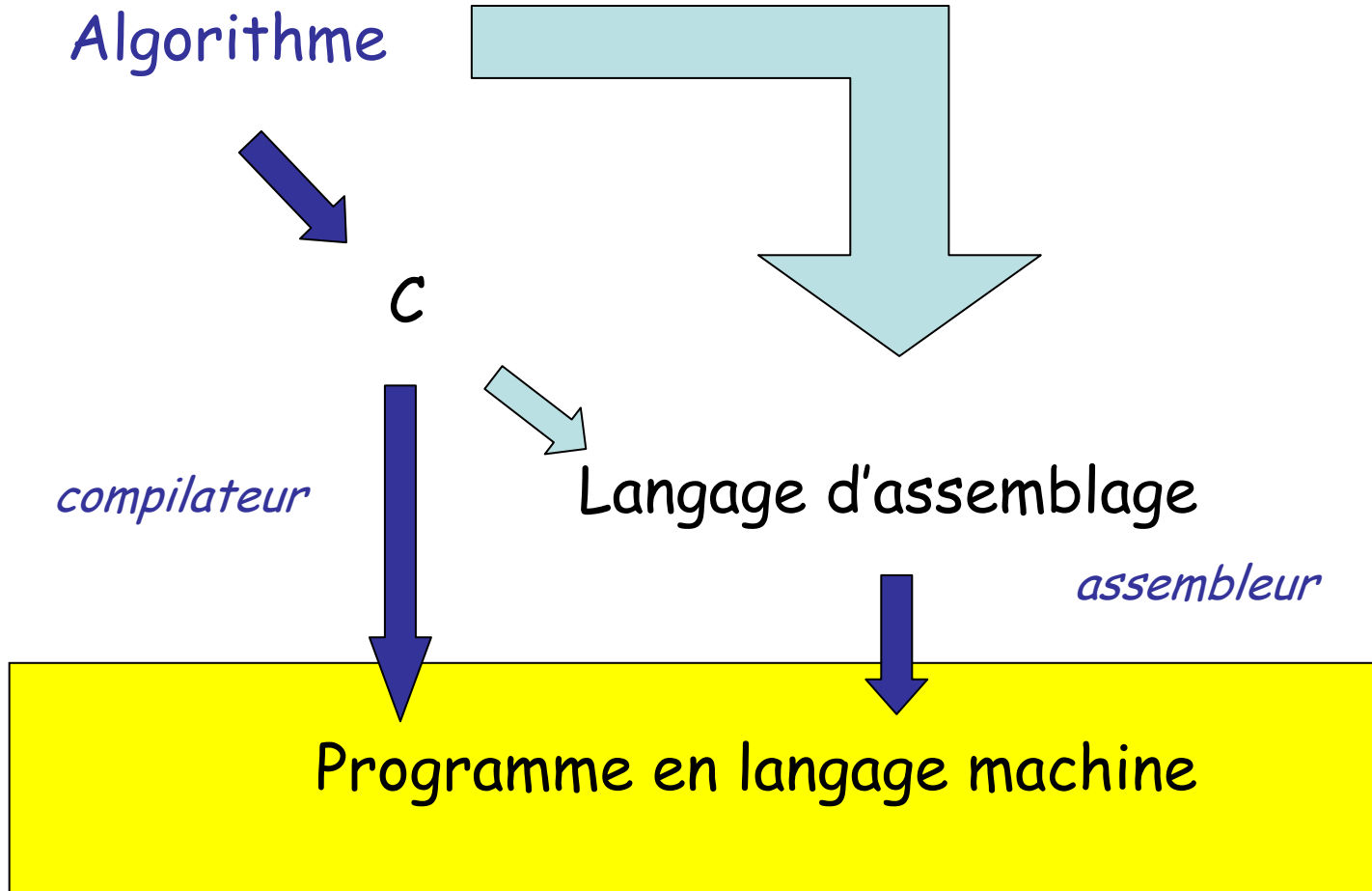


Langages d'assemblage

Introduction



Introduction (2)

- Langage machine
 - Jeu d'instructions
 - Modes d'adressage
 - Codage des instructions
 - Suite de tableaux de bits
- Langage d'assemblage
 - Notation textuelle
 - Introduction de mécanismes de nommage
- Langage haut niveau
 - Structures + ...

