BÁO CÁO THỰC HÀNH
XÂY DỰNG CHƯƠNG TRÌNH DỊCH
TUẦN 7: SINH MÃ ĐÍCH CƠ BẢN

Họ và tên: Bùi Quang Hưng

Mã số sinh viên: 20225849

I. Tổng quan dự án

1. Mục đích

Dự án này nhằm xây dựng hoàn chỉnh phần **Code Generation** (sinh mã) cho trình biên dịch ngôn ngữ KPL (K Programming Language). Phần Code Generation có nhiệm vụ chuyển đổi cấu trúc cú pháp đã được phân tích và kiểm tra ngữ nghĩa thành mã máy ảo (bytecode) có thể thực thi trên máy ảo KPL.

Mục tiêu cụ thể:

- Sinh mã cho các cấu trúc điều khiển cơ bản (IF-THEN-ELSE, WHILE, FOR).
- Sinh mã cho các biểu thức số học và logic.
- Xử lý khai báo và truy xuất biến, mảng.
- Hỗ trợ thủ tục và hàm với nested scope.
- Tính toán static link và dynamic link để truy xuất biến đúng scope.
- Sinh mã cho các hàm/thủ tục có sẵn (predefined functions/procedures).

2. Luồng hoạt động

Quy trình biên dịch một chương trình KPL diễn ra theo các bước sau:

- **Bước 1: Phân tích từ vựng (Lexical Analysis)**
  - Scanner đọc mã nguồn và tạo ra các token.
  - Nhận dạng từ khóa, định danh, hằng số, toán tử.
  - Hỗ trợ không phân biệt hoa thường cho từ khóa và định danh.
- **Bước 2: Phân tích cú pháp (Syntax Analysis)**
  - Parser xây dựng cây cú pháp dựa trên ngữ pháp KPL.
  - Kiểm tra tính đúng đắn về mặt cú pháp.
  - Gọi các hàm sinh mã tương ứng với mỗi cấu trúc.
- **Bước 3: Phân tích ngữ nghĩa (Semantic Analysis)**
  - Kiểm tra khai báo biến, hàm, thủ tục.
  - Kiểm tra kiểu dữ liệu.
  - Xây dựng bảng symbol table với thông tin về scope và offset.
- **Bước 4: Sinh mã (Code Generation)**
  - Sinh các lệnh cho máy ảo KPL.
  - Tính toán địa chỉ biến, tham số.

- Xử lý static link và dynamic link.
- Tối ưu hóa mã sinh ra.

- **Bước 5: Xuất mã (Code Emission)**
  - Ghi mã bytecode ra file nhị phân.
  - In ra mã assembly để debug (nếu có flag -dump).
- **Kiến trúc máy ảo KPL:**
  - Stack-based architecture.
  - Sử dụng các thanh ghi: PC (Program Counter), SP (Stack Pointer), BP (Base Pointer).
  - Các lệnh cơ bản: LA (Load Address), LV (Load Value), LC (Load Constant), ST (Store), CALL, JMP, FJ (False Jump)...

II. Các case đã implement

1. Khai báo và gán biến

- **Mô tả:** Khai báo biến trong KPL được thực hiện trong phần VAR của chương trình. Mỗi biến được cấp phát một vị trí (offset) trong stack frame của scope hiện tại.
- **Cách thực hiện:**
  - Khi gặp khai báo biến, compiler tính toán offset từ base pointer.
  - Offset được lưu trong symbol table cùng với thông tin về scope.
  - Khi gán giá trị cho biến, sinh lệnh LA (Load Address) để đẩy địa chỉ biến lên stack.
  - Tính toán biểu thức vế phải và đẩy kết quả lên stack.
  - Sinh lệnh ST (Store) để lưu giá trị vào địa chỉ.
- **Ví dụ:**

```
VAR x : INTEGER;
BEGIN
x := 10;
END.
```

- **Mã sinh ra:**

```
LA 0,4    ; Load address của biến x (level 0, offset 4)
LC 10     ; Load constant 10
ST        ; Store giá trị 10 vào địa chỉ x
```

- **Xử lý level:** Level được tính bằng hiệu số giữa scope hiện tại và scope chứa biến. Điều này cho phép truy xuất biến từ outer scope trong nested procedure/function.

