Bài 13
Sinh mã đích

ONE LOVE. ONE FUTURE.

Nội dung

- Tổng quan về sinh mã đích
- Máy ngăn xếp
  - Tổ chức bộ nhớ
  - Bộ lệnh
- Sinh mã cho các lệnh cơ bản
- Xây dựng bảng ký hiệu
  - Biến
  - Tham số
  - Hàm, thủ tục và chương trình

Chương trình đích

- Viết trên một ngôn ngữ trung gian
- Là dạng Assembly của máy giả định (máy ảo)
- Máy ảo làm việc với bộ nhớ stack
- Việc thực hiện chương trình thông qua một interpreter
- Interpreter mô phỏng hành động của máy ảo thực hiện tập lệnh assembly của nó

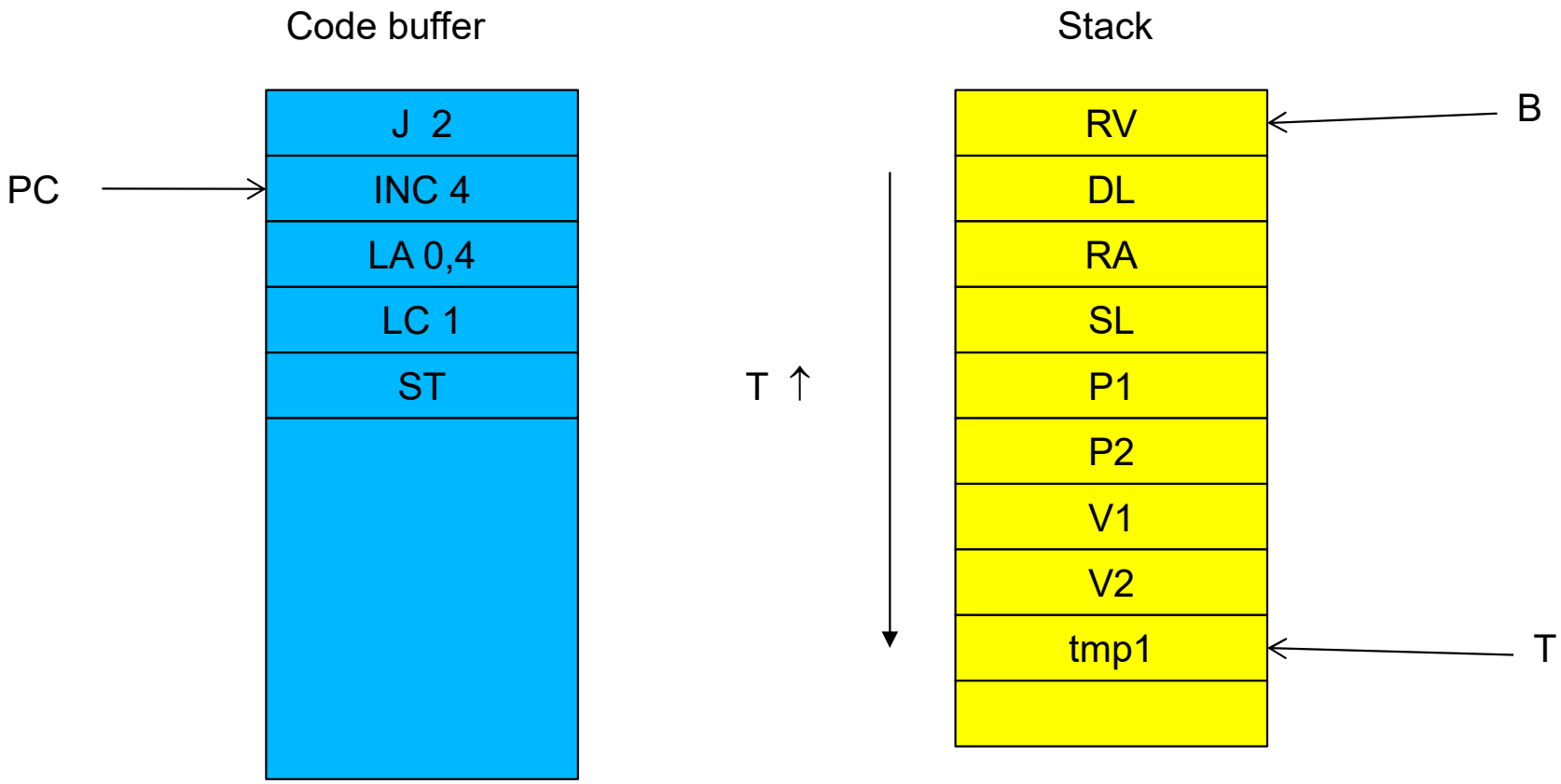
Chương trình đích được dịch từ

- Mã nguồn
- Mã trung gian

Máy ngăn xếp (stack calculator)

