

تولید کد میانی

کد میانی چیست؟

کدی است در سطح زبان ماشین، اما مستقل از ساختار هر ماشین خاص، در واقع یک شبه اسمبلی است، که متعلق به هیچ ماشین خاصی نیست، لذا به آن ماشین مجازی (Virtual Machine) نیز گفته می شود، برای مثال JVM که همان ByteCode است که :

- امکان پیاده سازی راست و بدون اشکال دستورالعمل های جاوا را به شبه اسمبلی فراهم نمود.
- (در پاسکال Pcode و در C، Ccode داریم)

علت استفاده از کد میانی چیست؟

کد میانی :

- قابل اجرا در روی هر کامپیوتری می باشد (Portability).
- بهینه سازی خوبی دارد (Optimization).

انواع کدهای میانی :

۱. دستورالعملهای شبه اسمبلی.
- در واقع این دستورالعملها بیانگر یک ماشین مجازی می باشند، زیرا محدودیت در تعداد ثباتها را ندارند.
۲. دستورالعملهای سه آدرسه.

۳. درختهای خلاصه نحوی. *Abstract Syntax Trees*

روش تولید کد میانی :

۱. آیا در مرحله ای جداگانه کد میانی ایجاد می شود (از لحاظ سرعت بسیار کند می باشد).
۲. آیا همگام با تحلیل نحوی، این کار صورت می گیرد (از لحاظ پیاده سازی دارای مشکل می باشد).

حل مشکل :

پاسخ در روشی است تحت عنوان ترجمه هدایت شده همگام با تحلیل نحوی (Syntax Directed)

(Translation).

زبان کد میانی → ۱-۱

روش :

- بعد از تشخیص هر ترم میانی.
- بعد از Reduce در روش بالا به پایین.
- بعد از فراخوانی روال هر ترم میانی در روش بالا به پایین، کد مربوطه تکمیل می شود.

مثال :

تولید کد میانی برای عبارت زیر :

$a*(b+c)+d*e$

```
mov T1, a
mov T2, b
add T2, c
mul T1, T2
mov T2, d
mul T2, e
add T2, T1
```

هدف :

تولید مولد کد شبه اسمبلی برای عبارات چهار عمل اصلی.

```
E → E + T | E - T | T
T → T * F | T / F | F
F → id | no | '(' E ')'
```

برای اینکه بتوان تجزیه گر کاهینه بازگشتی برای گرامر عبارات ایجاد نماییم، ابتدا باید گرامر را به فرم

LL(1) تبدیل می کنیم.

```
E → T{ ( + | - ) T }
T → F{ ( * | / ) F }
F → id | no | '(' E ')'
```

هر بار که New Temp اجرا می شود، String جدیدی را بعنوان متغیر کمکی بر می گرداند.

Var

Target, Source : Text;

TempNo : integer; // مقدار اولیه صفر

TempVar : string; // شامل رشته موقتی متغیر

Procedure NewTemp; // متغیر موقتی جدید ایجاد می کند

Begin

Tempno := TempNo + 1;

TempVar := 'T' + str(TempNo);

End;

Procedure RemTemp; // متغیر موقتی را برگشت می دهد

Begin

Tempno := TempNo - 1;

End;

Procedure Emitln(string s);

```

Begin
  Writeln( Target, S);
End;
Procedure Emit( string s );
Begin
  Write( Target, S);
End;