2. Biểu thức số học

- **Mô tả:** KPL hỗ trợ các phép toán số học cơ bản: cộng (+), trừ (-), nhân (*), chia (/), và phủ định (-).
- **Cách thực hiện:**
  - Biểu thức được parse theo thứ tự ưu tiên: Factor → Term → Expression.
  - Mỗi toán tử sinh ra một lệnh tương ứng trên stack.
  - Kết quả tính toán được giữ trên đỉnh stack.

- **Các lệnh sinh:**
  - AD: Addition (cộng)
  - SB: Subtraction (trừ)
  - ML: Multiplication (nhân)
  - DV: Division (chia)
  - NEG: Negation (phủ định)
- **Ví dụ:**

S := S + I * I

- **Mã sinh ra:**

```
LA 0,4    ; Load address của S
LV 0,4    ; Load value của S
LV 0,5    ; Load value của I
LV 0,5    ; Load value của I (lần 2)
ML        ; I * I
AD        ; S + (I * I)
ST        ; Store vào S
```

3. Phép so sánh

- **Mô tả:** KPL hỗ trợ 6 phép so sánh: = (bằng), != (khác), < (nhỏ hơn), <= (nhỏ hơn hoặc bằng), > (lớn hơn), >= (lớn hơn hoặc bằng).
- **Cách thực hiện:**
  - Tính giá trị hai biểu thức cần so sánh.
  - Sinh lệnh so sánh tương ứng.
  - Kết quả là 1 (true) hoặc 0 (false) trên stack.
- **Các lệnh sinh:**
  - EQ: Equal (=)
  - NE: Not Equal (!=)
  - LT: Less Than (<)
  - LE: Less or Equal (<=)
  - GT: Greater Than (>)
  - GE: Greater or Equal (>=)
- **Ví dụ:**

IF I <= 5 THEN ...

- **Mã sinh ra:**

```
LV 0,5    ; Load value của I
LC 5      ; Load constant 5
LE        ; So sánh I <= 5
```

4. Câu lệnh IF – THEN – ELSE

- **Mô tả:** Cấu trúc rẽ nhánh IF-THEN-ELSE cho phép thực thi một trong hai nhánh code dựa trên điều kiện.
- **Cách thực hiện:**
  - Tính toán điều kiện (condition).
  - Sinh lệnh FJ (False Jump) để nhảy đến ELSE nếu điều kiện sai.

- Sinh mã cho phần THEN.
- Nếu có ELSE: sinh lệnh J (Jump) để bỏ qua phần ELSE sau khi thực thi THEN.
- Cập nhật địa chỉ nhảy của FJ và J.

- **Sơ đồ luồng:**

<condition>

FJ label_else ; Nếu false, nhảy đến else

<then_statement>

J label_end ; Nhảy qua else

label_else:

<else_statement>

label_end:

- **Ví dụ:**

```
IF (n MOD 2) = 0 THEN
 CALL WRITEC('E')
ELSE
 CALL WRITEC('O')
```

- **Mã sinh ra:**

```
LV 0,4    ; Load n
LV 0,4    ; Load n
LC 2      ; Load 2
DV        ; n / 2
LC 2      ; Load 2
ML        ; (n/2) * 2
SB        ; n - (n/2)*2
LC 0      ; Load 0
```

```
EQ        ; So sánh với 0
FJ 18     ; Nếu false, nhảy đến địa chỉ 18 (else)
LC 69     ; Load 'E'
CALL 1,0  ; Gọi WRITEC
J 20      ; Nhảy qua else
LC 79     ; Load 'O' (địa chỉ 18)
CALL 1,0  ; Gọi WRITEC
          ; Địa chỉ 20 (end)
```

