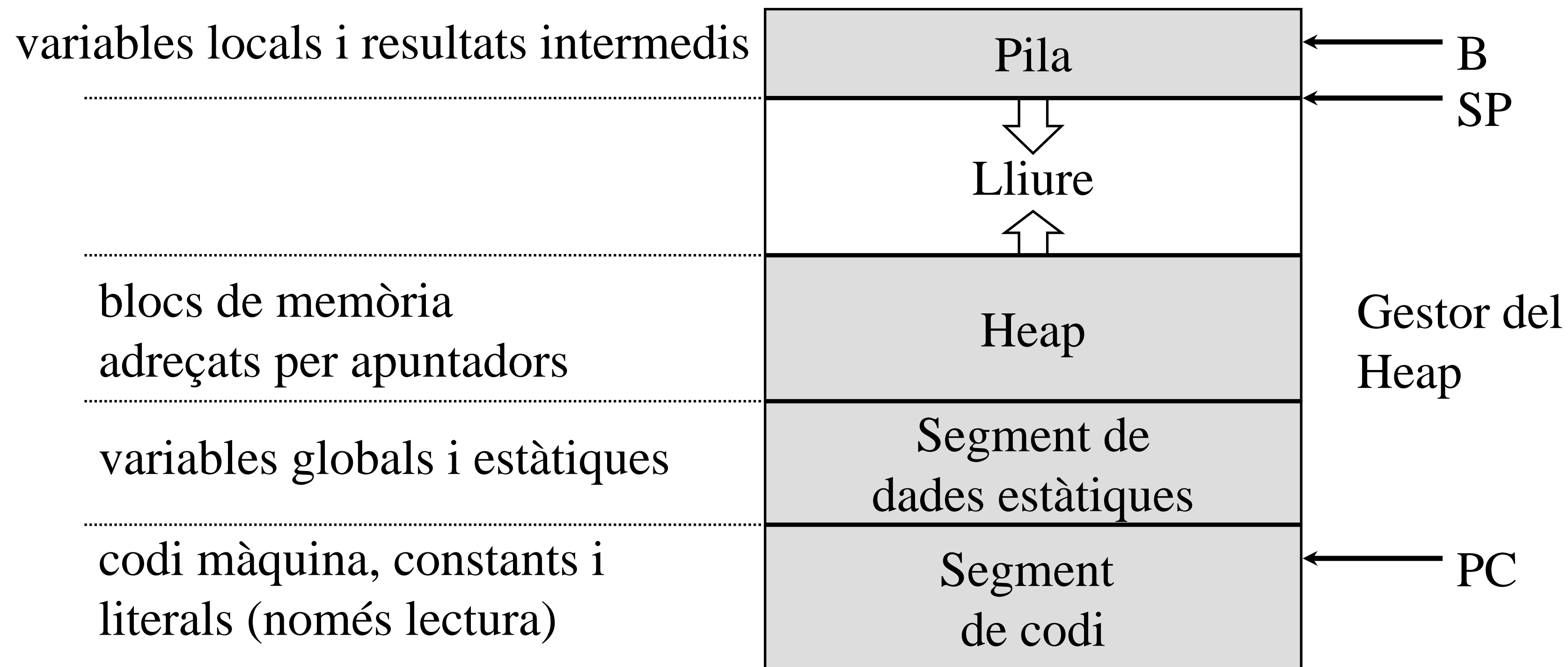
TEMA 6.
GENERACIÓ DE CODI

Lliçons 27,28,29,30

Màquina Abstracta per l'Execució de Programes Imperatius



- La màquina que considerarem no tindrà Heap
- Registres
  - B: Base del bloc d'activació
  - SP: Apuntador al final de la Pila (stack pointer)
  - PC: Comptador del programa (program counter)

Característiques

- Cada instrucció només pot referir-se a una única adreça de memòria.
- Totes les operacions es realitzen a la pila.
- Els tipus de dades que suporta són
  - Apuntadors (32 bits)
  - Sencers (32 bits)
  - Flotants (64 bits)
  - Caràcters (8 bits)
- La màquina suporta l’ús de display's
- Hi ha adreçament indirecte per poder implementar arrays i apuntadors.

Codificació de les instruccions

- Cada instrucció te un codi d'un byte
- Els paràmetres de la instrucció segueixen el codi de instrucció per ordre d'aparició
  - caràcter 1 byte
  - sencer 4 bytes
  - apuntador 4 bytes
  - real 8 bytes
- Exemple
  - Instrucció

    IPushBVar 4 8
  - Codificació

1	IPushBVar
4 0 0 0	4
8 0 0 0	8

Posar un valor a la pila

- Afegir al final de la pila un literal

IPushLit mida literal

```
var sz=Memoria.DWord(pc+1);
sp=sp-sz;
for (i<-0..sz-1) Memoria.Byte(sp+i)=Memoria.Byte(pc+5+i);
pc=pc+5+sz;
```

- Afegir al final de la pila el contingut d'una adreça de memòria (normalment variables globals).

IPushGVar mida adreça

```
var sz=Memoria.DWord(pc+1);
var s=Memoria.DWord(pc+5);
sp=sp-sz;
for (i<-0..sz-1) Memoria.Byte(sp+i)=Memoria.Byte(s+i);
pc=pc+9;
```

Posar un valor a la pila

- Afegir al final de la pila el contingut de una part del bloc de activació (normalment variable local)

  IPushBVar mida desplaçament

```
var sz=Memoria.DWord(pc+1);
var s=B+Memoria.DWord(pc+5);
sp=sp-sz;
for (i<-0..sz-1) Memoria.Byte(sp+i)=Memoria.Byte(s+i);
pc=pc+9;
```

- Afegir al final de la pila el contingut de una part de un bloc de activació a partir del display

  IPushDispVar mida nivell desplaçament

```
var sz=Memoria.DWord(pc+1);
var s=Memoria.DWord(B+Memoria.DWord(pc+5))+
        Memoria.DWord(pc+9);
sp=sp-sz;
for (i<-0..sz-1) Memoria.Byte(sp+i)=Memoria.Byte(s+i);
pc=pc+13;
```

Posar un valor a la pila

- Afegir al final de la pila el contingut de una adreça de memòria que es troba a al final de la pila (accés a arrays i apuntadors)

IPushInd mida

```
var sz=Memoria.DWord(pc+1);
var s=Memoria.DWord(sp);
sp=sp-sz+4;
for (i<-0..sz-1) Memoria.Byte(sp+i)=Memoria.Byte(s+i);
pc=pc+5;
```

Treure un valor de la pila

- Treure el valor del final la pila i guardar-lo en una adreça de memòria (normalment variable global).

IPopGVar mida adreça

```
var sz=Memoria.DWord(pc+1);
var d=Memoria.DWord(pc+5);
for (i<-0..sz-1) Memoria.Byte(d+i)=Memoria.Byte(sp+i);
sp=sp+sz;
pc=pc+9;
```

- Treure el valor del final la pila i guardar-lo en el bloc d'activació (normalment variable local).

IPopBVar mida desplaçament

```
var sz=Memoria.DWord(pc+1);
var d=B+Memoria.DWord(pc+5);
for (i<-0..sz-1) Memoria.Byte(d+i)=Memoria.Byte(sp+i);
sp=sp+sz;
pc=pc+9;
```

Treure un valor de la pila

- Treure el valor del final la pila i guardar-lo en el bloc d'activació referenciat des de el display (normalment variable local).

IPopDispVar mida nivell desplaçament

```
var sz=Memoria.DWord(pc+1);
var d=Memoria.DWord(B+Memoria.DWord(pc+5))+
          Memoria.DWord(pc+9);
for (i<-0..sz-1) Memoria.Byte(d+i)=Memoria.Byte(sp+i);
sp=sp+sz;
pc=pc+13;
```

Treure un valor de la pila

- Treure el valor antepenúltim de la pila i guardar-lo en una adreça de memòria que es troba a al final de la pila (accés a arrays i apuntadors)

IPopInd mida

```
var sz=Memoria.DWord(pc+1);
var d=Memoria.DWord(sp);
sp=sp+4;
for (i<-0..sz-1) Memoria.Byte(d+i)=Memoria.Byte(sp+i);
sp=sp+sz;
pc=pc+5;
```

Copiar un valor de la pila

- Copiar el valor del final la pila en una adreça de memòria (normalment variable global).

  IStoreGVar mida adreça

  ```
  var sz=Memoria.DWord(pc+1);
  var d=Memoria.DWord(pc+5);
  for (i<-0..sz-1) Memoria.Byte(d+i)=Memoria.Byte(sp+i);
  pc=pc+9;
  ```

- Copiar el valor del final la pila en el bloc d'activació (normalment variable local).

  IStoreBVar mida desplaçament

  ```
  var sz=Memoria.DWord(pc+1);
  var d=B+Memoria.DWord(pc+5);
  for (i<-0..sz-1) Memoria.Byte(d+i)=Memoria.Byte(sp+i);
  pc=pc+9;
  ```

Copiar un valor de la pila

- Copiar el valor del final la pila en el bloc d'activació referenciat des de el display (normalment variable local).

