngăn xếp hoạt động dựa trên cấu trúc dữ liệu ngăn xếp, nơi mọi phép tính và quản lý bộ nhớ đều thực hiện thông qua việc đẩy (push) và lấy ra (pop) các giá trị từ ngăn xếp.

   Các thanh ghi quan trọng:

   - PC (Program Counter): Lưu địa chỉ của lệnh đang được thực thi
   - B (Base): Trỏ đến địa chỉ đáy của khung ngăn xếp (stack frame) hiện tại
   - T (Top): Trỏ đến đỉnh ngăn xếp, nơi lưu dữ liệu tạm thời

   Cấu trúc Stack Frame: Mỗi khi gọi một chương trình con, một stack frame mới được tạo với cấu trúc:

   - Offset 0: RV (Return Value) - Vị trí lưu giá trị trả về của hàm
   - Offset 1: DL (Dynamic Link) - Liên kết động đến frame gọi nó
   - Offset 2: RA (Return Address) - Địa chỉ quay về sau khi thực thi xong
   - Offset 3: SL (Static Link) - Liên kết tĩnh để truy cập biến ngoài scope
   - Offset 4+: Tham số và biến cục bộ

3. Tập lệnh máy ngăn xếp

   a. Nhóm lệnh quản lý dữ liệu

      - LA p,q (Load Address): Tính và đẩy địa chỉ của biến lên ngăn xếp. Tham số p là độ lệch scope (static link level), q là offset trong frame
      - LV p,q (Load Value): Tải giá trị của biến lên ngăn xếp
      - LC q (Load Constant): Đẩy hằng số q trực tiếp lên ngăn xếp

- LI (Load Indirect): Lấy giá trị tại địa chỉ được lưu ở đỉnh ngăn xếp
- ST (Store): Lưu giá trị từ đỉnh ngăn xếp vào địa chỉ nằm ngay dưới nó

b. Nhóm lệnh số học
- AD (Add): Lấy hai số trên đỉnh ngăn xếp, cộng lại và đẩy kết quả lên
- SB (Subtract): Thực hiện phép trừ
- ML (Multiply): Thực hiện phép nhân
- DV (Divide): Thực hiện phép chia
- NEG (Negate): Đảo dấu số ở đỉnh ngăn xếp

c. Nhóm lệnh so sánh
- Các lệnh EQ, NE, GT, LT, GE, LE so sánh hai giá trị trên ngăn xếp và trả về 1 (đúng) hoặc 0 (sai)

d. Nhóm lệnh điều khiển
- J q (Jump): Nhảy vô điều kiện đến địa chỉ q
- FJ q (False Jump): Nhảy đến q nếu giá trị đỉnh ngăn xếp bằng 0
- HL (Halt): Dừng chương trình

e. Nhóm lệnh quản lý ngăn xếp
- INT q (Increment): Tăng T thêm q đơn vị (cấp phát bộ nhớ)
- DCT q (Decrement): Giảm T đi q đơn vị (thu hồi bộ nhớ)
- CV (Copy Value): Sao chép giá trị ở đỉnh ngăn xếp

II. <u>CÁC HÀM SINH MÃ CHỦ YẾU</u>

1. Sinh mã cho biến
   - Hàm genVariableAddress()
     Nhiệm vụ của hàm này là tính toán và sinh lệnh LA để đưa địa chỉ của một biến lên ngăn xếp. Quá trình thực hiện:
     + Bắt đầu từ scope hiện tại, đếm số bước phải lùi ngược để đến scope chứa biến đó (gọi là nestLevel)
     + Sử dụng offset đã được tính toán trong giai đoạn phân tích ngữ nghĩa
     + Phát ra lệnh LA với hai tham số: nestLevel và offset
   - Hàm genVariableValue()
     Tương tự như hàm trên nhưng thay vì đẩy địa chỉ, hàm này đẩy trực tiếp giá trị của biến lên ngăn xếp bằng lệnh LV.
2. Sinh mã cho biểu thức số học

