Experiment in Compiler Construction

Sinh mã đích(1)

Viện Công nghệ Thông tin và Truyền thông
Đại học Bách khoa Hà nội

Nội dung

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

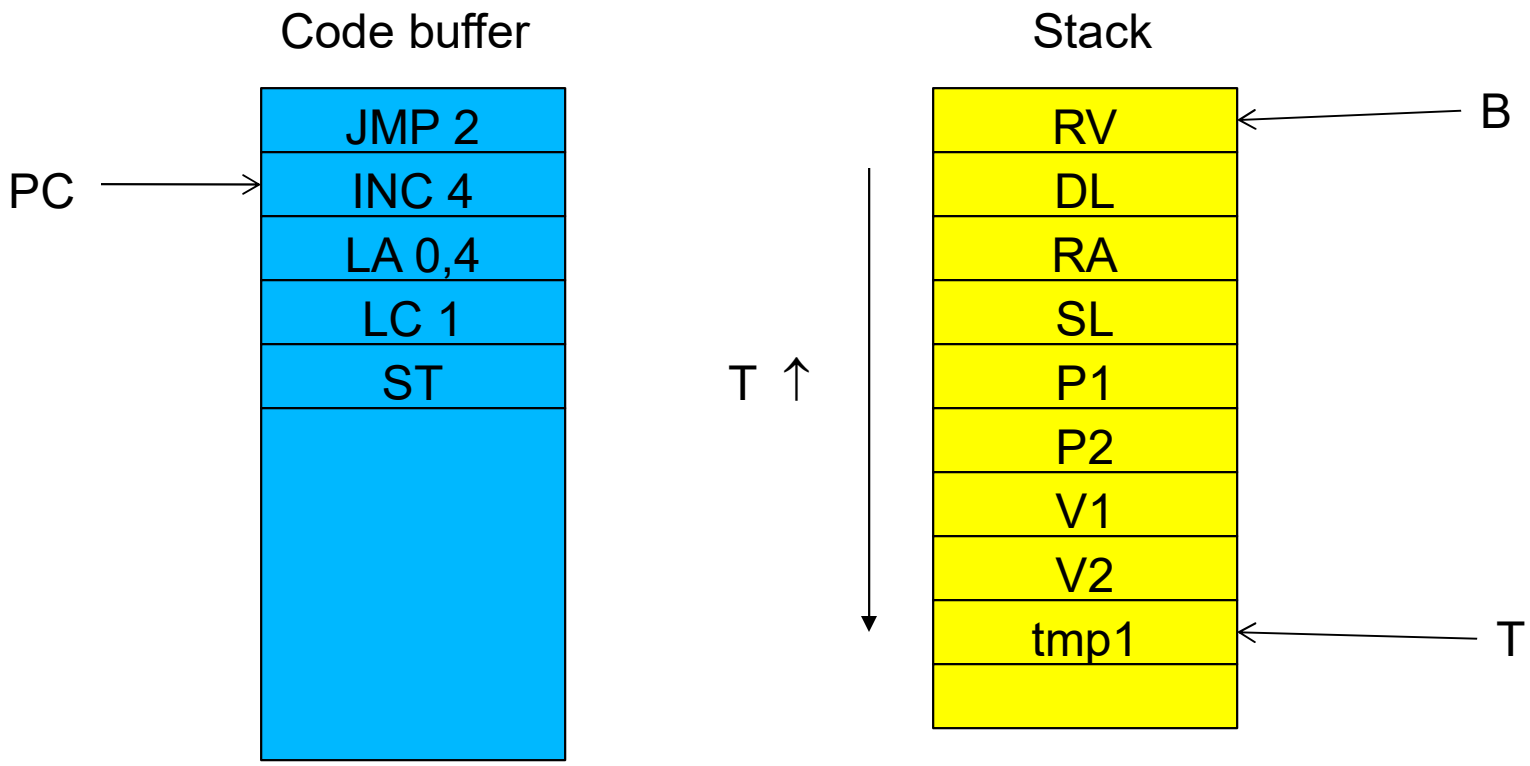
Sinh mã là gì?