IStoreDispVar mida nivell desplaçament

```
var sz=Memoria.DWord(pc+1);
var d=Memoria.DWord(B+Memoria.DWord(pc+5))+
          Memoria.DWord(pc+9);
for (i<-0..sz-1) Memoria.Byte(d+i)=Memoria.Byte(sp+i);
pc=pc+13;
```

Copiar un valor de la pila

- Copiar el valor antepenúltim de la pila en una adreça de memòria que es troba a al final de la pila (accés a arrays i apuntadors)

IStoreInd mida

```
var sz=Memoria.DWord(pc+1);
var d=Memoria.DWord(sp);
sp=sp+4;
for (i<-0..sz-1) Memoria.Byte(d+i)=Memoria.Byte(sp+i);
pc=pc+5;
```

Posar una adreça a la pila

- Afegir al final de la pila una adreça de memòria (normalment de variable global).

IPushAddressGVar adreça

```
var s=Memoria.DWord(pc+1);
sp=sp-4;
Memoria.DWord(sp)=s;
pc=pc+5;
```

Posar una adreça a la pila

- Afegir al final de la pila la adreça de una part del bloc de activació (normalment variable local)

  IPushAddressBVar desplaçament

  ```
  var s=B+Memoria.DWord(pc+1);
  sp=sp-4;
  Memoria.DWord(sp)=s;
  pc=pc+5;
  ```

- Afegir al final de la pila la adreça de una part de un bloc de activació a partir del display

  IPushAddressDispVar mida nivell desplaçament

  ```
  var s=Memoria.DWord(B+Memoria.DWord(pc+1))+
          Memoria.DWord(pc+5);
  sp=sp-4;
  Memoria.DWord(sp)=s;
  pc=pc+9;
  ```

Crida a funcions

- Crida a una funció

ICall adreça

```
sp=sp-4;
Memoria.DWord(sp)=pc+5;
pc=Memoria.DWord(pc+1);
```

- Retorn de funció

IRet

```
pc=Memoria.DWord(sp);
sp=sp+4;
```

Crida a funcions

- Reserva el espai de variables

lLink mida variables

```
sp=sp-4;
Memoria.DWord(sp)=b;
b=sp;
sp=sp-Memoria.DWord(pc+1);
pc=pc+5;
```

- Eliminar el espai de variables

lUnlink

```
sp=b+4;
b=Memoria.DWord(b);
pc=pc+1;
```

Crida a funcions

- Guardar B a la pila per construir un Display

IPushB

```
sp=sp-4;
Memoria.DWord(sp)=b;
pc=pc+1;
```

- Reserva espai per el valor de retorn o eliminar els arguments de la pila

IAddSP mida

```
sp=sp+Memoria.DWord(pc+1);
pc=pc+5;
```

Control de flux

- Salt incondicional

IJmp adreça

```
pc=Memoria.DWord(pc+1);
```

- Salt condicional si veritat

IJmpTrue adreça

```
if (Memoria.DWord(sp)==0) pc=pc+5;
else pc=Memoria.DWord(pc+1);
sp=sp+4;
```

- Salt condidcional si fals

IJmpFalse adreça

```
if (Memoria.DWord(sp)!=0) pc=pc+5;
else pc=Memoria.DWord(pc+1);
sp=sp+4;
```

Operacions aritmètiques

- Suma de sencers

IAddInt

```
Memoria.DWord(sp+4)=Memoria.DWord(sp+4)+
          Memoria.DWord(sp);
sp=sp+4;
pc=pc+1;
```

- Suma de reals

IAddReal

```
Memoria.Double(sp+8)=Memoria.Double(sp+8)+
          Memoria.Double(sp);
sp=sp+8;
pc=pc+1;
```

- Resta **ISubInt ISubReal**
- Producte **IMultInt IMultReal**
- Divisió **IDivInt IDivReal**

Canvi de signe

- Canvi de signe de sencers

INegInt

```
Memoria.DWord(sp)=-Memoria.DWord(sp);
pc=pc+1;
```

- Canvi de signe de reals

INegReal

```
Memoria.Double(sp)=-Memoria.Double(sp);
pc=pc+1;
```

Comparacions

- Comparació d'igualtat

IEqChar

```
Memoria.DWord(sp-2)=
        if (Memoria.Byte(sp+1)==Memoria.Byte(sp)) 1 else 0;
sp=sp-2;
pc=pc+1;
```

IEqInt

```
Memoria.DWord(sp+4)=
        if (Memoria.DWord(sp+4)==Memoria.DWord(sp)) 1 else 0;
sp=sp+4;
pc=pc+1;
```

IEqReal

```
Memoria.DWord(sp+12)=
        if (Memoria.Double(sp+8)==Memoria.Double(sp)) 1 else 0;
sp=sp+12;
pc=pc+1;
```

Comparacions

- Comparació desigualtat
  INEqChar INEqInt INEqReal
- Comparació de menor
  ILessChar ILessInt ILessReal
- Comparació de menor o igual
  ILessEqChar ILessEqInt ILessEqReal
- Comparació de major
  IGreaterChar IGreaterInt IGreaterReal
- Comparació de major o igual
  IGreaterEqChar IGreaterEqInt IGreaterEqReal

Operacions booleanes

- Negació

INot

```
Memoria.DWord(sp)=if (Memoria.DWord(sp)==0) 1 else 0;
pc=pc+1;
```

- I lògic

IAnd

```
Memoria.DWord(sp+4)=if (Memoria.Double(sp+4)!=0 &&
        Memoria.Double(sp)!=0) 1 else 0;
sp=sp+4;
pc=pc+1;
```

- O lògic

IOr

```
Memoria.DWord(sp+4)= if (Memoria.Double(sp+4)!=0 ||
        Memoria.Double(sp)!=0) 1 else 0;
sp=sp+4;
pc=pc+1;
```

Coercions

- Coerció de caràcter a sencer

ICharToInt

```
Memoria.DWord(sp-3)=Memoria.Byte(sp);
sp=sp-3;
pc=pc+1;
```

- Altres coerció

IIntToReal

IIntToChar

IRealToInt

I/O i Exit

- Lecture de la entrada **IReadChar IReadInt IReadReal**

IReadChar

```
sp=sp-1;
Memoria.Byte(sp)=Coerce(Int,cin.get());
pc=pc+1;
```

- Impressió a la sortida **IPrintChar IPrintInt IPrintReal**

IPrintChar

```
cout.PrintLn(Coerce(char,Memoria.Byte(sp)));
sp=sp+1;
pc=pc+1;
```

- Finalitzar l'execució

IExit

```
throw Exception("Ejecución finalizada correctamente");
```

Codi de Fib

```
function Fib(n:integer):integer
begin
    if n<=2 then return 1;
    else return fib(n-1)+fib(n-2);
end
```

valor de ret	B+12
n	B+8
PC	B+4
ED	B+0

```
fib:      ILink 0
          IPushBVar 4 8
          IPushLit 4 2
          ILessEqInt
          IJmpFalse L_Else3
          IPushLit 4 1
          IPopBVar 4 12
          IUnlink
          IRet
          IJmp L_FiIf4
```

```
L_Else3:  IAddSP -4
          IPushBVar 4 8
          IPushLit 4 1
          ISubInt
          ICall fib
          IAddSP 4
          IAddSP -4
          IPushBVar 4 8
          IPushLit 4 2
          ISubInt
          ICall fib
          IAddSP 4
          IAddInt
          IPopBVar 4 12
          IUnlink
          IRet
L_FiIf4:  IUnlink
          IRet
```

Codi de Factorial

```
function Fac(n:integer):integer
begin
    var i:integer;
    var r:integer;
    i:=1; r:=1;
    while i<=n do
    begin
        r:=r*i; i:=i+1;
    end
    return r;
end
```

valor de ret	B+12
n	B+8
PC	B+4
ED	B+0
i	B-4
r	B-8

```
fac:      ILink 8
          IPushLit 4 1
          IPopBVar 4 -4
          IPushLit 4 1
          IPopBVar 4 -8
L_Rep5:   IPushBVar 4 -4
          IPushBVar 4 8
          ILessEqInt
          IJmpFalse L_FiW6
          IPushBVar 4 -8
          IPushBVar 4 -4
          IMultInt
          IPopBVar 4 -8
          IPushBVar 4 -4
          IPushLit 4 1
          IAddInt
          IPopBVar 4 -4
          IJmp L_Rep5
L_FiW6:   IPushBVar 4 -8
          IPopBVar 4 12
          IUnlink
          IRet
          IUnlink
          IRet
```