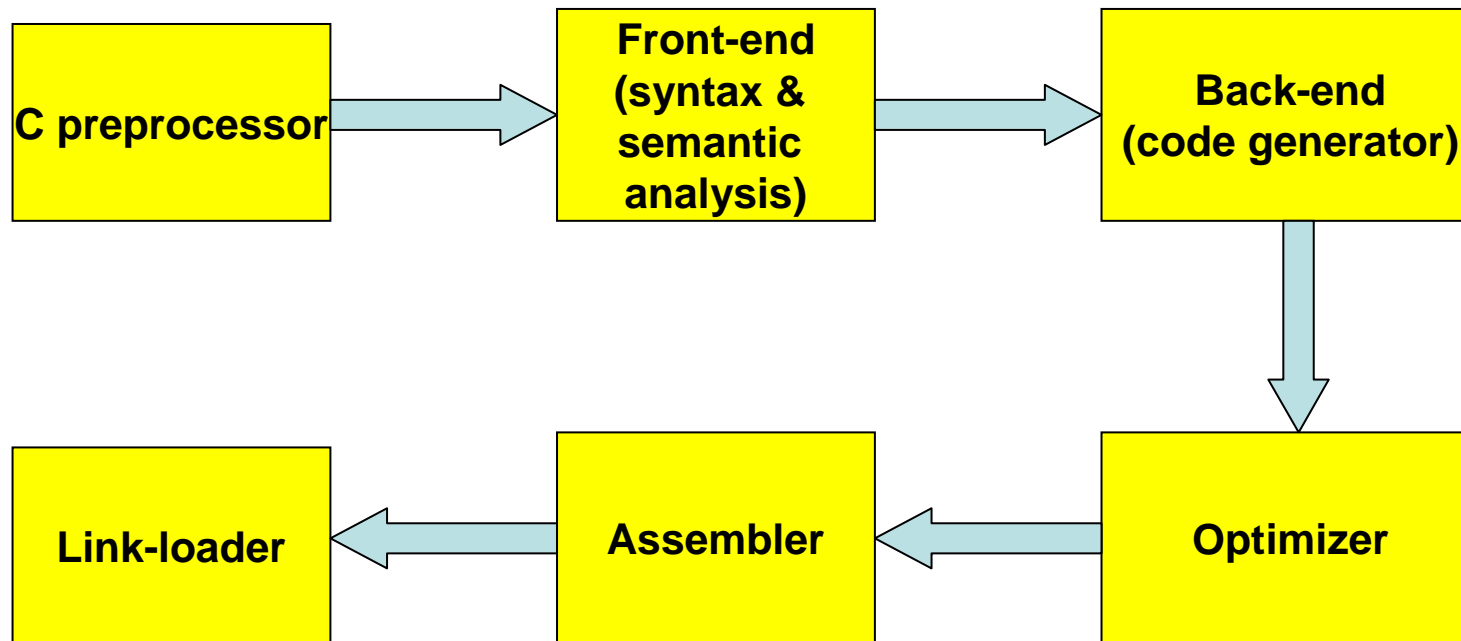
Introduction (3)

- Nécessité pour un langage d'assurer un modèle de calcul universel
- Définir une **syntaxe**
- Donner une **sémantique**
 - Pour les langages machine et les langages d'assemblage on utilise une **sémantique d'actions**
 - Notations : MEM[], REG[], ←, ... + C