- Hàm compileFactor(): Xử lý các thành phần cơ bản nhất trong biểu thức:
    + Với số nguyên: sinh lệnh LC để đẩy giá trị hằng lên ngăn xếp
    + Với ký tự: sinh lệnh LC với mã ASCII của ký tự
    + Với biến: gọi genVariableValue để đẩy giá trị biến lên ngăn xếp
    + Với hằng đã khai báo: đẩy giá trị hằng đó lên ngăn xếp
    + Với biểu thức con trong ngoặc: đệ quy xử lý biểu thức bên trong
- Hàm compileTerm2(): Xử lý phép nhân và chia với ưu tiên cao:
    + Sinh mã cho toán hạng bên trái, kết quả lên ngăn xếp
    + Sinh mã cho toán hạng bên phải, kết quả lên ngăn xếp
    + Sinh lệnh ML (nhân) hoặc DV (chia) để tính toán hai giá trị trên đỉnh
    + Kết quả được đẩy lại lên đỉnh ngăn xếp
- Hàm compileExpression3(): Xử lý phép cộng và trừ:
    + Sinh mã cho term đầu tiên
    + Sinh mã cho term tiếp theo
    + Sinh lệnh AD (cộng) hoặc SB (trừ)
    + Xử lý đệ quy nếu có nhiều phép toán liên tiếp
- Hàm compileExpression(): Xử lý dấu đơn nguyên:
    + Nếu có dấu âm phía trước: sinh mã cho biểu thức rồi sinh lệnh NEG để đảo dấu
    + Nếu có dấu cộng hoặc không có dấu: sinh mã bình thường cho biểu thức

3. Sinh mã cho điều kiện
   Hàm compileCondition(): Xử lý biểu thức so sánh theo các bước:
    - Sinh mã cho biểu thức bên trái, giá trị lên ngăn xếp
    - Sinh mã cho biểu thức bên phải, giá trị lên ngăn xếp
    - Sinh lệnh so sánh tương ứng (EQ, NE, GT, LT, GE, LE)
    - Kết quả là 1 (đúng) hoặc 0 (sai) được đẩy lên đỉnh ngăn xếp
4. Sinh mã cho lệnh gán
   Hàm compileAssignSt(): Thực hiện gán giá trị cho biến:
   Gọi compileLValue để sinh mã đẩy địa chỉ biến lên ngăn xếp
   Sinh mã cho biểu thức bên phải, giá trị lên ngăn xếp
   Sinh lệnh ST để lưu giá trị vào địa chỉ

Sau lệnh ST, cả địa chỉ và giá trị đều được lấy ra khỏi ngăn xếp

5. Sinh mã cho lệnh IF

   Hàm compileIfSt(): Xử lý hai trường hợp:

   - Trường hợp IF-THEN:
     + Sinh mã cho điều kiện, kết quả (0 hoặc 1) lên ngăn xếp
     + Sinh lệnh FJ với nhãn chưa xác định, nhảy đến cuối IF nếu điều kiện sai
     + Sinh mã cho câu lệnh trong nhánh THEN
     + Cập nhật nhãn của lệnh FJ trỏ đến vị trí ngay sau câu lệnh
   - Trường hợp IF-THEN-ELSE:
     + Sinh mã cho điều kiện
     + Sinh lệnh FJ nhảy đến nhánh ELSE nếu sai
     + Sinh mã cho nhánh THEN
     + Sinh lệnh J để nhảy qua nhánh ELSE
     + Cập nhật nhãn của FJ trỏ đến đầu nhánh ELSE
     + Sinh mã cho nhánh ELSE
     + Cập nhật nhãn của lệnh J trỏ đến sau cấu trúc IF

6. Sinh mã cho vòng lặp WHILE

   Hàm compileWhileSt(): Tạo vòng lặp với điều kiện kiểm tra trước:

   - Lưu địa chỉ đầu vòng lặp
   - Sinh mã cho điều kiện, kết quả lên ngăn xếp
   - Sinh lệnh FJ nhảy ra ngoài vòng lặp nếu điều kiện sai
   - Sinh mã cho thân vòng lặp
   - Sinh lệnh J nhảy về đầu vòng lặp
   - Cập nhật nhãn của FJ trỏ đến sau vòng lặp

7. Sinh mã cho vòng lặp FOR

   Hàm compileForSt(): Đây là phần phức tạp nhất vì cần duy trì địa chỉ biến đếm trên ngăn xếp:

   - Sinh mã đẩy địa chỉ biến đếm lên ngăn xếp
   - Sinh lệnh CV để sao chép địa chỉ (giữ trên ngăn xếp suốt vòng lặp)
   - Sinh mã cho giá trị khởi tạo
   - Sinh lệnh ST để gán giá trị khởi tạo (địa chỉ vẫn còn)
   - Đánh dấu điểm bắt đầu vòng lặp
   - Sao chép địa chỉ biến đếm bằng CV
   - Sinh lệnh LI để lấy giá trị hiện tại của biến đếm
   - Sinh mã cho giá trị giới hạn
   - Sinh lệnh LE để so sánh