Taula de Símbols i Processat de Declaracions

- Taula de Símbols
  - Entrada d'una Variable a la Taula de símbols
    - Nom
    - Tipus
    - On es guarda:
      - Adreça de memòria si és variable global (etiqueta)
      - Desplaçament si és variable local
  - Afegir el càlcul del nivell de l'àmbit a les cerques a la taula de símbols
- Processat de declaracions
  - Afegir el càlcul de l'adreça de memòria o desplaçament de la variable
  - Afegir a l'àmbit de la taula de símbols
    - És global: Mida del segment estàtic o adreça simbólica
    - És local: Mida del bloc d'activació

Modificacions de les entrades de la taula de símbols

```
Type ETSVariable(
      Tipus, // tipus de la variable
      Posicio // etiqueta/desplaçament
   ) from EntradaTaulaSimbols

Type ETSFuncio(
      Parametres, // Llista de paràmetres
      TipusRetorn, // tipus del valor de retorn (resultat de la funció o
               // void per procediment)
      DespRetorn, // Desplaçament del valor de retorn en el bloc d'activació
      Posicio     // etiqueta que apunta a l'inici del codi de la funció
   ) from EntradaTaulaSimbols

Type TSSeparadorFuncio(
    Funcio,
    VarSz) from TSSeparador; // Separador d'àmbits de funció
```

Modificació de la funció de cerca

```
Const Global;

Fun ts:TaulaDeSimbols.Buscar(nom:string)=>
{
   var nivell=0;
   for (ets<-ts.pila) {
      if (typep(EntradaTaulaDeSimbols,ets)) {
         if (ets.nom==nom) return ets,nivell;
      }
      else if (typep(TSSeparadorFuncio,ets)) {
         if (ets.Funcio==Unbound) nivell=Global;
         else nivell=nivell+1;
      }
   }
   throw Exception(nom," no declarat");
}
```

Càlcul de la mida dels tipus de dades

```
Fun MidaTipus(t)=>
{
    switch (t)
    {
        TipusInt => 4;
        TipusReal => 8;
        TipusChar => 1;
        TipusArray => t.Mida*MidaTipus(t.TipusElements);
        TipusApuntador => 4;
        TipusEstructura => foldl(\+,[MidaTipus(c.tipus) | c<-t.Camps]);
    }
}
```

Cues per guardà el codi generat

- El codi es genera en dos passos
  - Primer es guarda el codi a una o més cues que s'ajuntaran abans de generar el codi final. Això permetrà canviar l'ordre de la seqüència de codi.
    - La cua on es juntará tot el codi es CodiGlobal
    - A cada regla BNF se li passa la cua (Cod) on ha de generar el codi o deixarà el códi directament a CodiGlobal
  - Passar el codi de la cua al seu destí final i enllaçar-lo
- Operacions amb cues
  - Crear una nova cua: Queue()
  - Afegir una instrucció al final de la cua: cod.put(instrucció)
  - Afegir a una cua el contingut d'una altre: cod1.PutElements(cod2);
  - Bolcar el codi al destí final en memòria: Bolcar(cod)

Generació de Codi d'Expressions

- Generació de codi simplificada
  - Falten els tipus de dades
  - Falta diferenciar entre variables locals i globals

```
<exp>::= <ter> {+ <ter> @codi[IAdd?]; |
                - <ter> @codi[ISub?];}
<ter>::=<fac> {* <fac>   @codi[IMul?]; |
                / <fac> @codi[IDiv?]; }
<fac>::=-<fac> @codi[INeg?]; |
          "(" <exp> ")" |
          número#(v) @codi[IPushLit? v]; |
          identificador#(nom) @codi[IPush? Acces(nom)];
```

Generar el Codi d'Accés a Variables

- Modificacions a la taula de símbols
  - Una variable global indica si l'àmbit actual és local o global. Es guarda i recupera de la TS quan es creen i eliminen àmbits.
  - Les variables tenen un nou atribut que especifica el seu desplaçament respecte a B o la seva adreça.
  - Les cerques d'un símbol retornen la seva entrada i el número d'àmbits locals que s'han travessat (calcula la diferència de nivell)
- Amb les modificacions anteriors ja podem saber si s'ha de generar

  PushGVar? *adreça*

  PushBVar? *desp*

  PushDispVar? *nivell,desp*

Considerar dels Tipus de Dades i Coercions

Codi per la suma de dos termes de tipus sencer

<ter1> + <ter2>

Codi[<ter1>]

Codi[<ter2>]

AddInt

Codi per la suma d'un terme sencer amb un flotant

<ter1> + <ter2>

Codi[<ter1>]

IntToFloat

Codi[<ter2>]

AddFloat

Problema: Per generar la coerció és necessari saber el tipus de **<ter2>** i aquest no es coneix fins després de generar el seu codi.

Solució: Guardar el codi de **<ter2>** a una cua fins saber el seu tipus.

Generació de Codi amb més d'una cua

El codi es genera en dos passos

- Primer es guarda el codi a una o més cues que s'ajuntaran abans de generar el codi final. Això permetrà canviar l'ordre de la seqüència de codi.
  - A cada regla BNF se li passa la cua on ha de generar el codi
- Passar el codi de la cua al seu destí final.

Operacions

- Crear una nova cua de codi: `Queue()`
- Afegir a una cua el contingut d'una altre: `cod1.PutElements(cod2);`
- Bolcar el codi al destí final: `Bolcar(cod)`

Generació de Codi d'Expressions amb Coercions

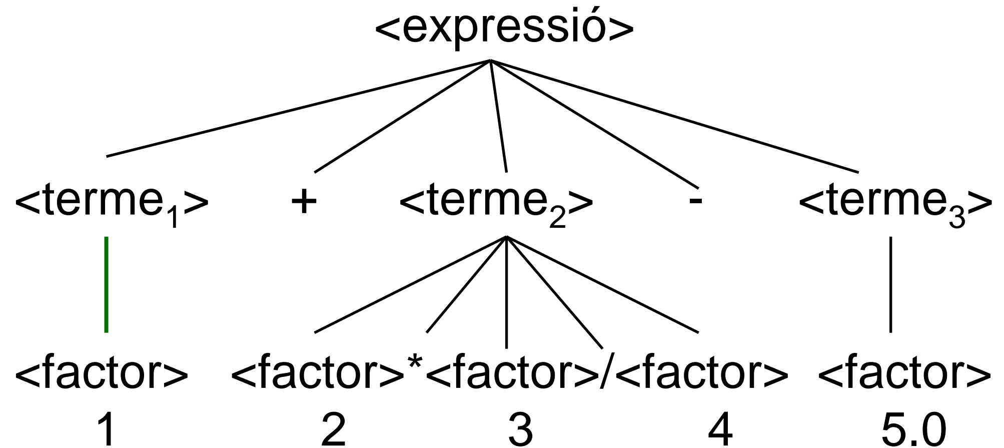
```
Rule <terme(cod,&t,Resultat)>::= @Var t2; <factor(cod,t,Resultat)> {
  @if (!Resultat) throw Exception("No s'aprofita el resultat");
  @Var cod2=queue(); (
    * <factor(cod2,t2,Resultat)>  @switch ((t,t2)) {
       (TInt,TInt) => {  t=TInt; Cod2.Put(IMultInt); }
       (TInt,TReal) => { t=TReal; Cod.Put(IIntToReal); Cod2.Put(IMultReal); }
       (TReal,TInt) => { t=TReal; Cod2.Put(IIntToReal, IMultReal); }
       (TReal,TReal) => { t=TReal; Cod2.Put(IMultReal);   }
       Others => throw Exception("Error de tipus de dades en resta ",t,t2); };
    | / <factor(cod2,t2,Resultat)>  @switch ((t,t2)) {
       (TInt,TInt) => { Cod.Put(IIntToReal); Cod2.Put(IIntToReal);  }
       (TInt,TReal) => { Cod.Put(IIntToReal);  }
       (TReal,TInt) => { Cod2.Put(IIntToReal);}
       (TReal,TReal) => {}
       Others => throw Exception( "Error de tipus de dades en divisio ",t,t2); };
    @t=TReal;
    @Cod2.Put(IDivReal);
  ) @cod.PutElements(cod2); }
```

Exemple de generació de codi per expressions

- Font: Print 1+2*3/4-5.0;

Codi generat

```
IPushLit 4 1      // <factor> 1
IIntToReal        // <expressió>
IPushLit 4 2      // <factor> 2
IPushLit 4 3      // <factor> 3
IMultInt          // <terme2>
IIntToReal        // <terme2>
IPushLit 4 4      // <factor> 4
IIntToReal        // <terme2>
IDivReal          // <terme2>
IAddReal          // <expressió>
IPushLit 8 5.0    // <factor> 5.0
ISubReal          // <expressió>
IPrintReal
```



Accés a variables i càlcul d'Adreces

- El càlcul d'adreces ha d'aprofitar las capacitats d' adreçament de la màquina. Exemple: a.camp
  - Sense aprofitar

    IPushAddressBVar *desp a*

    IPushLit 4 *desp camp*

    IAddInt

    IPushInd 4
  - Aprofitant

    IPushBVar 4 *desp a*+*desp camp*
- El resultat final del càlcul d'adreces depèn de si es vol fer
  - una lectura
  - una escriptura
- Es sap de quin cas es tracta al final de la generació (hi ha =<expressió> o no)

