Thực hành
Xây dựng chương trình dịch

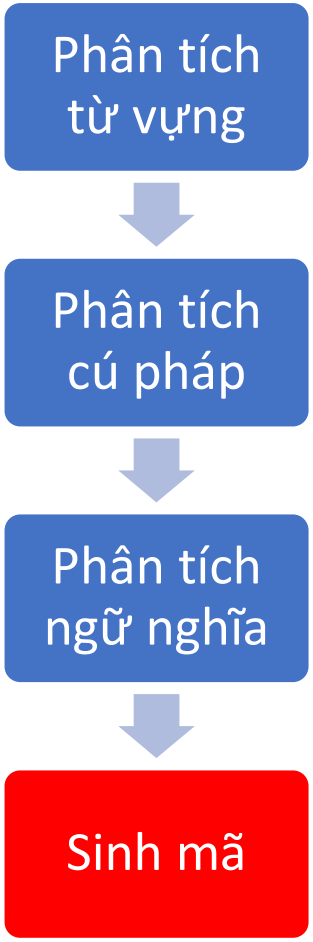
Bài 5: Sinh mã đích

ONE LOVE. ONE FUTURE.

Nội dung

- Tổng quan về sinh mã đích
- Máy ngăn xếp
  - Tổ chức bộ nhớ
  - Bộ lệnh
  - Giới thiệu kplrun
- Giới thiệu instructions.*, codegen.*
- Sinh mã (không chương trình con/array)
  - Sinh mã cho lệnh gán
  - Sinh mã cho lệnh if
  - Sinh mã cho lệnh while
  - Sinh mã cho lệnh for
  - Sinh mã cho điều kiện
  - Sinh mã cho biểu thức

Sinh mã là gì?