```

حال شروع به نوشتن تک تک تابع می کنیم :

```

Stops : set of symbols;
//-----
-----
(*F → id | no | '(' E ')' *)
Procedure F ( Stop : Stops; Var ResF : string; R : integer;);
Var
  R : integer;
  ResF : string;
Begin
  R := 0;
  If ( CurrentSymbol = S_id ) or ( CurrentSymbol = S_no) then
    Begin
      ResF := CurrentToken.Lexeme;
      NextSymbol;
    End
  Else
    Begin
      Expect( S_OpenPar, Stop + [S_id, S_no, S_OpenPar]);
      E ( Stop + [S_ClosePar], ResF, R);
      Expect( S_ClosePar, Stop);
    End;
  End;
End;
//-----
-----
(* T → F{ ( * | /) F } *)
Procedure E ( Stop : Stops; Var ResT : string; R : integer;);
Var
  OpCode : Symbols;
  Operand : string;
  R1 : integer;
Begin
  F( Stop + [S_mul, S_div], ResT, R);
  While ( CurrentSymbol = S_div) or ( CurrentSymbol = S_mul) do
    Begin
      OpCode := CurrentSymbol;
      NextSymbol;
      If R = 0 then
        Begin
          NewTemp;
          OpCode := TempVar;
          Emitln('mov ' + Operand + ',' + ResT);
          R := 1;
          ResT := Operand;

```

```

    End;
    F ( Stop + [ S_mul, S_div], Operand, R1);
    If (OpCode = S_mul) then
        Emit('mul ')
    Else Emit('div ');
    Emitln(ResT + ',' + Operand);
    If R1 = 1 then
        RemTemp;
    End;
End;
//-----
-----
(* E → T{ ( + | - ) T } *)
Procedure E ( Stop : Stops; Var ResE : string; R : integer);
Var
    OpCode : Symbols;
    Operand : string;
    R1 : integer;
Begin
    T( Stop + [S_add, S_sub], ResT, R);
    While (CurrentSymbol = S_add) or (CurrentSymbol = S_sub) do
        Begin
            OpCode := CurrentSymbol;
            If R = 0 then
                begin
                    NewTemp;
                    Operand := TempVar;
                    Emitln('mov ' + Operand + ',' + ResE);
                    R = 1;
                    ResE := Operand;
                End; { end of if }
            If OpCode = S_add then
                Emit('add ')
            Else Emit('sub ');
            T(Stop + [S_add, S_sub], Operand, R1);
            Emitln(ResE + ',' + Operand);
            If R1 = 1 then
                RemTemp;
            End; { end of while }
        End; { end of procedure }
End;

```

هدف :

تولید دستورالعملهای شبه اسمبلی برای انواع جملات.

روش :

ترجمه همراه با هدایت نحوی (Syntax Directed Translation)

```

(* id := E *)
Procedure Assignment( Stop : Stops; Var ResA : string; R : integer);
Var

```

```

ResE : string;
Begin
  ResA := CurrentToken.Lexeme;
  Expect(S_id, Stop + [S_assign]);
  Expect(S_assign, Stop + [S_id, S_no, S_OpenPar]);
  E(Stop, ResE);
  Emitln('mov ' + ResA + ',' + ResE);
  If R = 1 then
    RemTemp;
    R := 0;
End;

```

مثال :

الف-جملات تخصیصی

ب-جملات شرطی (جملات If)

```

If a > b then c := a - b
Else c := a + b;

```

```

mov T1, a
cmp T1, b
jle L1
mov T1, a
sub T1, b
mov c, T1
jmp L2
L1:
mov T1, a
add T1, b
mov c, T1
L2:

```

```

Var
  LabelNo : integer;
Function NewLabel : string;
Begin
  LabelNo := LabelNo + 1;
  NewLabel := 'L' + str(LabelNo);
End;

```

```

(* Condition > E Relop E *)
Procedure Condition( Stop : Stops; Var ResC : string; R : integer);
Var
  ResE , ResE1 : string;
  LFalse : string;
  R1, R2 : integer;
  OpCode : Symbols;
Begin
  E(Stop + [S_lt, S_le, S_gt, S_ge, S_ne, S_e], ResE, R1);
  OpCode := CurrentSymbol;
  Relop(Stop + [ S_id, S_if,...]);