Representació d'un càlcul d'adreça

- Adreça de memoria: Type RefEtiqueta(eti,Desp)
- Adreça del bloc d'activació: (nivell,desplaçament)
  - Si el nivell es 0 s'accedeix a partir de B
  - Si no s'accedeix a partir del display
- Adreça calculada en temps d'execució
  - Es representa amb una cua que conté el codi que la calcula i deixa l'adreça al final de la pila
- Exemple var a:???; (ambit global)
  - a: RefEtiqueta(a,0)
  - a.camp: RefEtiqueta(a,*desplaçament camp*)
  - a[i]: cua amb el següent codi

    IPushAddressGVar RefEtiqueta(a,0)
    IPushBVar 4 i
    IPushLit 4 *mida elements de l'array*
    MultInt
    AddInt

Accés a variables

```
Rule <factor(cod,&t,Resultat)>::= @var v,nom,ets,nivell; ( ...
   | Numero#(v)
    @t=if (typeP(int,v)) TInt else TReal;
    @if (!Resultat) throw Exception("No s'aprofita el resultat");
    @Cod.put(IPushLit,MidaTipus(t),v);
   | Identificador#(nom)
    @|ets,nivell|=TS.Buscar(nom);
    ( crida a funció |
       @if (!TypeP(ETSVariable,ets))
          throw Exception(nom," no es variable");
       @var texp;
       @var Acc= if (nivell==Global) RefEtiqueta(ets.Posicio)
                 else (Nivell,ets.Posicio);
       <Acces(Acc,ets.Tipus,t)> (
          = <Expressio(cod,texp,true)> escriptura variable
          | lectura variable $
   )))
```

Lectura d'una variable

```
Identificador#(nom)
 @|ets,nivell|=TS.Buscar(nom);
 @if (!TypeP(ETSVariable,ets))
    throw Exception(nom," no es variable");
@var Acc=if (nivell==Global) RefEtiqueta(ets.Posicio)
                   else (if (Nivell==0) 0 else 4+Nivell*4,ets.Posicio);
<Acces(Acc,ets.Tipus,t)>
@if (!Resultat) throw Exception("No s'aprofita el resultat");
@switch (Acc) {
   RefEtiqueta => cod.Put(IPushGVar,MidaTipus(t),Acc);
   Vector =>
      if (Acc[0]==0) cod.Put(IPushBVar,MidaTipus(t),Acc[1]);
      else cod.Put(IPushDispVar,MidaTipus(t),Acc[0],Acc[1]);
   Queue => {
      cod.PutElements(Acc);
      cod.Put(IPushInd,MidaTipus(t));
   }
};
```

Exemple de lectura de variables

Codi font

```
VAR global:integer;
PROCEDURE access()
BEGIN
   VAR local1:integer;
   PROCEDURE subaccess()
   BEGIN
      VAR local2:integer;
      print global;
      print local1;
      print local2;
   END
   print global;
   print local1;
END
```

Codi generat

```
global:       ESPAI 4 Bytes
subaccess:    ILink 4
              IPushGVar 4 global
              IPrintInt
              IPushDispVar 4 8 -4
              IPrintInt
              IPushBVar 4 -4
              IPrintInt
              IUnlink
              IRet
access:       ILink 4
              IPushGVar 4 global
              IPrintInt
              IPushBVar 4 -4
              IPrintInt
              IUnlink
              IRet
```

Assignació d'una variable

```
Identificador#(nom)
@|ets,nivell|=TS.Buscar(nom);
@if (!TypeP(ETSVariable,ets)) throw Exception(nom," no es variable");
@var texp;
@var Acc= if (nivell==Global) RefEtiqueta(ets.Posicio)
                else (if (Nivell==0) 0 else 4+Nivell*4,ets.Posicio);
 <Acces(Acc,ets.Tipus,t)> = <Expressio(cod,texp,true)>
@if (t!:=texp) throw Exception("Error de tipus de dades en =");
@switch (Acc) {
   RefEtiqueta => cod.Put(if (resultat) IStoreGVar else IPopGVar, MidaTipus(t),Acc);
   Vector =>  if (Acc[0]==0) cod.Put(if (resultat) IStoreBVar else IPopBVar,
                                                    MidaTipus(t), Acc[1]);
      else cod.Put(if (resultat) IStoreDispVar else IPopDispVar,
                         MidaTipus(t),Acc[0],Acc[1]);
   queue => {
      cod.PutElements(Acc);
      cod.Put(if (resultat) IStoreInd else IPopInd,MidaTipus(t));
    }
};
```

Exemple d'assignació de variables

Codi font

```
VAR global:integer;
PROCEDURE assignacio()
BEGIN
    VAR local1:integer;
    PROCEDURE subassignacio()
    BEGIN
        VAR local2:integer;
        global=1;
        local1=2;
        local2=3;
    END
    global=1;
    local1=2;
    print global=5;
END
```

Codi generat

```
subassignacio: ILink 4
               IPushLit 4 1
               IPopGVar 4 global
               IPushLit 4 2
               IPopDispVar 4 8 -4
               IPushLit 4 3
               IPopBVar 4 -4
               IUnlink
               IRet
assignacio:    ILink 4
               IPushLit 4 1
               IPopGVar 4 global
               IPushLit 4 2
               IPopBVar 4 -4
               IPushLit 4 5
               IStoreGVar 4 global
               IPrintInt
               IUnlink
               IRet
```

Càlcul d'adreces

```
Rule <Acces(&Acc,te,&tr)>::= @tr=te; {
   "[" <Expressio(Acc,ti,true)> "]" // Array
   |  . identificador#(nom)  // Camp d’estructura
   | ^  // Apuntador
}
```

Càlcul d'adreces per accés a array

```
Rule <Acces(&Acc,te,&tr)>::= @tr=te; {
   @var ti;
   @if (!typeP(TipusArray,tr)) throw Exception("Tipus erroni en array");
   @tr=tr.TipusElements;
   @switch (Acc) {
      RefEtiqueta => {
                 var q=Queue();
                 q.put(IPushAddressGVar,Acc);
                 Acc=q;  }
      Vector => {
         var q=Queue();
         if (Acc[0]==0) q.Put(IPushAddressBVar,Acc[1]);
         else q.Put(IPushAddressDispVar,Acc[0],Acc[1]);
         Acc=q;   }
      queue => {}
   };
   "[" <Expressio(Acc,ti,true)> "]"
   @if (ti!=TInt) throw Exception("Tipus erroni en index");
   @Acc.Put(IPushLit,4,MidaTipus(tr));
   @Acc.Put(IMultInt);
   @Acc.Put(IAddInt); | ... }
```