- Sinh mã là công đoạn biến đổi từ cấu trúc ngữ pháp của chương trình thành chuỗi các lệnh thực thi được của máy đích
- Cấu trúc ngữ pháp được quyết định bởi bộ phân tích cú pháp
- Các lệnh của máy đích được đặc tả bởi kiến trúc thực thi của máy đích

Máy ngăn xếp

- 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
JMP 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

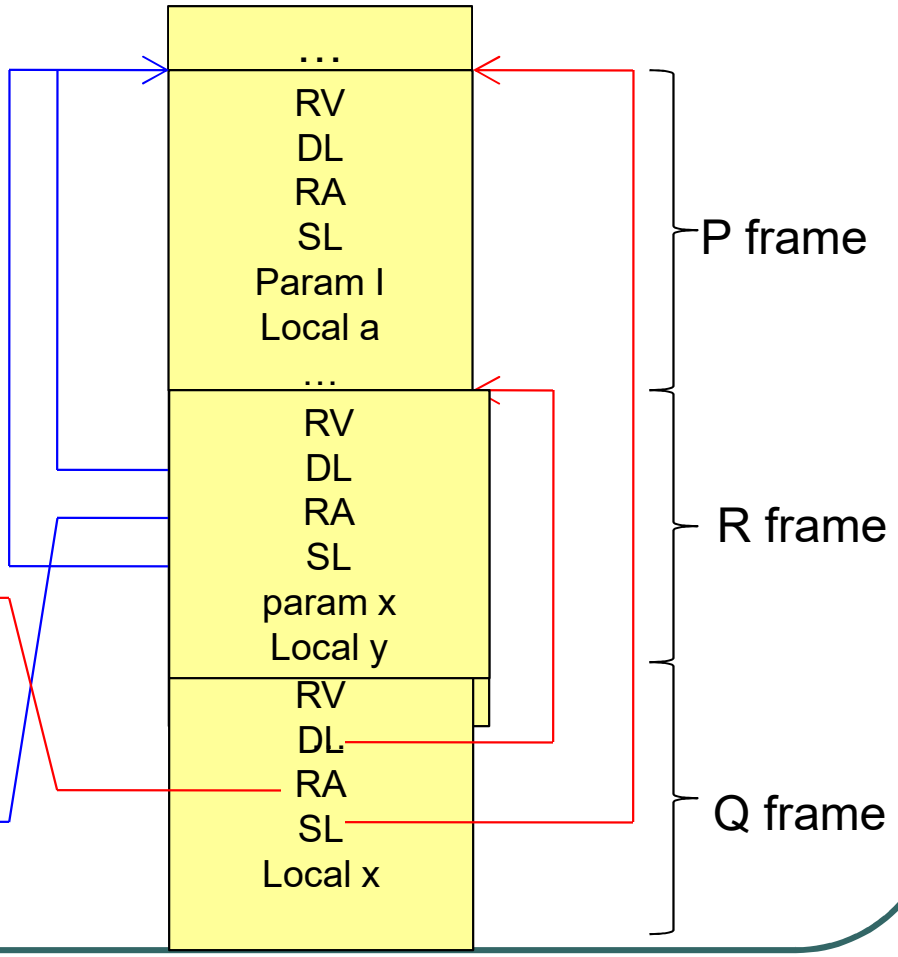
- Thanh ghi
  - PC (program counter): con trỏ lệnh trỏ tới lệnh hiện tại đang thực thi trên bộ đệm chương trình
  - B (base) : con trỏ trỏ tới địa chỉ gốc của vùng nhớ cục bộ. Các biến cục bộ được truy xuất gián tiếp qua con trỏ này
  - T (top); trỏ tới đỉnh của ngăn xếp

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

Máy ngăn xếp

- Bộ 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;

Máy ngăn xếp

- Bộ 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];

Máy ngăn xếp

- Bộ lệnh

RC	Read Character	read one character into s[s[t]]; t:=t-1;
RI	Read Integer	read integer to s[s[t]]; t:=t-1;
WRC	Write Character	write one character from s[t]; t:=t-1;
WRI	Write Integer	write integer from s[t]; t:=t-1;
WLN	New Line	CR & LF

Máy ngăn xếp

- Bộ lệnh

AD	Add	t:=t-1; s[t]:=s[t]+s[t+1];
SB	Subtract	t:=t-1; s[t]:=s[t]-s[t+1];
ML	Multiply	t:=t-1; s[t]:=s[t]*s[t+1];
DV	Divide	t:=t-1; s[t]:=s[t]/s[t+1];
NEG	Negative	s[t]:=-s[t];
CV	Copy Top of Stack	s[t+1]:=s[t]; t:=t+1;