```

```

E(Stop, ResE1, R2);
Emitln('cmp ' + ResE + ',' + ResE1);
If R1 = 1 then
  RemTemp;
  R := 1;
  ResC := NewTemp;
  Emitln('xor ' + ResC + ',' + ResC);
  LFalse :=NewLabel;
  Case OpCode of
    S_lt :
      Emitln('jge ' + LFalse);
    S_le :
      Emitln('jg ' + LFalse);
    S_ne :
      Emitln('je ' + LFalse);
    ...
  end;
  Emitln('inc ' + ResC);
  Emitln(LFalse + ':');
End;

```

تمرین :

برای عبارات If ، Case و While با در نظر گرفتن گرامر Condition ، کد شبه اسمبلی تولید نمایید.

```

(* Ifst > If Condition Then st Else st *)
Procedure Ifst( Stop : Stops; Var ResIf : string; R : integer);
Var
  ResCond : string;
  LabelElse, LabelEnd : string;
  RC : integer;
Begin
  Expect(S_if, Stop + [S_OpenPar, S_id, S_no, S_not]);
  Condition(Stop + [S_then], ResCond, RC);
  Emitln('cmp ' + ResCond + ',' + '0');
  LabelElse :=NewLabel;
  RemTemp;
  Emitln('je ' + LabelElse);
  Expect(S_then, Stop + [S_id, S_while, S_begin, ...]);
  st(Stop + [S_Else], '0', 0);
  if CurrentSymbol = S_Else then
    Begin
      LabelEnd :=NewLabel;
      Emitln('jmp ' + LabelEnd);
      Emitln(LabelElse + ':');
      NextSymbol;
      st(Stop, '0', 0);
    End
  Else
    LabelEnd := LabelElse;
    Emitln(LabelEnd + ':');
  End;
End;

```

مثال :

```
If a > b then c := a + 1
Else c := b + a;
  xor T1, T1
  mov T2, a
  cmp T2, b
  jle L1
  inc T1
L1:
  cmp T1, 0
  je L2
  mov T1, a
  add T1, 1
  mov c, T1
  jmp L3
L2:
  mov T1, b
  add T1, a
  mov c, T1
L3:
  ...
```

نکته :

باید توجه نمود که در تابع Condition که قبلا نوشته شده است، باید کد بصورتی تولید شود که ابتدا متغیر موقتی به ResC تخصیص داده شود تا پس از انجام مقایسه بتوان به درستی RemTemp را به اجرا در آورد.

تمرین :

رویه هایی بنویسید که برای جملات Case کد تولید نمایند.

```
Case> Case id of
      CasePart {CasePart}
      [ElsePart]
      End;
Casepart> (id | no ) {, ( id | no )} ':' st ';' ;
```

تولید کد اسمبلی

هدف :

استفاده از دستورالعملهای اسمبلی پروسسورهای اینتل جهت تولید کد اسمبلی.

در این راستا می بایست قوانین اسمبلی رعایت شود و در عملیاتی مانند جمع و ضرب از اکومولاتور (ثبات AX) به عنوان اولین عملگر استفاده شود.

محدودیت دیگر، تعداد ثباتها می باشد که محدود هستند، دیگر اینکه همواره اولین عملگر می بایست در يك ثبات قرار گیرد. برای چگونگی تخصیص ثباتها از تابعی به نام Register Management استفاده می شود.

: Register Management

RM یا مدیریت ثباتها، يك جدول به نام Register Table می باشد که دارای وضعیت تخصیصی Registerها را در آن مشخص می کنیم. این کلاس در تابع اصلی به نام Get و Free دارد که جهت گرفتن و با آزاد کردن Registerها مورد استفاده Code Generator قرار می دهد. يك نکته ای که RM در نظر می گیرد علاوه بر تخصیص ثبات مشخص می کند که ثبات حافظه مقدار کدام متغیر است.

```
mov DI, 1
add AX, DI
add AX, 1
mov DI, AX
mov AX, DI
mul AX, 2
mov SI, AX
Reg := GetReg('AX');
Reg2 := CheckReg(CurrentToken.Lexeme);
Reg := GetReg('j');
Reg := RM.GetReg(CurrentToken.Lexeme);
'mov DI, 1'
Reg := RM.GetReg('AX');
Reg2 := CheckReg(CurrentToken.Lexeme);
'mov AX, DI'
'add AX, 1'
Reg := GetReg(CurrentToken.Lexeme);
'mov DI, AX'
```