- Máy ngăn xếp là một hệ thống tính toán
  - Sử dụng ngăn xếp để lưu trữ các kết quả trung gian của quá trình tính toán
  - Kiến trúc đơn giản
  - Bộ lệnh đơn giản
- Máy ngăn xếp có hai vùng bộ nhớ chính
  - Khối lệnh: chứa mã thực thi của chương trình
  - Ngăn xếp: sử dụng để lưu trữ các kết quả trung gian

Máy ngăn xếp


Code buffer
Stack
J 2
INC 4
LA 0,4
LC 1
ST
PC
T
RV
DL
RA
SL
P1
P2
V1
V2
tmp1
B
T

Máy ngăn xếp

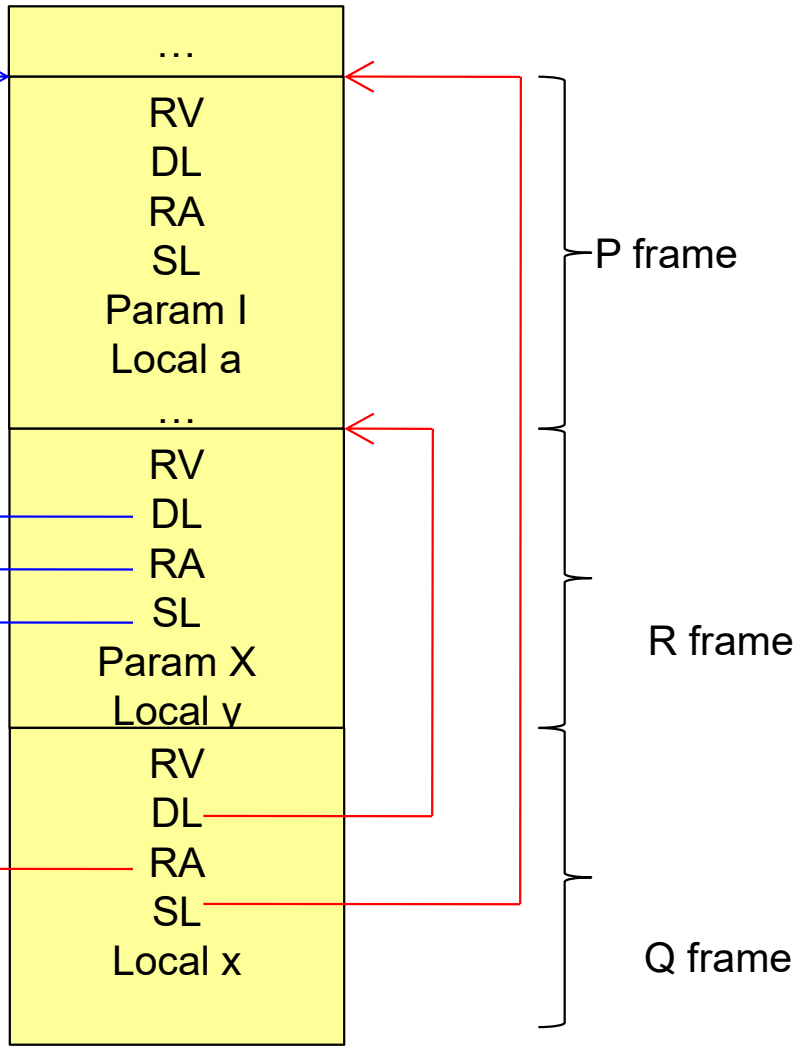
- Thanh ghi
  - PC (program counter): con trỏ lệnh trỏ tới lệnh hiện tại đang thực thi trên bộ đệm chương trình
  - B (base) : con trỏ trỏ tới địa chỉ gốc của vùng nhớ cục bộ. Các biến cục bộ được truy xuất gián tiếp qua con trỏ này
  - T (top); trỏ tới đỉnh của ngăn xếp