5. Vòng lặp WHILE

- **Mô tả:** Vòng lặp WHILE thực thi một khối lệnh lặp đi lặp lại trong khi điều kiện còn đúng.
- **Cách thực hiện:**
  - Đánh dấu địa chỉ bắt đầu vòng lặp (beginLoop).
  - Tính toán điều kiện.
  - Sinh lệnh FJ để thoát vòng lặp nếu điều kiện sai.
  - Sinh mã cho thân vòng lặp.
  - Sinh lệnh J để nhảy về đầu vòng lặp.
  - Cập nhật địa chỉ của FJ.
- **Sơ đồ luồng:**

```
beginLoop:
<condition>
FJ label_end      ; Nếu false, thoát vòng lặp
<loop_body>
J beginLoop       ; Quay lại đầu vòng lặp
label_end:
```

- **Ví dụ:**

```
WHILE I <= 5 DO
BEGIN
  S := S + I * I;
  I := I + 1;
END
```

- **Mã sinh ra:**

```
LV 0,5    ; (địa chỉ 8) Load I
LC 5      ; Load 5
LE        ; I <= 5
FJ 25     ; Nếu false, nhảy đến 25 (end)
LA 0,4    ; Load address của S
LV 0,4    ; Load value S
LV 0,5    ; Load value I
LV 0,5    ; Load value I
ML        ; I * I
AD        ; S + I*I
ST        ; Store vào S
LA 0,5    ; Load address của I
LV 0,5    ; Load value I
LC 1      ; Load 1
AD        ; I + 1
ST        ; Store vào I
J 8       ; Quay lại địa chỉ 8
          ; Địa chỉ 25 (end)
```

6. Vòng lặp FOR

- **Mô tả:** Vòng lặp FOR là một dạng vòng lặp đặc biệt với biến đếm, giá trị bắt đầu, giá trị kết thúc.
- **Cách thực hiện:**
  - Gán giá trị khởi tạo cho biến điều khiển.
  - Đánh dấu địa chỉ bắt đầu vòng lặp.
  - So sánh biến điều khiển với giá trị kết thúc (<=).
  - Sinh lệnh FJ để thoát nếu vượt giá trị kết thúc.
  - Sinh mã cho thân vòng lặp.
  - Tăng biến điều khiển lên 1.
  - Sinh lệnh J để quay lại đầu vòng lặp
- **Sơ đồ luồng:**

<control_var> := <start_value>
ST
beginLoop:
LV control_var
<end_value>
LE
FJ label_end
<loop_body>
<control_var> := <control_var> + 1
J beginLoop
label_end:

- **Ví dụ:**

FOR I := 1 TO N DO
A(.I.) := READI

- **Mã sinh ra:**

```
e  LA 0,15   ; Load address của I
LC 1      ; Load 1 (start value)
ST        ; I := 1
LV 0,15   ; (địa chỉ 8) Load I
LV 0,14   ; Load N
LE        ; I <= N
FJ 27     ; Nếu false, thoát vòng lặp
LA 0,4    ; Load base address của array A
LV 0,15   ; Load I
LC 1      ; Load 1
SB        ; I - 1 (array index from 1)
LC 1      ; Element size
ML        ; (I-1) * size
AD        ; Base + offset
CALL 1,0  ; READI
ST        ; Store vào A[I]
LA 0,15   ; Load address của I
LV 0,15   ; Load I
LC 1      ; Load 1
```

```
AD       ; I + 1
ST       ; Store vào I
J 8      ; Quay lại địa chỉ 8
         ; Địa chỉ 27 (end)
```