- Sinh lệnh FJ để thoát nếu vượt giới hạn
- Sinh mã cho thân vòng lặp
- Tăng biến đếm: CV hai lần, LI, LC 1, AD, ST
- Sinh lệnh J quay về đầu vòng lặp
- Sinh lệnh DCT để dọn địa chỉ biến đếm khỏi ngăn xếp

III. KẾT QUẢ THỰC HIỆN

1. Chương trình không chứa chương trình con

File test.kpl:

```
PROGRAM EXAMPLE5;
VAR J : INTEGER;
    I : INTEGER;

BEGIN
    FOR I := 1 TO 2 DO
        J := 1;
END.
```

Mã sinh ra:

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT\SinhMa\Bai5_Code_Gen> .\codegen.exe tests\test.kpl outputs\test.out -dump
  0:  J 1
  1:  INT 6
  2:  LA 0,5
  3:  CV
  4:  LC 1
  5:  ST
  6:  CV
  7:  LI
  8:  LC 2
  9:  LE
  10:  FJ 21
  11:  LA 0,4
  12:  LC 1
  13:  ST
  14:  CV
  15:  CV
  16:  LI
  17:  LC 1
  18:  AD
  19:  ST
  20:  J 6
  21:  DCT 1
  22:  HL
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT\SinhMa\Bai5_Code_Gen>
```

Phân tích chi tiết:

Dòng 0 lệnh nhảy qua phần khai báo đến thân chương trình chính. Đây là kỹ thuật tối ưu để bỏ qua các khai báo không cần thực thi.

Dòng 1 cấp phát không gian cho khung ngăn xếp với 6 ô nhớ, bao gồm 4 ô dành riêng cho hệ thống và 2 ô cho biến J (offset 4) và I (offset 5).