تا به حال برای تولید کدهای اسمبلی از متغیرهای موقتی استفاده می کردیم ، برای مثال

$(a+b*c) * (d+c*f)$

```
Mov t1,a
Mov t2,b
Mul t2,c
Add t1,t2
Mov t2,d
Mov t3,c
Mul t3,f
Add t2,t3
Mul t1,t2
```


اما چنانچه که میدانیم کدهای بالا به صورت عملی کاربردی ندارد و اجرا نخواهد شد. برای اینکه بتوانیم کدهای بالا را قابل اجرا کنیم باید به جای متغیرهای موقتی از ثباتها استفاده کنیم به این طدیق که به جای متغیرهای موقتی ، ثباتها را جایگزین کنیم.

سؤال: چرا نیاز به مدیریت ثباتها داریم؟

(۱) تعداد ثباتها به اندازه تعداد متغیرهای موقتی نیست (محدودیت ثباتها)

(۲) نسبت دادن متغیر های موقتی به ثباتها مشکل بوجود می آورد.

یک روش کلی (جایگزاری):

برای تبدیل کد بالا به کد قابل اجرا باید جای متغیرهای موقتی از ثباتها استفاده کنیم. برای این کار ابتدا کد برای متغیرهای موقتی ایجاد کرده و سپس در تبدیل به جای هر متغیر موقتی ثبات قرار می دهیم . در این اینجا دو سؤال پیش می آید :

(۱) تعداد ثباتها محدود است .

(۲) چه وقت نیاز به آن ثباتی که استفاده شده است نداریم؟

جواب (۱) فرض کنید ما دو ثبات داریم، داخل یکی مقدار T1 را قرار داده ایم و داخل دیگری T2 را. حال به یک ثبات دیگر نیاز داریم، می توانیم بسته به قراردادی که برای خودمان در نظر می گیریم یکی از ثباتها را برای انجام کارمان خالی کنیم یعنی مقدارش را در یک متغیر محلی به نام X1 قرار دهیم . سپس از آن ثبات استفاده کنیم در اینجا ما از FIFO استفاده می کنیم ، یعنی اینکه آن ثباتی که دارای شماره کمتری از نظر الویت است مقدارش را در متغیر موقتی X1 قرار می دهیم که خواهیم داشت :

ثبات	الویت
Ax -> T1	1
Bx -> T2	2
X1 -> Ax	
Ax -> T3	3

الویت Ax کمتر است :

برای انجام این کار به صورت عملی به رکوردهای زیر نیاز داریم :

رکورد RegTableRow :

(۱) نام ثبات (RegName) --> از نوع String

(۲) نام متغیر (VarName) --> از نوع String

(۳) شماره الویت (Turm) --> از نوع Integer

(۴) آیا ثبات در حال استفاده هست ؟ (Turn) --> از نوع Boolean

برای هر متغیر که به آن یک ثبات اختصاص میدهیم رکورد بالا را پر میکنیم ، برای مثال :

Ax -> T1

خواهیم داشت :

RegName = Ax
VarName = T1
Turn = 1
Busy = True

برای زمانی که ثابت کم می آوریم ، باید يك ثابت را خالی کنیم . يك رکورد دیگر باید داشته باشیم .