Notation pour la sémantique

Notation	Signification	Exemple	Signification
\leftarrow	Transfert de données. La longueur est donnée par la longueur de la destination.	Reg [R1] \leftarrow Reg [R2]	Transfère le contenu du registre R2 dans le registre R1
\leftarrow_n	Transfère n bits. Permet de lever l'ambiguïté sur la longueur.	Mem [y] \leftarrow_{16} Mem [x]	Transfère 16 bits à partir de la case mémoire x vers la case y.
X_n	Sélection du bit d'indice n	Reg [R1] ₀ \leftarrow 0	Mise à 0 du bit 0 du registre R1
$X_{m..n}$	Sélection d'un champ de bits.	Reg [R3] _{7..0} \leftarrow Mem [x]	Transfère le contenu de la case mémoire x dans l'octet bas de R3
X^n	Duplication d'un champ, n fois.	Reg [R3] _{31..8} \leftarrow 0^{24}	Met à 0 les trois octets de poids fort de R3
##	Concatène deux champs.	Reg [R3] \leftarrow $0^{24}##$ Mem [x]	Registre R3 reçoit le contenu de la case mémoire x dans l'octet bas et 0 dans les 3 autres octets

Compilation C



Langage d'assemblage

Proche du langage machine, mais humainement plus lisible et manipulable

Ex: mov ax,2 → 0xB80200
 add ax,3 → 0x050300

Apports :

1. Notation textuelle

- Mnémoniques pour les opérations
- Identificateurs et expressions pour les opérandes

2. Directives

- Réservations, initialisations mémoires
- Imposer certains choix

Langage d'assemblage (2)

3. Commentaires

4. Séparation plus claire des données et instructions

- TEXT zone des instructions
- DATA, BSS zones des données
- Mécanisme de **nommage des positions** dans les zones permettant de faire abstraction des valeurs explicites d'adresses mémoire.

Remarque

- A un processeur on peut associer plusieurs langages d'assemblage
- Notion de famille de processeurs (style Intel, Motorola, ...)
- Nous verrons des exemples de
 - Intel (x86 et Pentium)
 - Motorola (68k, 68HC11)
 - RICS (SPIM, Spark)

Exemples

- Le plus simple:
 - Calculer $C = A - B$ pour A et B entiers
- Un peu plus élaboré:
 - Calculer le pgcd de deux entiers positifs A et B

$$C = A - B$$

load A

sub B

store C



Ex: A=12; B=18

C = A - B (pentium)

```
.data
a    dd    12
b    dd    18
c    dd    ?

.code
mov  eax, [a]           ; eax ← A
sub  eax, dword ptr [b] ; eax ← eax - B
mov  [c], eax          ; C ← eax
```



Algorithme

- Calcul pgcd:
- Données: A, B entiers positifs
- Résultat: $\text{pgcd}(A, B)$

• **Tant que** $A \neq B$ **faire**

Si $A > B$ **alors**

$A \leftarrow A - B$

Sinon

$B \leftarrow B - A$

Fsi

Fait

$\text{Pgcd} \leftarrow A$

Pre-condition:
 $A > 0$ et $B > 0$

Invariant de boucle:
 $A > 0$ et $B > 0$

Post-condition:
 $A = B$

Pgcd (C)

```
// gcd(a,b)
// Charles André
// February 12, 2002
int a=12, b=18, gcd;
int main () {
    while ( a != b ) {
        if ( a > b )
            a = a - b;
        else
            b = b - a;
    }
    gcd = a;
}
```

Pgcd sur machine à accumulateur

```
    load A
    store C      ; C ← A
    load B
    store D      ; D ← B
bcl: sub C        ; accu = B - A
     beqz done
     bltz then   ; branch if B < A
     store D     ; B ← B - A
     bt bcl
then: neg         ; accu = A - B
     store C     ; A ← A - B
     load D      ; accu = B
     bt bcl
done: load C      ; accu ← A
     store gcd   ; result
```