Dòng 2-5 thực hiện khởi tạo vòng lặp FOR:

- Dòng 2 đẩy địa chỉ của biến I lên ngăn xếp
- Dòng 3 sao chép địa chỉ này để giữ lại trong suốt vòng lặp
- Dòng 4 đẩy giá trị khởi tạo là 1
- Dòng 5 gán giá trị 1 cho I, nhưng địa chỉ I vẫn còn trên ngăn xếp

Dòng 6-10 kiểm tra điều kiện vòng lặp:

- Dòng 6 sao chép địa chỉ I từ đáy ngăn xếp
- Dòng 7 lấy giá trị hiện tại của I thông qua địa chỉ
- Dòng 8 đẩy giá trị giới hạn là 2
- Dòng 9 so sánh I với 2 (kiểm tra I <= 2)
- Dòng 10 nhảy đến dòng 21 nếu điều kiện sai (tức I > 2)

Dòng 11-13 thực hiện thân vòng lặp:

- Dòng 11 đẩy địa chỉ biến J
- Dòng 12 đẩy giá trị 1
- Dòng 13 gán giá trị 1 cho J

Dòng 14-19 tăng biến đếm:

- Dòng 14-15 sao chép địa chỉ I hai lần (một cho việc tính toán, một để giữ lại)
- Dòng 16 lấy giá trị hiện tại của I
- Dòng 17 đẩy hằng số 1
- Dòng 18 cộng I với 1
- Dòng 19 lưu giá trị mới vào I, địa chỉ vẫn còn trên ngăn xếp

Dòng 20 nhảy về dòng 6 để tiếp tục vòng lặp.

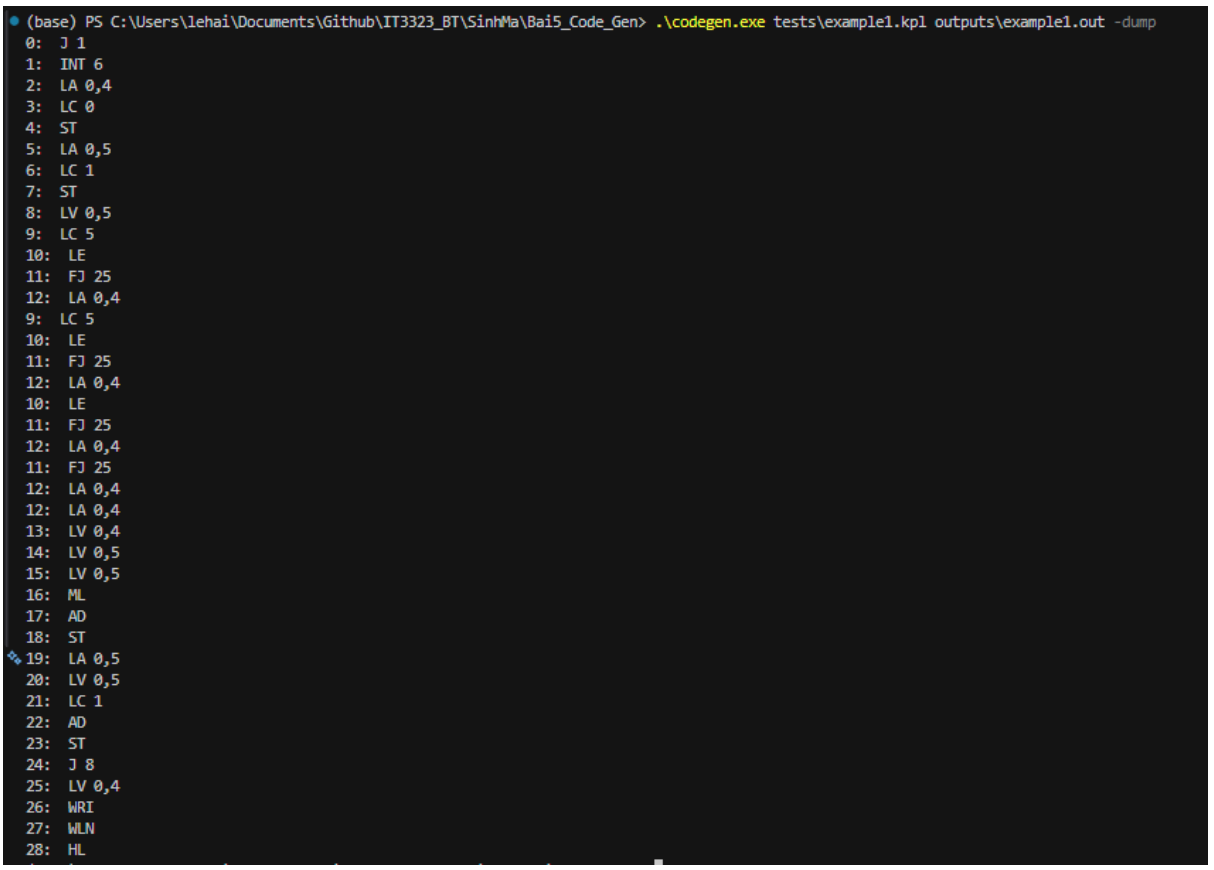
Dòng 21 giảm con trỏ đỉnh ngăn xếp để loại bỏ địa chỉ I đã giữ lại.

Dòng 22 kết thúc chương trình.

File example1.kpl