Máy ngăn xếp

- Bản hoạt động (activation record/stack frame)
  - Không gian nhớ cấp phát cho mỗi chương trình con (hàm/thủ tục/chương trình chính) khi chúng được kích hoạt
    - Lưu giá trị tham số
    - Lưu giá trị biến cục bộ
    - Lưu các thông tin khác
      - Giá trị trả về của hàm – RV
      - Địa chỉ cơ sở của bản hoạt động của chương trình con gọi tới (caller) – DL
      - Địa chỉ lệnh quay về khi kết thúc chương trình con – RA
      - Địa chỉ cơ sở của bản hoạt động của chương trình con bao ngoài – SL
  - Một chương trình con có thể có nhiều bản hoạt động

Máy ngăn xếp

```
Procedure P(I : integer);
  Var a : integer;
  Function Q:integer;
    Var x : char;
    Begin
      …
      Q:= … ..
    End;
  Procedure R(X: integer);
    Var y : char;
    Begin
      …
      y := Q + 1;
      …
    End;
  Begin
    …
    Call R(1);
    …
  End;
…
```



Máy ngăn xếp

- RV (return value): Lưu trữ giá trị trả về cho mỗi hàm
- DL (dynamic link): Sử dụng để hồi phục ngữ cảnh của chương trình gọi (caller) khi chương trình được gọi (callee) kết thúc
- RA (return address): Sử dụng để tìm tới lệnh tiếp theo của caller khi callee kết thúc
- SL (static link): Sử dụng để truy nhập các biến phi cục bộ

Bộ lệnh của máy ngăn xếp

- Dạng lệnh:

LA	Load Address	t:=t+1; s[t]:=base(p)+q;
LV	Load Value	t:=t+1; s[t]:=s[base(p)+q];
LC	Load Constant	t:=t+1; s[t]:=q;
LI	Load Indirect	s[t]:=s[s[t]];
INT	Increment T	t:=t+q;
DCT	Decrement T	t:=t-q;

Các lệnh chuyển điều khiển

- Dạng lệnh

J	Jump	pc:=q;
FJ	False Jump	if s[t]=0 then pc:=q; t:=t-1;
HL	Halt	Halt
ST	Store	s[s[t-1]]:=s[t]; t:=t-2;
CALL	Call	s[t+2]:=b; s[t+3]:=pc; s[t+4]:=base(p); b:=t+1; pc:=q;
EP	Exit Procedure	t:=b-1; pc:=s[b+2]; b:=s[b+1];
EF	Exit Function	t:=b; pc:=s[b+2]; b:=s[b+1];

Các lệnh vào ra

- Dạng lệnh

RC	Read Character	Đọc 1 ký tự vào địa chỉ trên đỉnh stack s[s[t]]; t:=t-1;
RI	Read Integer	Đọc 1 số nguyên vào địa chỉ trên đỉnh stack s[s[t]]; t:=t-1;
WRC	Write Character	In ký tự ở đỉnh (s[t]); t:=t-1;
WRI	Write Integer	write integer from s[t]; t:=t-1;
WLN	New Line	CR & LF

Các lệnh tính toán

- Dạng lệnh

AD	Cộng	t:=t-1; s[t]:=s[t]+s[t+1];
SB	Trừ	t:=t-1; s[t]:=s[t]-s[t+1];
ML	Nhân	t:=t-1; s[t]:=s[t]*s[t+1];
DV	Chia	t:=t-1; s[t]:=s[t]/s[t+1];
NEG	Đổi dấu	s[t]:=-s[t];
CV	Sao chép nội dung ô đỉnh stack	s[t+1]:=s[t]; t:=t+1;

Các lệnh so sánh

- Bộ lệnh

op	p	q

EQ	Bằng	t:=t-1; if s[t] = s[t+1] then s[t]:=1 else s[t]:=0;
NE	Khác	t:=t-1; if s[t] != s[t+1] then s[t]:=1 else s[t]:=0;
GT	Lớn hơn	t:=t-1; if s[t] > s[t+1] then s[t]:=1 else s[t]:=0;
LT	Nhơ hơn	t:=t-1; if s[t] < s[t+1] then s[t]:=1 else s[t]:=0;
GE	Lớn hơn hoặc bằng	t:=t-1; if s[t] >= s[t+1] then s[t]:=1 else s[t]:=0;
LE	Nhỏ hơn hoặc bằng	t:=t-1; if s[t] <= s[t+1] then s[t]:=1 else s[t]:=0;