رکورد TempTableRow :

(۱) نام متغیر موقتی (TempName) -- از نوع String

(۲) نام متغیر یا ثابت (varName) -- از نوع String

به مثال زیر توجه کنید :

(a+b*c) * (d+c*f)

Mov t1,a $\xrightarrow{1}$

Mov Ax,a

Mov t2,b $\xrightarrow{2}$

Mov Bx, b

Mul t2,c $\xrightarrow{3}$

Mul Bx,c

Add t1,t2 $\xrightarrow{4}$ RemTemp

Add Ax,Bx

Mov t2,d $\xrightarrow{5}$

Mov Bx,d

Mov t3,c $\xrightarrow{6}$

{
Mov X1,Ax
Mov Ax,c

Mul t3,f $\xrightarrow{7}$

Mul Ax,f (for level 7)

Add t2,t3 $\xrightarrow{8}$

Add Bx,Ax

Mul t1,t2 $\xrightarrow{9}$

{
Mov Ax,X1

در مورد کد بالا به موارد زیر توجه کنید :

(۱) ما فقط دو ثابت داریم : AX,BX

(۲) به انجام مراحل زیر توجه کنید :

مرحله ۱ :

Regname=AX
Varname=T1
Turn=1
Busy=True

داخل جدول ثباتها و متغیرها را جستجو میکنیم .

اگر به T1 ثابت اختصاص نداده باشیم , به آن

ثبات اختصاص می دهیم.

مرحله ۲ :

Regname=BX
Varname=T2
Turn=2
Busy=True

داخل جدول ثباتها و متغیرها را جستجو میکنیم ،
اگر به T1 ثبات اختصاص نداده باشیم ، به آن
ثبات اختصاص می دهیم.

مرحله ۳ :

در داخل جدول ثباتها جستجو می کنیم چون به T2 یکبار ثبات داده شده است از همان در این مرحله استفاده میکنیم

مرحله ۴ :

در داخل جدول ثباتها جستجو می کنیم چون به T2 ، T1 یکبار ثبات داده شده است از همان در این مرحله استفاده میکنیم .

نکته : وقتی یک متغیر در سمت راست آورده می شود در هر مرحله ای که باشیم آن متغیر را در داخل جدول ثباتها جستجو می کنیم اگر آن متغیر دارای یک ثبات باشد ، اشغالی آن ثبات را False می کنیم چون دیگر نیازی به آن نیست ، چون مقدارش در متغیر سمت چپ ریخته شده است (کاملاً مثل قبل عمل می کنیم در قبل ما Remteoup می کردیم) .

باید یک رجیستر پیدا کنیم که خالی باشد تا مقدار D را در آن قرار دهیم داخل جدول رجیستر جستجو می کنیم چون Busy ، bx اش False است می یوانیم از این ثبات برای قرار دادن d استفاده کنیم .
باید داخل جدول ثباتها جستجو کنیم تا یک ثبات خالی پیدا کنیم ولی چون دیگر ثبات خالی دیگری وجود ندارد مجبوریم یکی از ثباتها را به یک متغیر موقتی بریزیم و از آن ثبات خالی شده استفاده کنیم ، برای این کار به صورت زیر عمل می کنیم .

داخل جدول متغیرها (1)

Temp name = x1
Vov name = T1
X1 ← ax

(2) **Reg name = ax**

Vav name = T3
Turn = 3
Busy = True

مانند قبل عمل می کنیم داخل جدول ثباتها چون به T3 یک بار ثبات اختصاص داده شده است دیگر برای آن ثبات جدیدی در نظر نمی گیریم و عمل ضرب را انجام می دهیم .

عمل جم ++ انجام می شود ولی چون ax در سمت راست قرار دارد دیگر نیازی به آن نیست و (Busy = False) اشغال آنرا برابر False قرار می دهیم (یعنی می توان از این استفاده کرد) .

در مرحله (q) باید T1, T2 را جستجو کنیم T2 را در جدول مربوط به ثباتها پیدا کرده و مقدارش که bx را در نظر می گیریم ولی T1 در جدول ثباتها وجود ندارد پس جدول مربوط به متغیرها را نیز جستجو می کنیم آنرا پیدا می کنیم

می فهمیم مقدارش در x_1 وجود دارد آنرا داخل يك ثابت خالی قرار می دهیم (اگر وجود نداشت يك ثابت را خالی می کنیم) و سپس عمل Mul را انجام می دهیم .