7. Mảng (ARRAY)

- **Mô tả:** KPL hỗ trợ mảng một chiều với cú pháp: ARRAY(. size .) OF type
- **Cách thực hiện:**
  - Mảng được cấp phát liên tiếp trong stack frame.
  - Địa chỉ phần tử thứ i: base_address + (i - 1) × element_size.
  - KPL sử dụng chỉ số bắt đầu từ 1.
  - Khi truy xuất phần tử mảng:
    - Tính địa chỉ phần tử: base + (index - 1) × size.
    - Sinh lệnh LI (Load Indirect) để đọc giá trị.
    - Hoặc dùng địa chỉ để gán giá trị.
- **Công thức tính địa chỉ:**

$$address = base\_address + (index - 1) \times element\_size$$

- **Ví dụ:**

```
TYPE T = INTEGER;
VAR A : ARRAY(. 10 .) OF T;
BEGIN
  A(.5.) := 100;
END
```

- **Mã sinh ra:**

```
LA 0,4    ; Load base address của A
LC 5      ; Load index 5
LC 1      ; Load 1
SB        ; 5 - 1 = 4
LC 1      ; Load element_size
ML        ; 4 * 1 = 4
AD        ; base + 4
LC 100    ; Load value 100
ST        ; Store 100 vào A[5]
```

8. Thủ tục và hàm (Procedure & Function)

- **Mô tả:** KPL hỗ trợ khai báo procedure (không trả về giá trị) và function (trả về giá trị).
- **Cách thực hiện:**
  - Khai báo:
    - Mỗi procedure/function có một địa chỉ code (code address).
    - Có scope riêng chứa tham số và biến local.
    - Sinh lệnh J ở đầu để bỏ qua phần khai báo.
    - Sau khi thực thi xong, sinh lệnh EP (Exit Procedure) hoặc EF (Exit Function).
  - **Gọi procedure/function:**

- Đẩy các tham số lên stack.
- Sinh lệnh CALL với level và code address.
- Lệnh CALL thực hiện:
  - Lưu dynamic link (BP cũ)
  - Lưu return address (PC cũ)
  - Lưu static link (BP của outer scope)
  - Cập nhật BP và PC

- **Stack frame layout:**

[offset 0]: Return value (chỉ cho function)

[offset 1]: Dynamic Link (old BP)

[offset 2]: Return Address (old PC)

[offset 3]: Static Link (BP of outer scope)

[offset 4+]: Parameters

[offset n+]: Local variables

- **Ví dụ:**

```
PROCEDURE DISPLAY(N: INTEGER);
BEGIN
  CALL WRITEI(N);
END;


BEGIN
  CALL DISPLAY(10);
END.
```

- **Mã sinh ra:**

```
e  J 2       ; Bỏ qua phần khai báo procedure
INT 5     ; (địa chỉ 2) Allocate stack frame
LV 0,4    ; Load parameter N
CALL 2,0  ; Gọi WRITEI (level 2, predefined)
```

```
EP       ; Exit procedure

INT 5    ; (địa chỉ 5) Main program frame

LC 10    ; Load 10

CALL 1,2 ; Gọi DISPLAY (level 1, address 2)

HL       ; Halt
```

9. Predefined Functions/Procedures

- **Mô tả:** KPL cung cấp sẵn các hàm và thủ tục thư viện:
  - **READI**: Đọc một số nguyên từ input.
  - **READC**: Đọc một ký tự từ input.
  - **WRITEI(n)**: In số nguyên n ra output.
  - **WRITEC(ch)**: In ký tự ch ra output.
  - **WRITELN**: In xuống dòng.
- **Cách thực hiện:**
  - Các hàm/thủ tục này được đăng ký sẵn trong global symbol table.
  - Khi gọi, compiler nhận biết và sinh lệnh đặc biệt:
    - RI: Read Integer
    - RC: Read Char
    - WRI: Write Integer
    - WRC: Write Char
    - WLN: Write Line
  - Không cần tính toán level và address như user-defined function
- **Ví dụ:**

```
BEGIN
  n := READI;
  CALL WRITEI(n);
  CALL WRITELN;
END.
```