Sinh mã lệnh gán

V := exp

```
<code of l-value v>  // Sinh ra lệnh đẩy địa chỉ của v
                     //lên stack
<code of exp>        // Sinh ra lệnh đẩy giá trị của exp
                     //lên stack
ST       //Sinh ra lệnh ST
```

Cú pháp của lệnh gán

$S \rightarrow id := E$

$E \rightarrow - E_2 \mid + E_2 \mid E_2$

$E_2 \rightarrow TE_3$

$E_3 \rightarrow +TE_3 \mid -TE3 \mid \varepsilon$

$T \rightarrow FT_2$

$T_2 \rightarrow {}^*FT_2 \mid /FT_2 \mid \varepsilon$

$F \rightarrow id \mid num \mid (E)$

(Trường hợp F là biến có chỉ số hoặc lời gọi hàm xét sau)

Lvalue

```
case OBJ_VARIABLE:
    genVariableAddress(var);
    if (var->varAttrs->type->typeClass ==
TP_ARRAY)
    {varType = compileIndexes
    (var->varAttrs->type);}
    else
      varType = var->varAttrs->type;
    break;
```

Expression3

```
switch (lookAhead->tokenType)
{   case SB_PLUS:
      eat(SB_PLUS);
      checkIntType(argType1);
      argType2 =
      compileTerm();
      checkIntType(argType2);
      genAD();
      resultType =
  compileExpression3(argType1);
      break;
```

```
case SB_MINUS:
      eat(SB_MINUS);
      checkIntType(argType1);
      argType2 = compileTerm();
      checkIntType(argType2);
      genSB();
      resultType =

compileExpression3(argType1);
      break;
```

Term2

```
switch (lookAhead->tokenType) {
  case SB_TIMES:
    eat(SB_TIMES);
    checkIntType(argType1);
    argType2 = compileFactor();
    checkIntType(argType2);
    genML();
    resultType =
    compileTerm2(argType1);
    break;
```

```
case SB_SLASH:
    eat(SB_SLASH);
    checkIntType(argType1);
    argType2 =
compileFactor();
    checkIntType(argType2);
    genDV();
    resultType =
    compileTerm2(argType1);
    break;
```

Sinh mã lệnh if

If condition Then statement;

```
   <code of condition>      // đẩy giá trị điều kiện lên stack
   FJ L
   <code of statement>
L:
   …
```

If condition Then st1 Else st2;

```
   <code of condition>      // đẩy giá trị điều kiện lên stack
   FJ L1
   <code of st1>
   J L2
L1:
   <code of st2>
L2:
   …
```

Sinh mã lệnh while

While <dk> Do statement

```
L1:
   <code of dk>
   FJ L2
   <code of statement>
   J L1
L2:
   …
```

Sinh mã lệnh for

For v := exp1 to exp2 do statement

```
  <code of l-value v>
  CV   // nhân đôi địa chỉ của v
  <code of exp1>
  ST   // lưu giá trị đầu của v
L1:
  CV
  LI   // lấy giá trị của v
  <code of exp2>
  LE
  FJ L2
  <code of statement>
  CV;CV;LI;LC 1;AD;ST;  // Tăng v lên 1
  J L1
L2:
  DCT 1
  …
```

Lấy địa chỉ/giá trị biến

- Khi lấy địa chỉ/giá trị một biến cần tính đến phạm vi của biến
  - Biến cục bộ được lấy từ frame hiện tại
  - Biến phi cục bộ được lấy theo các StaticLink với cấp độ lấy theo "độ sâu" của phạm vi hiện tại so với phạm vi của biến

```
computeNestedLevel(Scope* scope)
```

Lấy địa chỉ của tham số hình thức

- Khi **LValue** là tham số
- Cũng cần tính độ sâu như biến
  - Nếu là tham trị: địa chỉ cần lấy chính là địa chỉ của tham trị
  - Nếu là tham biến: vì giá trị của tham biến chính là địa chỉ muốn truy nhập, địa chỉ cần lấy chính là giá trị của tham biến.