Máy ngăn xếp

- Bộ lệnh

EQ	Equal	t:=t-1; if s[t] = s[t+1] then s[t]:=1 else s[t]:=0;
NE	Not Equal	t:=t-1; if s[t] != s[t+1] then s[t]:=1 else s[t]:=0;
GT	Greater Than	t:=t-1; if s[t] > s[t+1] then s[t]:=1 else s[t]:=0;
LT	Less Than	t:=t-1; if s[t] < s[t+1] then s[t]:=1 else s[t]:=0;
GE	Greater or Equal	t:=t-1; if s[t] >= s[t+1] then s[t]:=1 else s[t]:=0;
LE	Less or Equal	t:=t-1; if s[t] <= s[t+1] then s[t]:=1 else s[t]:=0;

Xây dựng bảng ký hiệu

- Bổ sung thông tin cho biến
  - Vị trí trên frame
  - Phạm vi
- Bổ sung thông tin cho tham số
  - Vị trí trên frame
  - Phạm vi
- Bổ sung thông tin cho hàm/thủ tục/chương trình
  - Địa chỉ bắt đầu
  - Kích thước của frame
  - Số lượng tham số của hàm/thủ tục

Xây dựng bảng ký hiệu

- Bổ sung thông tin cho biến
  - Vị trí trên frame (vị trí tính từ base của frame)
  - Phạm vi

```
struct VariableAttributes_ {
  Type *type;
  struct Scope_ *scope;
  int localOffset;
};
```

Xây dựng bảng ký hiệu

- Bổ sung thông tin cho tham số
  - Vị trí trên frame
  - Phạm vi

```
struct ParameterAttributes_ {
  enum ParamKind kind;
  Type* type;
  struct Scope_ *scope;
  int localOffset;
};
```

Xây dựng bảng ký hiệu

- Bổ sung thông tin cho phạm vi
  - Kích thước frame

```
struct Scope_ {
  ObjectNode *objList;
  Object *owner;
  struct Scope_ *outer;
  int frameSize;
};
```

Xây dựng bảng ký hiệu

- Bổ sung thông tin cho hàm
  - Vị trí
  - Số lượng tham số

```
struct FunctionAttributes_ {
  struct ObjectNode_ *paramList;
  Type* returnType;
  struct Scope_ *scope;

  int paramCount;
  CodeAddress codeAddress;
};
```

Xây dựng bảng ký hiệu

- Bổ sung thông tin cho thủ tục
  - Vị trí
  - Số lượng tham số

```
struct ProcedureAttributes_ {
  struct ObjectNode_ *paramList;
  struct Scope_* scope;

  int paramCount;
  CodeAddress codeAddress;
};
```

Xây dựng bảng ký hiệu

- Bổ sung thông tin cho chương trình
  - Vị trí

```
struct ProgramAttributes_ {
  struct Scope_ *scope;
  CodeAddress codeAddress;
};
```

Nhiệm vụ

- Viết các hàm sau trên symtab.c

```
int sizeOfType(Type* type);
void declareObject(Object* obj);
```

- Lưu ý: Để đơn giản hóa, mỗi giá trị interger/char đều chiếm một từ (4 bytes) trên ngăn xếp.
- Thứ tự các từ trên 1 frame như sau
  - 0: RV
  - 1: DL
  - 2: RA
  - 3: SL
  - 4 $\rightarrow$ (4+k): k tham số
  - (4+k+1) $\rightarrow$ (4+k+n): biến cục bộ

int sizeOfType(type* t)

- Trả về số ngăn nhớ trên stack mà một biến thuộc kiểu đó sẽ chiếm.

void declareObject(Object* obj)

- Đối tượng toàn cục
  - Đưa vào symtab->GlobalObjectList
- Đối tượng khác:
  - Đưa vào symtab->currentScope->objList
  - Variable:
    - Cập nhật scope = currentScope
    - Cập nhật localOffset = currentScope->frameSize
    - Tăng kích thước frameSize

void declareObject(Object* obj)

- Đối tượng khác:
  - Parameter
    - Cập nhật scope = currentScope
    - Cập nhật localOffset = currentScope->frameSize
    - Tăng kích thước frameSize
    - Cập nhật paramList của owner
    - Tăng paramCount của owner.

void declareObject(Object* obj)

- Đối tượng khác:
  - Function
    - Cập nhật outer = currentScope
  - Procedure
    - Cập nhật outer = currentScope