- **Mã sinh ra:**

```
LA 0,4   ; Load address của n
CALL 1,0 ; Gọi READI (sinh lệnh RI)
ST       ; Store vào n
LV 0,4   ; Load value của n
CALL 1,0 ; Gọi WRITEI (sinh lệnh WRI)
CALL 1,0 ; Gọi WRITELN (sinh lệnh WLN)
```

10. Static Link và Nested Scope

- **Mô tả:** KPL cho phép nested procedure/function (procedure/function lồng nhau). Để truy xuất biến từ outer scope, compiler sử dụng static link.
- **Cách thực hiện:**
  - Tính level:

$$level = current\_scope\_level - variable\_scope\_level$$

- Sinh lệnh truy xuất biến:
    - LV (Load Value): Đọc giá trị biến.
    - LA (Load Address): Lấy địa chỉ biến.
    - Tham số p trong lệnh LA/LV p,q là level difference.

- **Ví dụ với nested procedure:**

```
PROGRAM NESTED;
VAR x : INTEGER;

PROCEDURE OUTER;
VAR y : INTEGER;

 PROCEDURE INNER;
 BEGIN
   x := x + y;  (* Truy xuất x từ program scope, y từ outer scope *)
 END;

BEGIN
 y := 5;
 CALL INNER;
END;

BEGIN
 x := 10;
 CALL OUTER;
END.
```

- **Phân tích level:**
    - Biến x: program scope (level 0)
    - Biến y: OUTER scope (level 1)
    - Trong INNER (level 2):
        - Truy xuất x: level difference = 2 - 0 = 2
        - Truy xuất y: level difference = 2 - 1 = 1
- **Mã sinh trong INNER:**

```
LA 2,4    ; Load address của x (level 2, từ program scope)
LV 2,4    ; Load value của x
LV 1,4    ; Load value của y (level 1, từ OUTER scope)
AD        ; x + y
ST        ; Store vào x
```

III. Kết quả

1. Tệp example1.kpl không có chương trình con

- Mã nguồn:

Bai5_Code_Gen > tests > example1.kpl

```
1   PROGRAM CODEGEN9;
2   TYPE T = INTEGER;
3   VAR  S : T;
4            I : INTEGER;
5
6   BEGIN
7       S:=0;
8       I:=1;
9     WHILE I<=5 DO
10  BEGIN
11  S:=S+I*I;
12  I:=I+1;
13  END;
14  CALL WRITEI(S);
15  CALL WRITELN;
16  END.
```

- Kết quả:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

```
PS D:\CTD_Lab\Lab5.2\Bai5_Code_Gen> .\codegen.exe tests\example1.kpl output\example1
 0:  J 1
 1:  INT 6
 2:  LA 0,4
 3:  LC 0
 4:  ST
 5:  LA 0,5
 6:  LC 1
 7:  ST
 8:  LV 0,5
 9:  LC 5
 10:  LE
 11:  FJ 25
 12:  LA 0,4
 13:  LV 0,4
 14:  LV 0,5
 15:  LV 0,5
 16:  ML
 17:  AD
 18:  ST
 19:  LA 0,5
 20:  LV 0,5
 21:  LC 1
 22:  AD
 23:  ST
 24:  J 8
 25:  LV 0,4
 26:  CALL 1,0
 27:  CALL 1,0
 28:  HL
PS D:\CTD_Lab\Lab5.2\Bai5_Code_Gen>
```

2. Tệp example3.kpl có chương trình con

- Mã nguồn:

Bai5_Code_Gen > tests > example3.kpl

```
1   PROGRAM  EXAMPLE3;  (* TOWER OF HANOI *)
2   VAR  I:INTEGER;
3        N:INTEGER;
4        P:INTEGER;
5        Q:INTEGER;
6        C:CHAR;
7
8   PROCEDURE  HANOI(N:INTEGER;  S:INTEGER;  Z:INTEGER);
9   BEGIN
10    IF  N != 0  THEN
11      BEGIN
12        CALL  HANOI(N-1,S,6-S-Z);
13        I:=I+1;
14        CALL  WRITELN;
15        CALL  WRITEI(I);
16        CALL  WRITEI(N);
17        CALL  WRITEI(S);
18        CALL  WRITEI(Z);
19        CALL  HANOI(N-1,6-S-Z,Z)
20      END
21  END;  (*END OF HANOI*)
22
23  BEGIN
24    FOR  N := 1  TO  4  DO
25      BEGIN
26        FOR  I:=1  TO  4  DO
27          CALL  WRITEC(' ');
28        C :=  READC;
29        CALL  WRITEC(C)
30      END;
31    P:=1;
32    Q:=2;
33    FOR  N:=2  TO  4  DO
34      BEGIN
35        I:=0;
36        CALL  HANOI(N,P,Q);
37        CALL  WRITELN
38      END
39  END.  (* TOWER OF HANOI *)
```

- Kết quả:

```
PS D:\CTD_Lab\Lab5.2\Bai5_Code_Gen> .\codegen.exe tests\example3.kpl output\example3.bin
0:  J 54
1:  J 2
2:  INT 7
3:  LV 0,4
4:  LI
5:  LC 0
6:  NE
7:  FJ 53
8:  LV 0,4
9:  LI
10:  LC 1
11:  SB
12:  LV 0,5
13:  LI
14:  LC 6
15:  LV 0,5
16:  LI
17:  SB
18:  LV 0,6
19:  LI
20:  SB
21:  CALL 0,1
22:  LA 1,4
23:  LV 1,4
24:  LC 1
25:  AD
26:  ST
27:  CALL 2,0
28:  LV 1,4
29:  CALL 2,0
30:  LV 0,4
31:  LI
32:  CALL 2,0
33:  LV 0,5
34:  LI
35:  CALL 2,0
36:  LV 0,6
37:  LI
```

```
 37:  LI
 38:  CALL 2,0
 39:  LV 0,4
 40:  LI
 41:  LC 1
 42:  SB
 43:  LC 6
 44:  LV 0,5
 45:  LI
 46:  SB
 47:  LV 0,6
 48:  LI
 49:  SB
 50:  LV 0,6
 51:  LI
 52:  CALL 0,1
 53:  EP
 54:  INT 9
 55:  LA 0,5
 56:  LC 1
 57:  ST
 58:  LV 0,5
 59:  LC 4
 60:  LE
 61:  FJ 88
 62:  LA 0,4
 63:  LC 1
 64:  ST
 65:  LV 0,4
 66:  LC 4
 67:  LE
 68:  FJ 77
 69:  LC 32
 70:  CALL 1,0
 71:  LA 0,4
 72:  LV 0,4
 73:  LC 1
 74:  AD
 75:  ST
 76:  J 65
 77:  LA 0,8
 78:  CALL 1,0
 79:  ST
 80:  LV 0,8
 81:  CALL 1,0
 82:  LA 0,5
```

```
 82:  LA 0,5
 83:  LV 0,5
 84:  LC 1
 85:  AD
 86:  ST
 87:  J 58
 88:  LA 0,6
 89:  LC 1
 90:  ST
 91:  LA 0,7
 92:  LC 2
 93:  ST
 94:  LA 0,5
 95:  LC 2
 96:  ST
 97:  LV 0,5
 98:  LC 4
 99:  LE
 100:  FJ 115
 101:  LA 0,4
 102:  LC 0
 103:  ST
 104:  LV 0,5
 105:  LV 0,6
 106:  LV 0,7
 107:  CALL 1,1
 108:  CALL 1,0
 109:  LA 0,5
 110:  LV 0,5
 111:  LC 1
 112:  AD
 113:  ST
 114:  J 97
 115:  HL
PS D:\CTD_Lab\Lab5.2\Bai5_Code_Gen>
```