Lấy giá trị của tham số thực sự

- Khi tính toán giá trị của **Factor**
- Cũng cần tính độ sâu như biến
  - Nếu là tham trị: giá trị của tham trị chính là giá trị cần lấy.
  - Nếu là tham biến: giá trị của tham số là địa chỉ của giá trị cần lấy.

Lấy địa chỉ của giá trị trả về của hàm

- Giá trị trả về luôn nằm ở offset 0 trên frame
- Chỉ cần tính độ sâu giống như với biến hay tham số hình thức

Sinh lời gọi hàm/thủ tục

- Lời gọi
  - Hàm gặp trong sinh mã cho **`factor`**
  - Thủ tục gặp trong sinh mã lệnh **`CallSt`**
- Trước khi sinh lời gọi hàm/thủ tục cần phải nạp giá trị cho các tham số hình thức bằng cách
  - Tăng giá trị T lên 4 (bỏ qua RV,DL,RA,SL)
  - Sinh mã cho k tham số thực tế
  - Giảm giá trị T đi 4 + k
  - Sinh lệnh CALL

Sinh mã cho lệnh CALL (p, q)

```
CALL (p, q)
                s[t+2]:=b;          // Lưu lại dynamic link
                s[t+3]:=pc;        // Lưu lại return address
                s[t+4]:=base(p);   // Lưu lại static link
                b:=t+1;            // Base mới và return value
                pc:=q;             // địa chỉ lệnh mới
```

Giả sử cần sinh lệnh CALL cho hàm/thủ tục A
Lệnh CALL(p, q) có hai tham số:

p: Độ sâu của lệnh CALL, chứa static link.
Base(p) = base của frame chương trình con chứa khai báo của A.

q: Địa chỉ lệnh mới
q + 1 = địa chỉ đầu tiên của dãy lệnh cần thực hiện khi gọi A.

Hoạt động khi thực hiện lệnh CALL(p, q)

1. Điều khiển **pc** chuyển đến địa chỉ bắt đầu của chương trình con **/* pc = p */**
2. **pc** tăng thêm 1 **/* pc ++ */**
3. Lệnh đầu tiên thông thường là lệnh nhảy J để bỏ qua mã lệnh của các khai báo hàm/ thủ tục cục bộ trên **code buffer.**
4. Lệnh tiếp theo là lệnh **INT** tăng **T** đúng bằng kích thước frame để bỏ qua frame chứa vùng nhớ của các tham số và biến cục bộ.

Hoạt động khi thực hiện lệnh CALL(p, q)

5. Thực hiện các lệnh và stack biến đổi tương ứng.
6. Khi kết thúc
   a. Thủ tục (lệnh **`EP`**): toàn bộ frame được giải phóng, con trỏ **`T`** đặt lên đỉnh frame cũ.
   b. Hàm (lệnh **`EF`**): frame được giải phóng, chỉ chừa giá trị trả về tại **`offset 0,`** con trỏ **`T`** đặt lên đầu frame hiện thời (**`offset 0`**).

Ví dụ 2

```
Function F(N : Integer) : Integer;
  Begin
    If N = 0 Then F := 1 Else F := N * F (N - 1);
  End;

Begin
      Call WriteLn;
      Call WriteI(F(5));
End. (* Factorial *)
```

Code ASM được sinh ra

```
0:  J 22
1:  J 2
2:  INT 5
3:  LV 0,4
4:  LC 0
5:  EQ
6:  FJ 11
7:  LA 0,0
8:  LC 1
9:  ST
10:  J 21
11:  LA 0,0
12:  LV 0,4
13:  INT 4
14:  LV 0,4
15:  LC 1
16:  SB
17:  DCT 5
18:  CALL 1,1
19:  ML
20:  ST
21:  EF
22:  INT 4
23:  WLN
24:  INT 4
25:  LC 5
26:  DCT 5
27:  CALL 0,1
28:  WRI
29:  HL
```

Sinh mã đích từ mã ba địa chỉ

- Bộ sinh mã trung gian đưa ra mã ba địa chỉ
- Tối ưu trên mã ba địa chỉ
- Từ mã ba địa chỉ đã tối ưu sinh ra mã đích phù hợp với một mô tả máy ảo
- Sinh trực tiếp từ mã trung gian, không cần qua cây phân tích cú pháp có chú giải.