Pgcd (Pentium)

```
bcl:    mov     eax,[a]
        cmp     eax,dword ptr [b]
        je      done           ; go to done if a == b
        mov     ecx,dword ptr [a]
        cmp     ecx,dword ptr [b]
        jle     else         ; go to else if a <= b
        mov     edx,dword ptr [a]
        sub     edx,dword ptr [b]
        mov     dword ptr [a],edx ; a <- a - b
        jmp     bcl          ; iterate
else:   mov     eax,dword ptr [b]
        sub     eax,dword ptr [a]
        mov     [b],eax      ; b <- b - a
        jmp     bcl          ; iterate
done:   mov     ecx,dword ptr [a] ; result in a
        mov     dword ptr [gcd], ecx ; store it in gcd
```



Pgcd (SPIIM)

```
    .data
va:   .word 12
vb:   .word 18
gcd:  .word 0
      .text
      .globl main
main:
      lw     $t0,va           # t0 <- va
      lw     $t1,vb           # t1 <- vb
bcl:  beq    $t0,$t1,done
      blt    $t0,$t1,else     # go to else if va<vb
      sub    $t0,$t0,$t1      # va <- va - vb
      j     bcl
else: sub    $t1,$t1,$t0      # vb <- vb - va
      j     bcl
done: sw     $t0,gcd          # gcd <- result
      li    $v0,10           # magic: exit
      syscall
```



Aspects lexicographiques

Commentaires

- Depuis un symbole jusqu'à la fin de la ligne
 - Pentium: ; ...
 - 68HC11: *... ou après instruction complète
 - SPIM: ; ...
 - C //...

Notation de constantes en plusieurs bases

Décimale
Octale		...O (Intel)	0...
Héxadécimale	\$... (Motorola)	...h (Intel)	0x...
Binaire	%... (Motorola)	...b (Intel)	

Aspects lexicographiques (2)

Macro assembleur

Substitutions textuelles

- Constantes textuelles
 - Motorola foo EQU value
- Macros avec paramètres
 - Dans les assembleurs plus évolués

Structuration des programmes

Distinction de zones ou segments

Zone d'instructions : TEXT

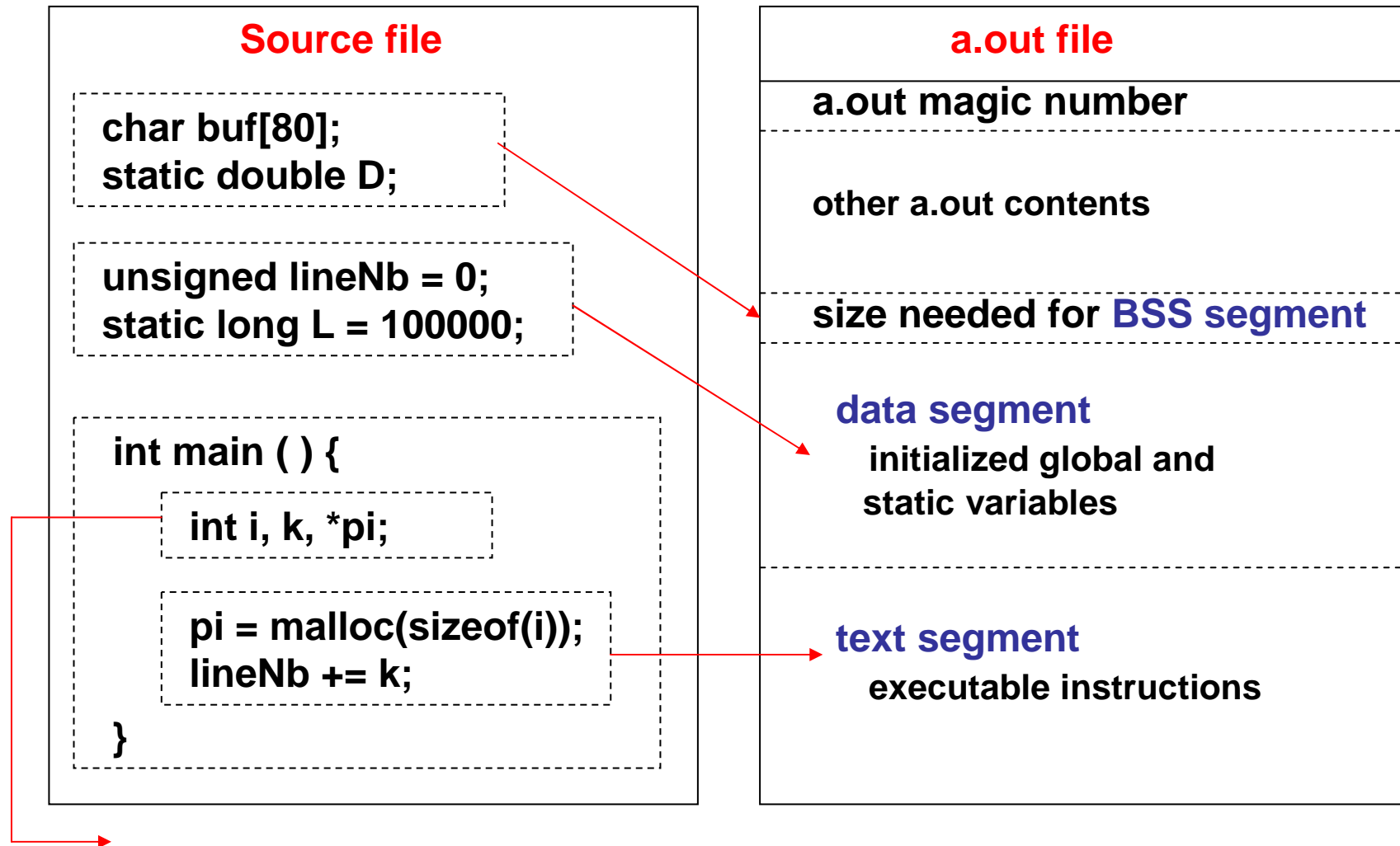
Zone de données :