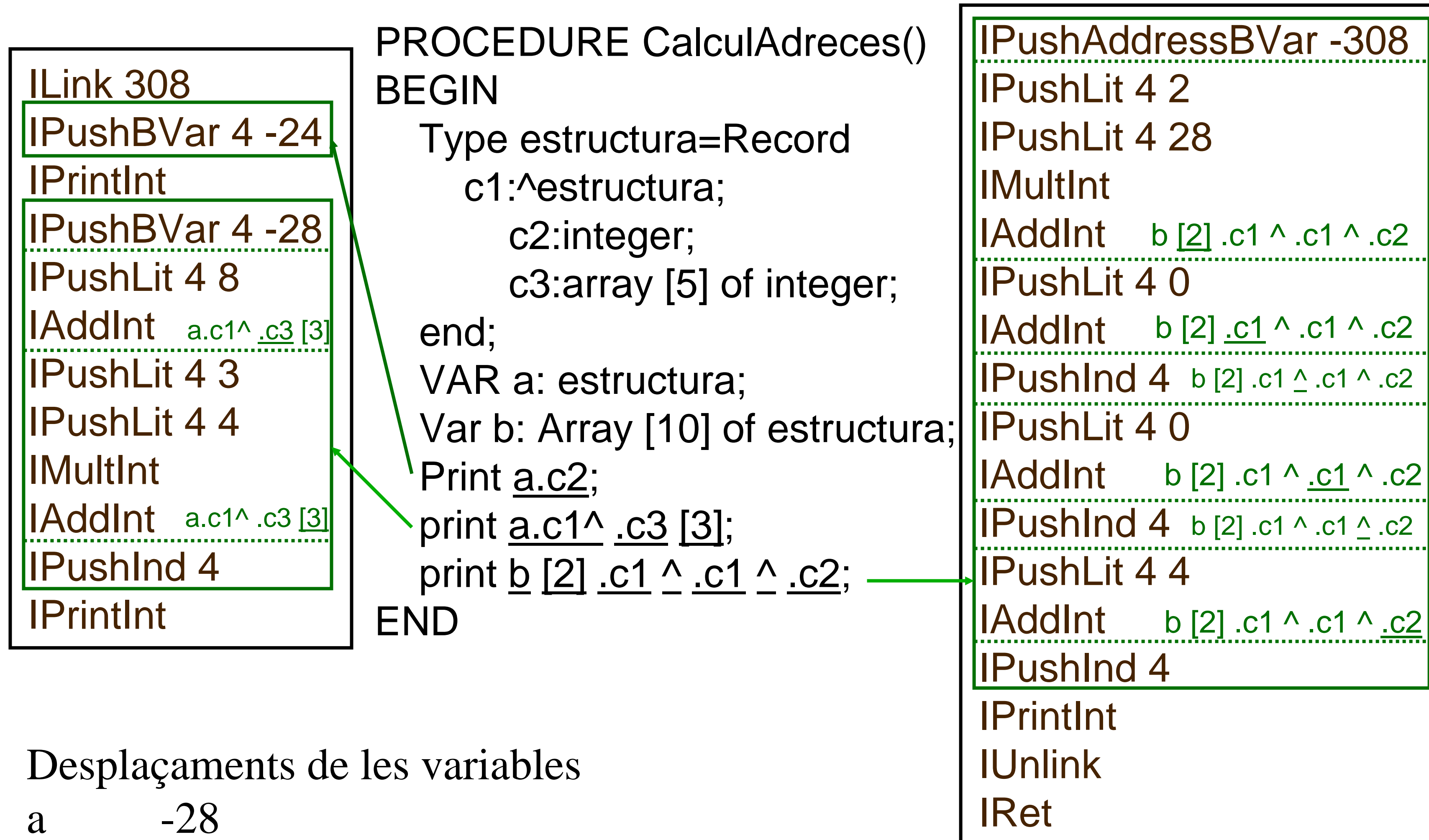
Càlcul d'adreces per accés a camp d'estructura

```
Rule <Acces(&Acc,te,&tr)>::= @tr=te; { ...
 | @var nom;
   . identificador#(nom)
   @if (!typeP(TipusEstructura,tr)) throw Exception("Tipus erroni en
    .camp");
   @Search (c<-tr.Camps,c.nom==nom) {
      tr=c.tipus;
      switch (Acc) {
         RefEtiqueta => Acc=RefEtiqueta(Acc.eti,Acc.Desp+c.Desp);
         vector => Acc=(Acc[0],Acc[1]+c.Desp);
         queue => {
            Acc.Put(IPushLit,4,c.Desp);
            Acc.Put(IAddInt);
         }
      }
   }
   else throw Exception("Camp ",nom," no declarat");
 | ... }
```

Càlcul d'adreces per accés a apuntador

```
Rule <Acces(&Acc,te,&tr)>::= @tr=te; { ...
  | ^
    @if (!TypeP(TipusApuntador,tr)) throw Exception("Tipus erroni en ^");
    @tr=tr.TipusBase;
    @switch (Acc) {
       RefEtiqueta => {
          var q=Queue();
          q.put(IPushGVar,4,Acc);
          Acc=q;
       }
       Vector => {
          var q=Queue();
          if (Acc[0]==0) q.Put(IPushBVar,4,Acc[1]);
          else q.Put(IPushDispVar,4,Acc[0],Acc[1]);
          Acc=q;
       }
       Queue => Acc.Put(IPushInd,4);
    };
}
```

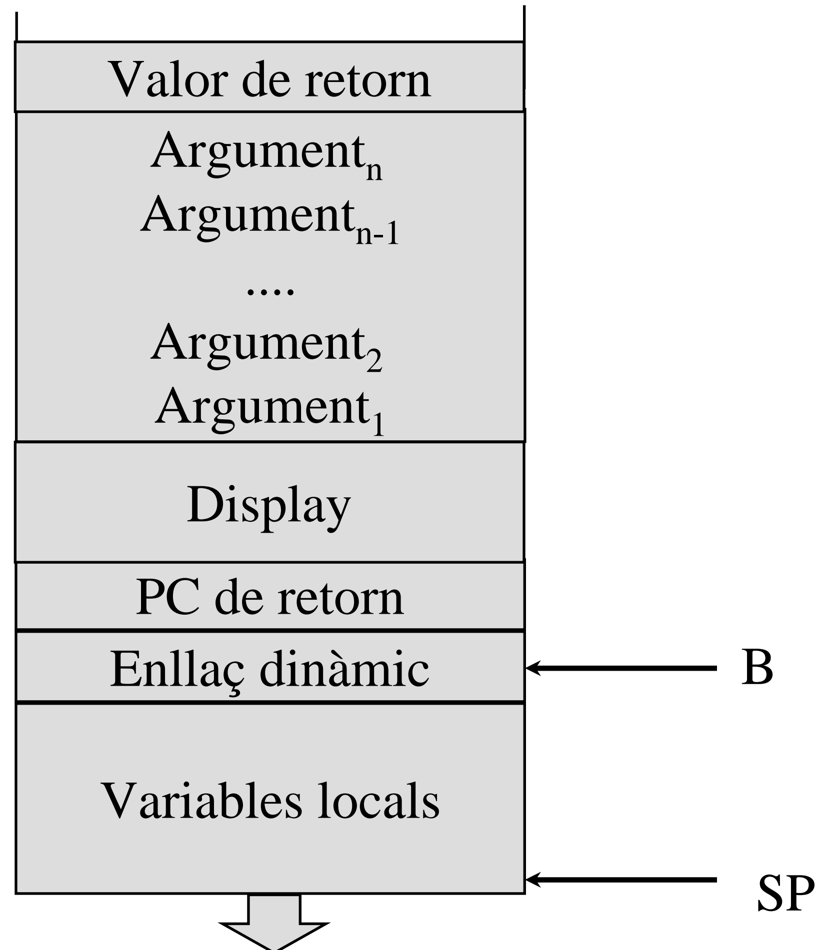
Exemples de codi de càlcul d'adreces



Desplaçaments de les variables

a	-28
b	-308

Bloc d'activació de funcions en LS



Crida a funció (I)

```
Identificador#(nom)
@|ets,nivell|=TS.Buscar(nom);
@if (!TypeP(ETSFuncio,ets)) throw Exception(nom," no es funció");
@if (Resultat) {
   if (ets.TipusRetorn=:=TVoid) throw Exception(
      "crida a procediment dintre d'expressió");
   }
   else {
      if (ets.TipusRetorn!:=TVoid) throw Exception(
         "No s'aprofita el resultat de la funció");
};
@if (ets.TipusRetorn!:=TVoid)
   cod.put(IAddSP,-MidaTipus(ets.TipusRetorn));
```

Crida a funció (II)

```
@var ltparams=ets.Parametres;
@var codisparams=[];
"(" [ @Var codparam=queue();
    <Expressio(codparam,t,true)>
   @if (ltparams==[]) throw Exception("masses paràmetres");
   @if (ltparams.Head!:=t) throw Exception("error de tipus");
   @ltparams=ltparams.tail;
   @codisparams=codparam::codisparams;
   { @Var codparam=queue();
    , <Expressio(codparam,t,true)>
      @if (ltparams==[]) throw Exception("masses paràmetres");
      @if (ltparams.Head!:=t) throw Exception("error de tipus");
      @ltparams=ltparams.tail;
      @codisparams=codparam::codisparams;
    }] ")"
@for (cp<-codisparams) cod.PutElements(cp);
@if (ltparams!=[]) throw Exception("pocs paràmetres");
@t=ets.TipusRetorn;
```

Crida a funció (III)

```
// Display
@if (nivell!=Global) {
   var NivellCodi=TS.Nivell();
   if (nivell==0) {
      cod.Put(IPushBVar,4*(NivellCodi-1),8);
      cod.Put(IPushB);
   }
   else cod.Put(IPushBVar, 4*(NivellCodi-nivell), 8+(nivell-1)*4);
};
@cod.Put(ICall,RefEtiqueta(ets.Posicio));
@cod.Put(IAddSP,ets.DespRetorn-8);
```

Generació del Codi d’una Funció

- La mida del bloc d'activació no es coneix fins que no s’ha compilat la funció (variables temporals del for, etc).
  - Usar etiquetes que tenen per valor una mida i no una adreça.
  - Generà el codi d’entrada a la funció després de generar el codi del cos de la funció
- Una variable global acumula les mides de les variables. Per poder compilar funcions aniuades es guarda als separadors de la taula de símbols
- El codi de las funcions aniuades es posa fora del codi de la funció on es defineixen (CodiGlobal)

Generació de codi per una funció en LS (I)

```
Rule <DecFun>::=
  @Var tret,params,nom,ef;
  @Var cod=queue();
  @Var desp= 8+ts.Nivell*4;
  Function identificador#(nom)
  @{
     TS.ComprovarDuplicat(nom);
     var inici=etiqueta(ts.Cami(nom));
     ef=ETSFuncio(nom,inici);
     TS.Insertar(ef);
     TS.NouAmbit(ef);
  }
  <DecParams(params, desp)> : <tipus(tret,Unbound)>
  @ef.Parametres=params;
  @ef.TipusRetorn=tret;
  @ef.DespRetorn=desp;
  <bloc(cod)>
```