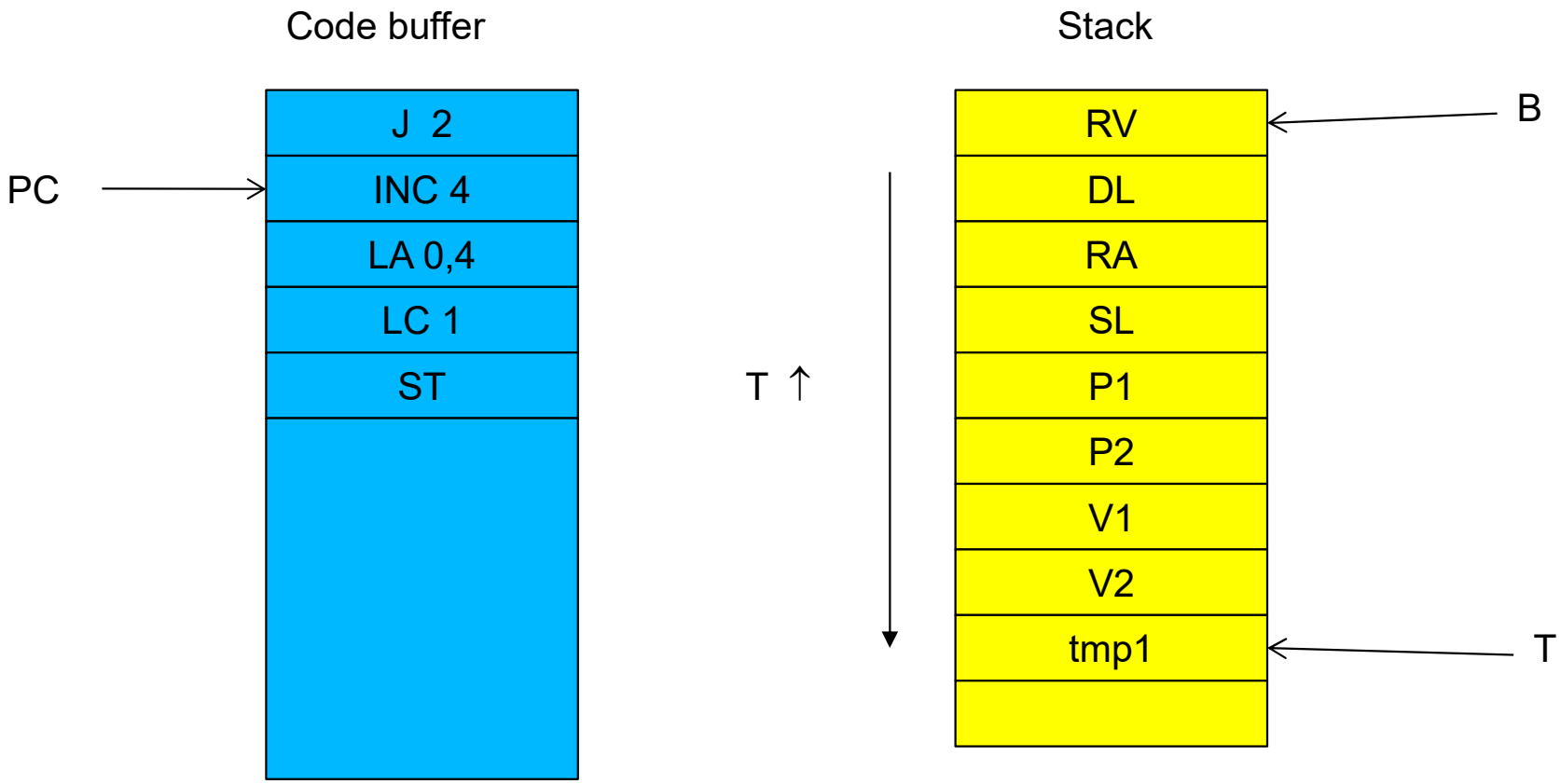


- Sinh mã là công đoạn biến đổi từ cấu trúc ngữ pháp của chương trình thành chuỗi các lệnh thực thi được của máy đích
- Cấu trúc ngữ pháp được quyết định bởi bộ phân tích cú pháp
- Các lệnh của máy đích được đặc tả bởi kiến trúc thực thi của máy đích

Máy ngăn xếp (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

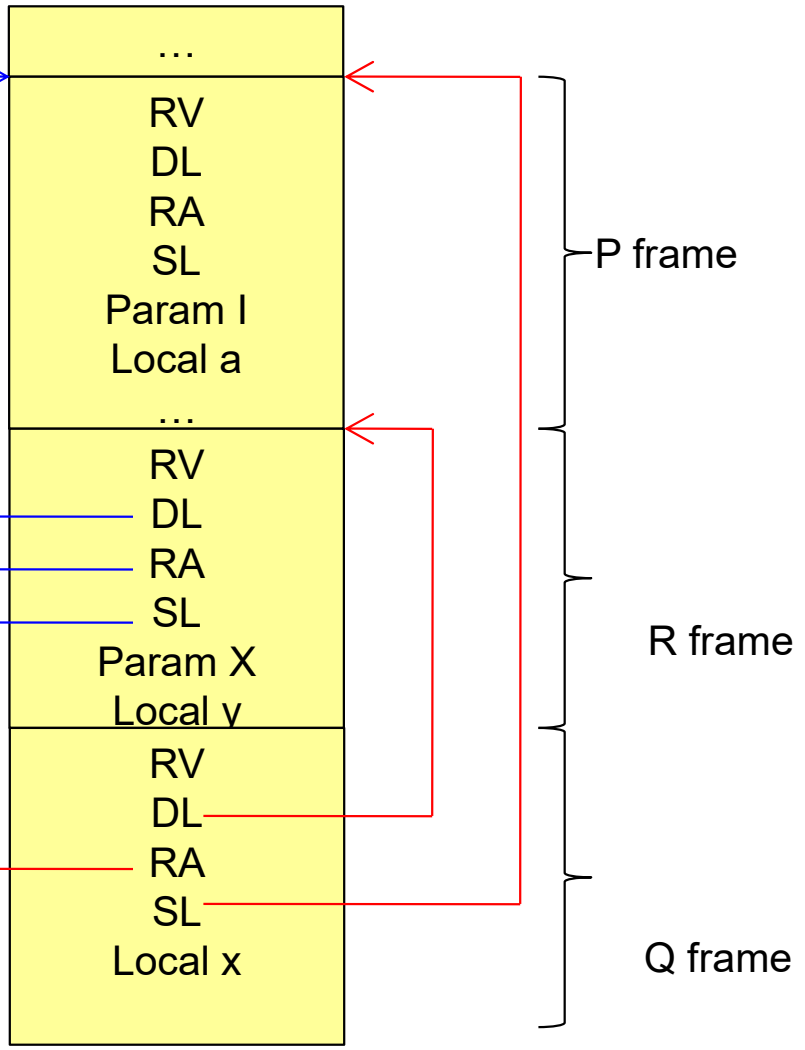
- Thanh ghi
  - PC (program counter): con trỏ lệnh trỏ tới lệnh hiện tại đang thực thi trên bộ đệm chương trình
  - B (base) : con trỏ trỏ tới địa chỉ gốc của vùng nhớ cục bộ. Các biến cục bộ được truy xuất gián tiếp qua con trỏ này
  - T (top); trỏ tới đỉnh của ngăn xếp

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

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;

kplrun

- Là bộ thông dịch cho máy ngăn xếp
- Cú pháp

```
$ kplrun <source> [-s=stack-size] [-c=code-size] [-debug] [-dump]
```