- Données initialisées : DATA
- Données non initialisées : BSS

BSS = Block Started by Symbol (original acronym)

I prefer "Better Save Space"

Mapping C statements to Segments



Local variables don't go in a.out,
but are created at run-time

Nommage des adresses

Pour repérer des instructions dans la zone TEXT ou des données dans les zones DATA ou BSS

Etiquettes (ou Labels)

Problème de l'assembleur de calculer les adresses (relatives ou absolues).

Notion de programme relogeable

instructions

[étiquette] mnémonique [operande1 [,operande2]] [commentaires]

Mnémonique

Acronyme: ASL (Arithmetic Shift Left)

Abréviation: ADD (Addition)

Opérandes

Registres: nom

Constantes: valeurs immédiates

Désignation d'adresses en mémoire

Grande variété de notations suivant les assembleurs,

Danger: ordre des opérandes

Src, Dest pour Motorola

Dest, Src pour Intel, SPIM

Pseudo-instructions

- Ce sont des instructions supportées par l'assembleur, mais pas directement implantées par une instruction en langage machine.
- Exemple en SPIM:

```
blt    a,b,Less           ; go to Less if a < b
```

Est transformé en

```
slt    $at,a,b           ; REG[at] ← (a < b)
```

```
bne    $at,$zero,Less    ; go to Less if at≠0
```


Zones et directives de réservation de mémoire

Intel : `.data`
étiquette directive valeur | ?

directive

DBIT	variables de type bit
DB	variables de type byte (8)
DW	variables de type word (16)
DD	variables double word (32)
DP	variables de type pword (48)
DQ	variables quad word (64)
DT	variables de ten byte (80)

Zones et directives de réservation de mémoire (2)

68HC11 :

Non initialisée : *Reserve Memory Byte*
étiquette **RMB** entier

Initialisée : *Form Double Byte*
étiquette **FDB** valeur

Form Constant Byte
étiquette **FCB** valeur

Form Constant Char
étiquette **FCC** 'chaine'

Assembleur

C'est le programme chargé de traduire le langage assembleur en langage machine

3 fonctions principales

1. Analyse lexicale et syntaxique

Détecte des erreurs

Extension de macros si nécessaire

2. Codage en binaire

3. Traduction des étiquettes en adresses absolues ou en déplacements

Table des symboles