Generació de codi per una funció en LS (II)

```
// Inici de la funció
@CodiGlobal.put(DefEtiqueta(inici));
@CodiGlobal.Put(ILink,TS.VarSz);
// Cos de la funció
@CodiGlobal.PutElements(cod);
// Final de la funció
@CodiGlobal.put(IUnlink);
@CodiGlobal.put(IRet);
@TS.EliminarAmbit();

Rule <bloc(cod)>::= "BEGIN"
   {
      <DecFun> | <DecProc> | <DecVar> | <DecTipus>
      | <instruccio(cod)>
   }
   "END"
```

Generació de codi per una funció en LS (III)

```
Rule <DecParams(&TipusParametres,&desp)>::=
        @var LT=queue(),t;
        "(" [
          <DecParametre(t,desp)>  @LT.Put(t);
          { , <DecParametre(t,desp)> @LT.Put(t); }
          ]")"
        @TipusParametres=LT.GetList();

Rule <DecParametre(&t,&desp)>::=
        @Var nom;
        identificador#(nom) : <tipus(t,Unbound)>
        @TS.ComprovarDuplicat(nom);
        @TS.Insertar(ETSVariable(nom,t,desp));
        @desp=desp+MidaTipus(t);
```

Generació de Codi per return

```
Rule <instruccio(cod)>::= ... |
  return ( <Expressio(cod,t,true)>
    @if (ts.Funcio.TipusRetorn!:=t)
          Throw Exception("Error de tipus a return");
    @cod.put(IPopBVar,MidaTipus(t),ts.Funcio.DespRetorn);
   | $
    @if (ts.Funcio.TipusRetorn!:=TVoid) Throw Exception(
                "Falta el valor de retorn a return");
  ) ";"
  @cod.put(IUnlink);
  @cod.put(IRet);
  | ...
```

Generació de codi de les Estructures de Control

IF <expressió> THEN <instrucció$_1$>ELSE <instrucció$_2$>

```
        Codi[<expressió>]
        JmpFalse Else
        Codi[<instrucció1>]
        Jmp FinIf
Else:
        Codi[<instrucció2>]
FinIf:
```

- Problema
  - S’han de generar salts a adreces de codi desconegudes.
- Solució
  - Utilitzar etiquetes simbòliques si el codi generat és assembler.
  - Si el codi generat és codi màquina s’han d’utilitzar unes estructures de dades que facin la funció d’etiqueta.

Generació de Codi de la Instrucció IF

```
IF <expressió>
    @e1=CrearEtiqueta();
    @Codi[JmpFalse e1]        <—— Referencia
    THEN <instrucció> (
        λ @Codi[e1:] |        <—— Instanciar

        @e2=CrearEtiqueta();
        @Codi[Jmp e2]         <—— Referencia
        @Codi[e1:]            <—— Instanciar
        ELSE <instrucció>
        @Codi[e2:]
    )
```

Sense ELSE

```
    Codi[<expressió>]
    JmpFalse e1
    <instrucció>
e1:
```

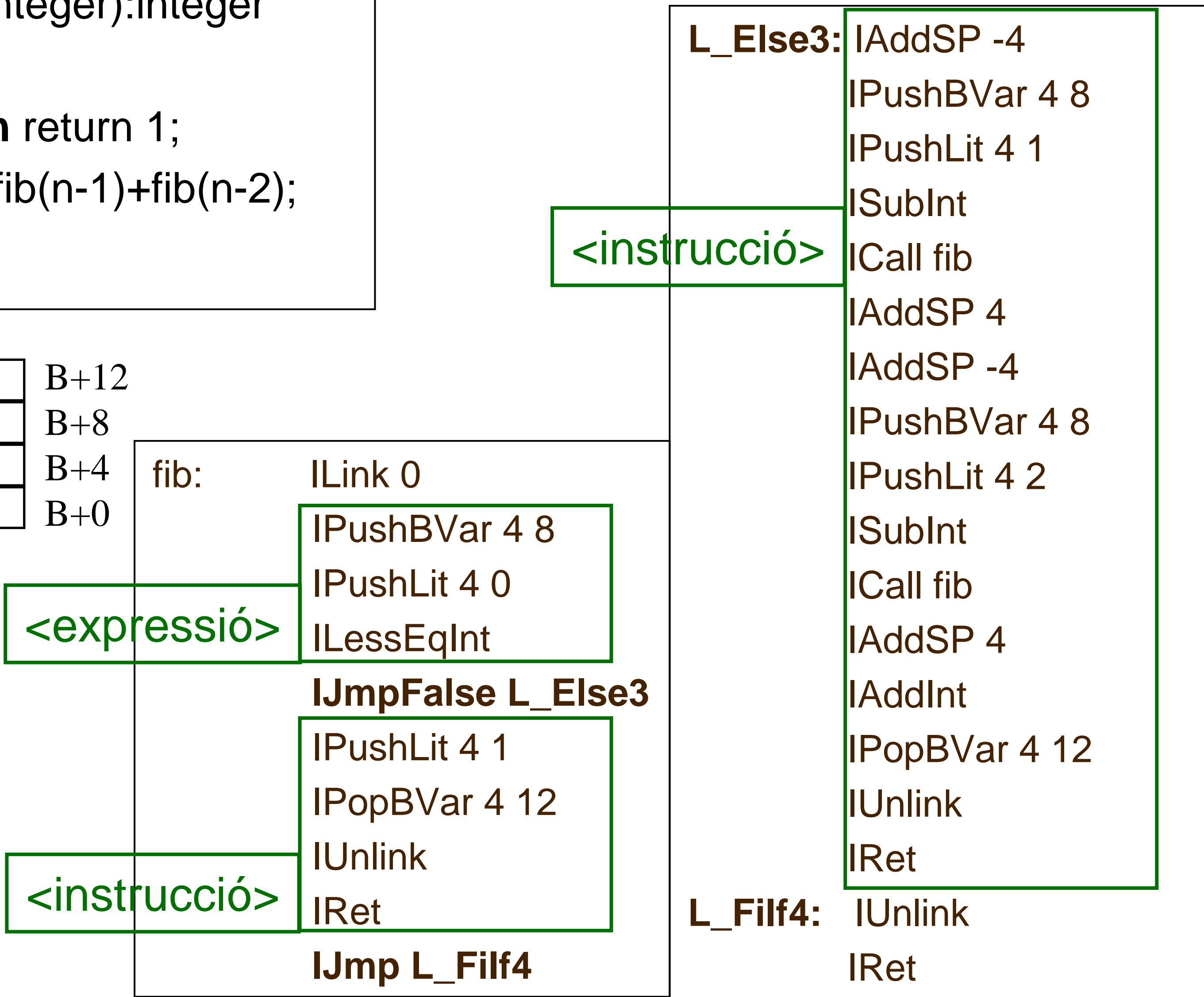
Amb ELSE

```
    Codi[<expressió>]
    JmpFalse e1
    <instrucció>
    Jmp e2
e1:
    <instrucció>
e2:
```

Generació de Codi de la Instrucció IF

```
Rule <instruccio(cod)>::= ...|
  if <Expressio(cod,t,true)>
  @if (t!=TInt) throw Exception("Error de tipus");
  @Var EtiElse=Etiqueta("L_Else%");
  @cod.put(IJmpFalse,RefEtiqueta(EtiElse));
  then <instruccio(cod)> (
    else
    @Var EtiEndIf=Etiqueta("L_FiIf%");
    @cod.put(IJmp,RefEtiqueta(EtiEndIf));
    @cod.Put(DefEtiqueta(EtiElse));
    <instruccio(cod)>
    @cod.Put(DefEtiqueta(EtiEndIf));
    | $
    @cod.Put(DefEtiqueta(EtiElse));
    ) | ...
```

Exemple de codi generat per l'if



Generació de Codi de la Instrucció WHILE

WHILE <expressió> DO <instrucció>

```
WHILE
  @repetir=CrearEtiqueta();
  @fi=CrearEtiqueta();
  @Codi[repetir:]          <---------- Instanciar
<expressió>
   @Codi[JmpFalse fi]      <---------- Referencia
DO <instrucció>
  @Codi[Jmp repetir]       <---------- Referencia
  @Codi[fi:]               <---------- Instanciar
```

```
repetir:
        Codi[<expressió>]
        JmpFalse fi
        <instrucció>
        Jmp repetir
fi:
```

Generació de Codi de la Instrucció WHILE

```
Rule <instruccio(cod)>::= ...|
    while
    @var EtiWhile=Etiqueta("L_RepetirWhile%");
    @var EtiEndWhile=Etiqueta("L_FiWhile%");
    @Cod.Put(DefEtiqueta(EtiWhile));
    <Expressio(cod,t,true)>
    @if (t!=TInt) throw Exception("Error de tipus");
    @cod.put(IJmpFalse,RefEtiqueta(EtiEndWhile));
    do <instruccio(cod)>
    @cod.put(IJmp,RefEtiqueta(EtiWhile));
    @Cod.Put(DefEtiqueta(EtiEndWhile));
     | ...
```