```
PROGRAM CODEGEN9;
TYPE T = INTEGER;
VAR  S : T;
        I : INTEGER;

BEGIN
    S:=0;
    I:=1;
  WHILE I<=5 DO
BEGIN
S:=S+I*I;
I:=I+1;
END;
CALL WRITEI(S);
CALL WRITELN;
END.
```



Phân tích chi tiết:

0: J 1 => Nhảy qua phần khai báo đến thân chương trình. Đây là bước tối ưu để tránh thực thi nhầm các định nghĩa.

1: INT 6 => Cấp phát stack frame cho chương trình chính: RV(0), DL(1), RA(2), SL(3), S(4), I(5).

Khối S := 0:

- 2: LA 0,4 => Đẩy địa chỉ biến S (offset 4) lên ngăn xế
- 3: LC 0 => Đẩy hằng số 0 lên
- 4: ST => Lưu 0 vào địa chỉ S
- Kết quả: S = 0

Khối I := 1:

- 5: LA 0,5
- 6: LC 1
- 7: ST
- Tương tự, gán giá trị 1 cho biến I tại offset 5 => Kết quả: I = 1

Bắt đầu vòng lặp WHILE:

- 8: LV 0,5 => Đẩy giá trị I lên ngăn xếp → Stack: [1]
- 9: LC 5 => Đẩy hằng 5 lên → Stack: [1, 5]
- 10: LE => So sánh I <= 5, kết quả 1 (đúng) → Stack: [1]

- 11: FJ 28 => Nhảy đến 28 nếu kết quả = 0, ở đây không nhảy vì điều kiện đúng

Thân vòng lặp - S := S + I*I:

- 12: LA 0,4 => Đẩy địa chỉ S → Stack: [addr_S]
- 13: LV 0,4 => Đẩy giá trị S hiện tại (0 ở lần đầu) → Stack: [addr_S, 0]
- 14: LV 0,5 => Đẩy giá trị I hai lần → Stack: [addr_S, 0, 1, 1]
- 15: LV 0,5 => Đẩy giá trị I hai lần → Stack: [addr_S, 0, 1, 1]
- 16: ML => Nhân I*I → Stack: [addr_S, 0, 1]
- 17: AD => Cộng S + (I*I) → Stack: [addr_S, 1]
- 18: ST => Lưu kết quả vào S
- Kết quả lần 1: S = 0 + 1×1 = 1

Tăng I:

- 19: LA 0,5 => Đẩy địa chỉ I
- 20: LV 0,5 => Đẩy giá trị I (1) → Stack: [addr_I, 1]
- 21: LC 1 => Đẩy hằng 1 → Stack: [addr_I, 1, 1]
- 22: AD => I + 1 → Stack: [addr_I, 2]
- 23: ST => Lưu I = 2
- 24: J 8 => Quay về lệnh 8 kiểm tra điều kiện
- Kết quả: I = 2

Các vòng lặp tiếp theo:

- Lần 2: I=2, điều kiện 2≤5 đúng → S = 1 + 2×2 = 5, I = 3
- Lần 3: I=3, điều kiện 3≤5 đúng → S = 5 + 3×3 = 14, I = 4
- Lần 4: I=4, điều kiện 4≤5 đúng → S = 14 + 4×4 = 30, I = 5
- Lần 5: I=5, điều kiện 5≤5 đúng → S = 30 + 5×5 = 55, I = 6
- Lần 6: I=6, điều kiện 6≤5 sai → Nhảy đến lệnh 28

Sau vòng lặp:

- 25: LV 0,4 => Đẩy giá trị S (55) lên ngăn xếp để chuẩn bị cho WRITEI
- 26: WRI => Placeholder cho WRITEI
- 27: WLN => Placeholder cho WRITELN
- 28: HL => Kết thúc chương trình
- Kết quả cuối cùng: S = 55 (đúng với công thức $1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55$)

2. Chương trình chứa chương trình con

File test2.kpl:

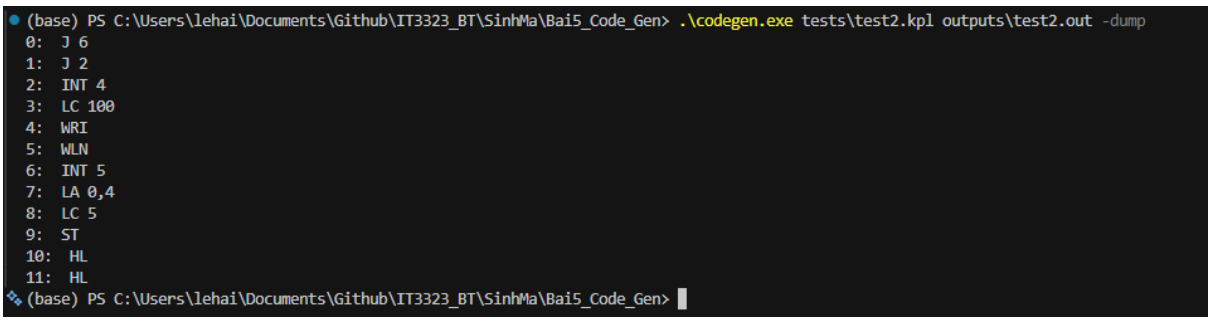
```
PROGRAM TEST2;
```

```
VAR x : INTEGER;

PROCEDURE ShowMessage;
BEGIN
  CALL WRITEI(100);
  CALL WRITELN;
END;

BEGIN
  x := 5;
  CALL ShowMessage;
END.
```

Mã sinh ra:



Phân tích chi tiết:

Dòng 0 nhảy qua vùng định nghĩa thủ tục đến chương trình chính tại dòng 6.

Dòng 1-5 là vùng mã của thủ tục ShowMessage:

- Dòng 1 là điểm vào của thủ tục (sẽ được gọi bởi lệnh CALL)
- Dòng 2 cấp phát khung ngăn xếp với 4 ô cho thủ tục
- Dòng 3 đẩy giá trị 100 lên ngăn xếp để chuẩn bị gọi WRITEI
- Dòng 4 là lệnh HL thay thế cho lời gọi WRITEI (chưa được cài đặt đầy đủ)
- Dòng 5 là lệnh HL thay thế cho WRITELN và lệnh thoát thủ tục

Dòng 6-11 là chương trình chính:

- Dòng 6 cấp phát khung ngăn xếp với 5 ô (4 ô hệ thống + 1 ô cho biến x)
- Dòng 7 đẩy địa chỉ của biến x (offset 4)
- Dòng 8 đẩy giá trị 5
- Dòng 9 thực hiện gán x := 5
- Dòng 10 là lệnh HL thay thế cho lời gọi thủ tục ShowMessage
- Dòng 11 kết thúc chương trình