- Tùy chọn –s: định nghĩa kích thước stack
- Tùy chọn –c: định nghĩa kích thước tối đa của mã nguồn
- Tùy chọn –dump: In mã ASM
- Tùy chọn –debug: chế độ gỡ rối

kplrun

- Tùy chọn –debug: chế độ gỡ rối
  - a: địa chỉ tuyệt đối của địa chỉ tương đối (level, offset)
  - v: giá trị tại địa chỉ tương đối (level,offset)
  - t: giá trị đầu ngăn xếp
  - c: thoát khỏi chế độ gỡ rối

Cấu trúc của bộ sinh mã(chưa hoàn thiện)

#	Tên tệp	Nhiệm vụ
1	Makefile	Quản lý project. Sinh viên tự make
2	scanner.c, scanner.h	Phân tích từ vựng
3	reader.h, reader.c	Đọc từng ký tự của chương trình nguồn
4	charcode.h, charcode.c	Phân loại các ký tự
5	token.h, token.c	Phát hiện các từ tố
6	error.h, error.c	Xử lý các loại lỗi
7	parser.c, parser.h	Bộ phân tích cú pháp
8	debug.c, debug.h	In ấn
9	symtab.c symtab.h	Xây dựng bảng ký hiệu
10	semantics.c. semantics.h	Các hàm hỗ trợ cho phân tích ngữ nghĩa
11	instruction.c, instruction.h	Quản lý khối lệnh
12	codegen.c, codegen.h	Sinh mã các lệnh
13	main.c	Chương trình chính

Instructions.c

```
enum OpCode {
  OP_LA,    // Load Address:
  OP_LV,    // Load Value:
  OP_LC,    // Load Constant
  OP_LI,    // Load Indirect
  OP_INT,   // Increment t
  OP_DCT,   // Decrement t
  OP_J,     // Jump
  OP_FJ,    // False Jump
  OP_HL,    // Halt
  OP_ST,    // Store
  OP_CALL,  // Call
  OP_EP,    // Exit Procedure
  OP_EF,    // Exit Function
```

```
  OP_RC,    // Read Char
  OP_RI,    // Read Integer
  OP_WRC,   // Write Char
  OP_WRI,   // Write Int
  OP_WLN,   // WriteLN
  OP_AD,    // Add
  OP_SB,    // Substract
  OP_ML,    // Multiple
  OP_DV,    // Divide
  OP_NEG,   // Negative
  OP_CV,    // Copy Top
  OP_EQ,    // Equal
  OP_NE,    // Not Equal
  OP_GT,    // Greater
  OP_LT,    // Less
  OP_GE,    // Greater or Equal
  OP_LE,    // Less or Equal

  OP_BP     // Break point.
};
```

Instructions.c

```
struct Instruction_ {
  enum OpCode op;
  WORD p;
  WORD q;
};


struct CodeBlock_ {
  Instruction* code;
  int codeSize;
  int maxSize;
};
```

```
CodeBlock* createCodeBlock(int maxSize);
void freeCodeBlock(CodeBlock* codeBlock);
void printInstruction(Instruction* instruction);
void printCodeBlock(CodeBlock* codeBlock);

void loadCode(CodeBlock* codeBlock, FILE* f);
void saveCode(CodeBlock* codeBlock, FILE* f);

int emitLA(CodeBlock* codeBlock, WORD p, WORD q);
int emitLV(CodeBlock* codeBlock, WORD p, WORD q);
int emitLC(CodeBlock* codeBlock, WORD q);
….
int emitLT(CodeBlock* codeBlock);
int emitGE(CodeBlock* codeBlock);
int emitLE(CodeBlock* codeBlock);

int emitBP(CodeBlock* codeBlock);
```

codegen.c

```
void initCodeBuffer(void);
void printCodeBuffer(void);
void cleanCodeBuffer(void);
int serialize(char* fileName);

int genLA(int level, int offset);
int genLV(int level, int offset);
int genLC(WORD constant);
….
int genLT(void);
int emitGE(void);
int emitLE(void);
```

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
  ...
```

Ví dụ 5

```
Program Example5;

Var j : Integer;
    i : Integer;

Begin
  for i := 1 to 2 do
    j := 1;
End.
```

```
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 23
11:  LA 0,4
12:  LC 1
13:  ST
14:  CV
15:  CV
16:  LI
17:  LC 1
18:  AD
19:  ST
20:  CV
21:  LI
22:  J 8
23:  DCT 1
24:  HL
```

Nhiệm vụ

- Điền vào codegen.c
  - `genVariableAddress(Object* var)`

    // Đẩy địa chỉ một biến lên stack
  - `genVariableValue(Object* var)`

    // Đẩy giá trị một biến lên stack
- Tạm thời xem các biến đều nằm mức 0 (trên frame hiện tại)

Nhiệm vụ

- Điền vào parser.c
  - Sinh mã l-value cho biến
  - Sinh mã các câu lệnh: gán, if, while,for
  - Sinh mã điều kiện
  - Sinh mã biểu thức