Exemple de codi generat per while

```
function Fac(n:integer):integer
begin
    var i:integer;
    var r:integer;
    i:=1; r:=1;
    while i<=n do
    begin
        r:=r*i; i:=i+1;
    end
    Fac:=r;
end
```

valor de ret	B+12
n	B+8
PC	B+4
ED	B+0
i	B-4
r	B-8

```
fac:    ILink 8
        IPushLit 4 1
        IPopBVar 4 -4
        IPushLit 4 1
        IPopBVar 4 -8
L_Rep5: IPushBVar 4 -4      <expressió>
        IPushBVar 4 8
        ILessEqInt
        IJmpFalse L_FiW6
        IPushBVar 4 -8      <instrucció>
        IPushBVar 4 -4
        IMultInt
        IPopBVar 4 -8
        IPushBVar 4 -4
        IPushLit 4 1
        IAddInt
        IPopBVar 4 -4
        IJmp L_Rep5
L_FiW6: IPushBVar 4 -8
        IPopBVar 4 12
        IUnlink
        IRet
        IUnlink
        IRet
```

Generació de Codi de la Instrucció FOR de C

```
for ( <inicialització>; <condició>; <increment>)
    <instrucció>

    Codi[<inicialització>]
repetir:
    Codi[<condició>]
    JmpFalse fi
    Codi[<instrucció>]
    Codi[<increment>]
    Jmp repetir
fi:
```

- Problema
  - El codi es genera en un ordre diferent al de l'anàlisi sintàctica.
- Solució
  - Generar llistes d’instruccions que després es puguin bolcar al codi final quan sigui necessari.

Generació de Codi de la Instrucció FOR de PASCAL

```
FOR <var>:=<ini> TO <lim> DO <instrucció>
    Codi[<ini>]
    PopBVar4 desp(<var>)
    Codi[<lim>]
    PopBVar4 desp(var temporal)
repetir:
    PushBVar4 desp(<var>)
    PushBVar4 desp(var temporal)
    Greater4
    JmpTrue fi
    Codi[<instrucció>]
    PushBVar4 desp(<var>)
    PushLit4 1
    AddInt
    PopBVar4 desp(<var>)
    Jmp repetir
fi:
```

- Problema: S'ha de crear una variable temporal pel límit del bucle.

For en LS (I)

```
Rule <instruccio(cod)>::= ...|
  for identificador#(nom) = <Expressio(cod,t,true)>
  @if (t!:=TInt) throw Exception("Error de tipus");
  @TS.NouAmbit();
  @ts.VarSz=ts.VarSz+MidaTipus(TInt);
  @TS.Insertar(ETSVariable(nom,TInt,-ts.VarSz));
  @var DespIndex=-ts.VarSz;
  @cod.Put(IPopBVar,4,DespIndex);
  to <Expressio(cod,t,true)>
  @if (t!:=TInt) throw Exception("Error de tipus");
  @ts.VarSz=ts.VarSz+MidaTipus(TInt);
  @TS.Insertar(ETSVariable(".Limit",TInt,-ts.VarSz));
  @var DespLimit=-ts.VarSz;
  @cod.Put(IPopBVar,4,DespLimit);
  @var repetirfor=etiqueta("L_RepetirFor%");
  @var fifor=etiqueta("L_FiFor%");
```

For en LS (II)

```
@cod.put(DefEtiqueta(repetirFor));
@cod.put(IPushBVar,4,DespIndex);
@cod.put(IPushBVar,4,DespLimit);
@cod.put(ILessEqInt);
@cod.put(IJmpFalse,RefEtiqueta(fiFor));
do <instruccio(cod)>
@cod.put(IPushBVar,4,DespIndex);
@cod.put(IPushLit,4,1);
@cod.put(IAddInt);
@cod.put(IPopBVar,4,DespIndex);
@Cod.Put(IJmp,RefEtiqueta(repetirFor));
@Cod.Put(DefEtiqueta(FiFor));
@TS.EliminarAmbit(); | ...
```

Exemple de codi generat per for

```
Function Fac2(n:Integer):Integer
begin
   Var r:Integer;
   r=1;
   for i=2 to n do r=r*i;
   return r;
end
```

valor de ret	B+12
n	B+8
PC	B+4
ED	B+0
r	B-4
i	B-8
límit for	B-12

```
fac2:     ILink 12
          IPushLit 4 1
          IPopBVar 4 -4
          IPushLit 4 2          <expressió> índex
          IPopBVar 4 -8
          IPushBVar 4 8         <expressió> límit
          IPopBVar 4 -12
L_RFor:   IPushBVar 4 -8
          IPushBVar 4 -12
          ILessEqInt
          IJmpFalse L_FiFor
          IPushBVar 4 -4        <instrucció>
          IPushBVar 4 -8
          IMultInt
          IPopBVar 4 -4
          IPushBVar 4 -8
          IPushLit 4 1
          IAddInt
          IPopBVar 4 -8
          IJmp L_RFor
L_FiFor:  IPushBVar 4 -4
          IPopBVar 4 12
          IUnlink
          IRet
          IUnlink
          IRet
```

Instruccions de Control de Flux Dependents del Context

- Break i Continue
  - Break i continue són instruccions que salten al principi o fi d'un bucle i es compilen separadament del bucle. Exemple:

    ```
    for (i=1;i<100;++i)
      if (a[i]==v) break;
    ```

    el break del bucle està dins d'un if i per això el seu codi no el genera la funció que compila el for.
  - Compilació:
    - Quan s'entra a compilar el bucle es posen a la TS una entrada amb les etiquetes d'inici i final del bucle.
    - Break i Continue obtenen l'adreça de salt de la primera entrada de classe bucle que troben a la taula de símbols.
    - Al sortir del bucle s'elimina l'entrada de la taula de símbols.

Instruccions de Control de Flux Dependents del Context

- Goto no local
  - Un Goto pot sortir d'una funció. En aquest cas s'ha d'eliminar el bloc d'activació de la funció.
  - A la cerca de l'etiqueta a la que salta es poden detectar els àmbits dels que s'ha de sortir i com s'ha de modificar la pila.
- Try Catch Throw
  - No es pot saber l'adreça de salt d'un Throw fins que s'executa el programa. S'utilitzen blocs d'activació d'excepcions que contenen l'adreça de salt i la informació necessària per recuperar l'estat de la pila i els registres del processador.

Generació de Codi d'Expressions amb Variables Temporals

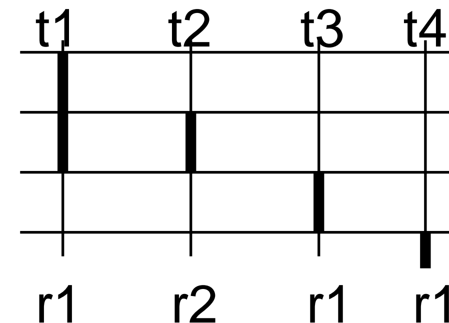
- La generació de codi basada en pila té el problema que no aprofita els registres del processador.
- Solució:
  - Utilitzar variables temporals pels resultats intermedis.
  - Definir instruccions de dos o tres operands. Ex
    - Add t1,t2 t1=t1+t2
    - Add t1,t2,t3 t1=t2+t3
  - Posar les variables temporals a registres sempre que sigui possible.
- Exemple: a*b+c*d-e*f

```
mul t1,a,b
mul t2,c,d
add t1,t1,t2
mul t2,e,f        (es reutilitza t2)
sub t1,t1,t2
```

Assignació de Registres

- Les variables temporals es guarden als registres del processador sempre que sigui possible. En cas contrari es guarden a la pila.
- Les variables temporals es creen i destrueixen per delimitar el seu temps de vida i poder reutilitzar els registres de la CPU.
- Exemple: (a+b)*(c+d)-e

```
add t1,a,b
add t2,c,d
mul t3,t1,t2
sub t4, t3,e
```



- Codi generat després de l'assignació

```
add r1,a,b
add r2,c,d
mul r1,r1,r2
sub r1, r1,e
```

Assignació Simple de Registres (I)

- Idea. Controlar els registres que s'estan utilitzan en cada moment

```
type [print=contents] reg(n);

var Ocupats=[]; // Registres en us

fun NouReg()=>  // Obtenir un registre lliure
   Found (i<-1..,!Ocupats.memberp(i)) {
      Ocupats=i::Ocupats;
      reg(i);
   }

proc AlliberarReg(r)=> // Alliberar un registre
   if (typeP(reg,r)) Ocupats=[x|x<- Ocupats, x!=r.n];
```

Assignació Simple de Registres (II)

- Cada regla BNF de expressió retorna el lloc on esta el seu resultat. Aquest pot ser:
  - Un registre: Reg(i)
  - Una addreça de variable global: &a
  - Un literal: 10
- S'aprofiten els registres dels operands pel resultat.

Factor

```
Rule <factor(&r)>::=  @var v,nom; (
      - <factor(r)>
        @{
          var out=if (typeP(reg,r)) r  else NouReg();
          cout.PrintLn("NEG ",out,", ",r);
          // if (out!=r) AlliberarReg(r);
          r=out;
        }
     | "(" <Expressio(r)> ")"
     | Numero#(v) @r=v;
     | Identificador#(nom)  @r=strprint("&",nom);
     )
```

Terme

```
Rule <terme(&r)>::=  @Var r2,op;
     <factor(r)> {
        ( * @op="MUL";   | / @op="DIV"; )  <factor(r2)>
        @{
           var out=if (typeP(reg,r)) r
              else if (typeP(reg,r2)) r2
              else NouReg();
           cout.PrintLn(op," ",out,", ",r,", ",r2);
           if (out!=r) AlliberarReg(r);
           if (out!=r2) AlliberarReg(r2);
           r=out;
        }
     }
```

Resultat

- Codi generat per: 10+a*b+30*(c-b*j)+30*(c-b*j)

```
MUL reg(1), &a, &b              // reg(1)=a*b
ADD reg(1), 10, reg(1)          // reg(1)=10+a*b
MUL reg(2), &b, &j              // reg(2)=b*j
SUB reg(2), &c, reg(2)          // reg(2)=c-b*j
MUL reg(2), 30, reg(2)          // reg(2)=30*(c-b*j)
ADD reg(1), reg(1), reg(2)      // reg(1)= 10+a*b+ 30*(c-b*j)
MUL reg(2), &b, &j              // reg(2)=b*j
SUB reg(2), &c, reg(2)          // reg(2)=c-b*j
MUL reg(2), 30, reg(2)          // reg(2)=30*(c-b*j)
ADD reg(1), reg(1), reg(2)      // reg(1)= 10+a*b+ 30*(c-b*j)+
                                           30*(c-b*j)
```

Exemples de generació de codi

- Us del display
- Accés a arrays

Crides i display

```
PROCEDURE P1()
BEGIN
    var v1:integer;
    PROCEDURE P2()
    BEGIN
        var v2:integer;
        PROCEDURE P3()
        BEGIN
            var v3:integer;
            P1();
        END
        PROCEDURE P4()
        BEGIN
            var v4:integer;
            PROCEDURE P5()
            BEGIN
                var v5:integer;
                P2();
            END
            PROCEDURE P6()
            BEGIN
                var v6:integer;
                P5();
            END
            P6();
            P3();
        END
        P4();
    END
    PROCEDURE P7()
    BEGIN
        var v7:integer;
        P2();
    END
    P7();
END
```

Crides i display: codi generat (I)

```
p1:     ILink 4
        IPushB
        ICall p7
        IAddSP 4
        IUnlink
        IRet
p2:     ILink 4
        IPushBVar 4 8
        IPushB
        ICall p1.p4
        IAddSP 8
        IUnlink
        IRet
```

Crides i display: codi generat (II)

```
p1.p3:  ILink 4
        ICall p1
        IAddSP 0
        IUnlink
        IRet
p1.p4:  ILink 4
        IPushBVar 8 8
        IPushB
        ICall p1.p2.p6
        IAddSP 12
        IPushBVar 8 8
        ICall p1.p3
        IAddSP 8
        IUnlink
        IRet
```

Crides i display: codi generat (III)

```
p1.p2.p5:ILink 4
         IPushBVar 4 16
         ICall p2
         IAddSP 4
         IUnlink
         IRet
p1.p2.p6:ILink 4
         IPushBVar 12 8
         ICall p1.p2.p5
         IAddSP 12
         IUnlink
         IRet
p7:      ILink 4
         IPushBVar 4 8
         ICall p2
         IAddSP 4
         IUnlink
         IRet
```

Taula de símbols al procediment P5 (I)

```
FUNCIO:
 Nom          => p1
 Parametres   =>
 Posició      => p1
 Tipus retorn => Void
 Desp Retorn  => 8
SEPARADOR DE FUNCIO-----------
 Funció => unbound
 VarSz  => Unbound
==============================
VARIABLE:
 Nom     => v1
 Posició => -4
 Tipus   => Integer
FUNCIO:
 Nom          => p2
 Parametres   =>
 Posició      => p2
 Tipus retorn => Void
 Desp Retorn  => 12
SEPARADOR DE FUNCIO-----------
 Funció => p1
 VarSz  => 4
==============================
VARIABLE:
 Nom     => v2
 Posició => -4
 Tipus   => Integer
FUNCIO:
 Nom          => p3
 Parametres   =>
 Posició      => p1.p3
 Tipus retorn => Void
 Desp Retorn  => 16
FUNCIO:
 Nom          => p4
 Parametres   =>
 Posició      => p1.p4
 Tipus retorn => Void
 Desp Retorn  => 16
```

Taula de símbols al procediment P5 (II)

```
SEPARADOR DE FUNCIO-----------
  Funció => p2
  VarSz  => 4
============================
VARIABLE:
  Nom     => v4
  Posició => -4
  Tipus   => Integer
FUNCIO:
  Nom          => p5
  Parametres   =>
  Posició      => p1.p2.p5
  Tipus retorn => Void
  Desp Retorn  => 20
```

```
SEPARADOR DE FUNCIO-----------
  Funció => p4
  VarSz  => 4
============================
VARIABLE:
  Nom     => v5
  Posició => -4
  Tipus   => Integer
```

Display i variables

```
PROCEDURE P5()
BEGIN
    var v5:integer;
    P2();
    print v1;
    print v2;
    print v4;
    print v5;
END
```

```
p1.p2.p5:ILink 4
        IPushBVar 8 8
        ICall p2
        IAddSP 4
        IPushDispVar 4 16 -4
        IPrintInt
        IPushDispVar 4 12 -4
        IPrintInt
        IPushDispVar 4 8 -4
        IPrintInt
        IPushBVar 4 -4
        IPrintInt
        IUnlink
        IRet
```

Accés a arrays

```
PROCEDURE P1()
BEGIN
   Type tabla=array [10] of array [5] of integer;
   Var a:tabla;
   Var b:tabla;
   a[2][3]=20;
   print a[2][3];
   b[4]=a[6];
END
```

Accés a arrays (a[2][3]=20;)

p1: ILink 400

```
ILink 400
IPushLit 4 20
IPushAddressBVar -200
IPushLit 4 2
IPushLit 4 20
IMultInt
IAddInt
IPushLit 4 3
IPushLit 4 4
IMultInt
IAddInt
IPopInd 4
```

Accés a arrays (print a[2][3];)

```
IPushAddressBVar -200
IPushLit 4 2
IPushLit 4 20
IMultInt
IAddInt
IPushLit 4 3
IPushLit 4 4
IMultInt
IAddInt
IPushInd 4
IPrintInt
```

a[2][3]

lectura

impressió

Accés a arrays (b[4]=a[6];)

```
IPushAddressBVar -200
IPushLit 4 6
IPushLit 4 20
IMultInt
IAddInt
IPushInd 20
IPushAddressBVar -400
IPushLit 4 4
IPushLit 4 20
IMultInt
IAddInt
IPopInd 20
IUnlink
IRet
```

a[6] lectura

b[4] adreça

assignació

Operacions booleanes amb dreceres (I)

```
Rule <TerBool(cod,&t,Resultat)>::=
        @var t2;
        @var EtiFalse=Etiqueta("L_AndFalse%");
        @var Ands=false;
        <FacBool(cod,t,Resultat)> {
              @if (!Resultat) throw Exception("No s'aprofita el resultat");
              @cod.Put(IJmpFalse,RefEtiqueta(EtiFalse));
              && <FacBool(cod,t2,Resultat)>
              @if (t!:=TInt || t2!:=TInt) throw Exception(
                    "Error de tipus de dades en && ",t,t2);
              @Ands=true;
          }
```

Operacions booleanes amb dreceres (II)

```
@if (Ands) {
    var EndAnds=Etiqueta("L_EndAnds%");
    Cod.Put(IJmp,RefEtiqueta(EndAnds));
    Cod.Put(DefEtiqueta(EtiFalse));
    Cod.Put(IPushLit,4,0);
    Cod.Put(DefEtiqueta(EndAnds));
 };
```

Exemple de generació de codi

Procedure P(a:integer,b:integer)
BEGIN
    if a>0 && a>b && b>0 then Print 1;
    else Print 0;
END

```
p:        ILink 0
          IPushBVar 4 8
          IPushLit 4 0
          IGreaterInt
          IJmpFalse L_AndF
          IPushBVar 4 8
          IPushBVar 4 12
          IGreaterInt
          IJmpFalse L_AndF
          IPushBVar 4 12
          IPushLit 4 0
          IGreaterInt
          IJmp L_End
L_AndF:   IPushLit 4 0
L_End:    IJmpFalse L_Else3
          IPushLit 4 1
          IPrintInt
          IJmp L_FiIf5
L_Else3:  IPushLit 4 0
          IPrintInt
L_FiIf5:  IUnlink
          IRet
```