Microprocesseurs (MIC)

Chapitre 5 : Assembleurs

L'Assembleur

- Le terme « assembleur » désigne :
  - Un langage de programmation
  - Le compilateur de ce langage

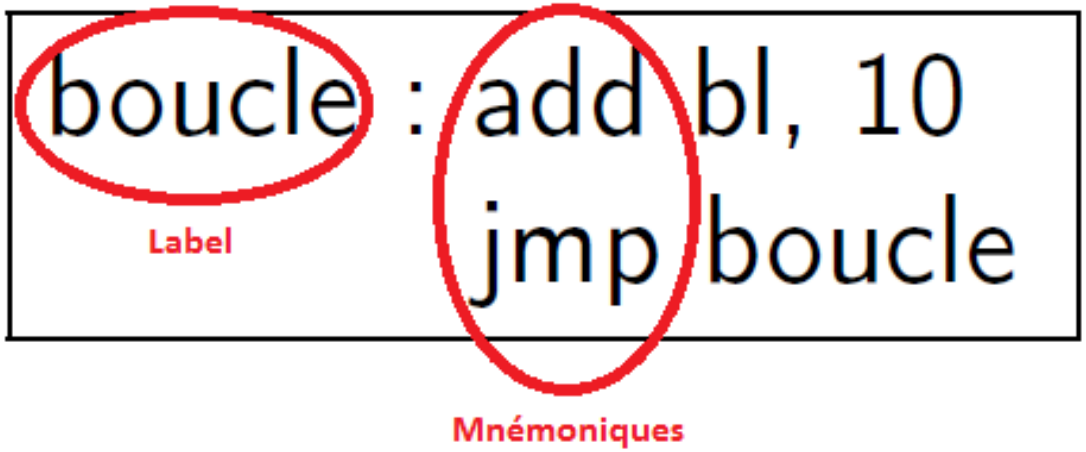
L'Assembleur (langage)

- Langage de « seconde génération » :
  - « Première génération » = code machine binaire
- Assembleur = forme de code machine humainement lisible :
  - Nommage explicite des instructions (*mnémoniques* : MOV, ADD, ...)
  - Nommage explicite des adresses du code (*labels* : main, if, else, ...)
  - Nommage explicite des adresses de données (*variables* : var DB 5, ...)

L'Assembleur (langage)

```
boucle : add bl, 10
         jmp boucle
```

L'Assembleur (langage)



Assembleur vs code binaire (ex. 80386)

Code ASM		Code machine (binaire)
boucle : add bl, 10 jmp boucle	$\longrightarrow$	10000000 11000011 00001010 11101011 11111011

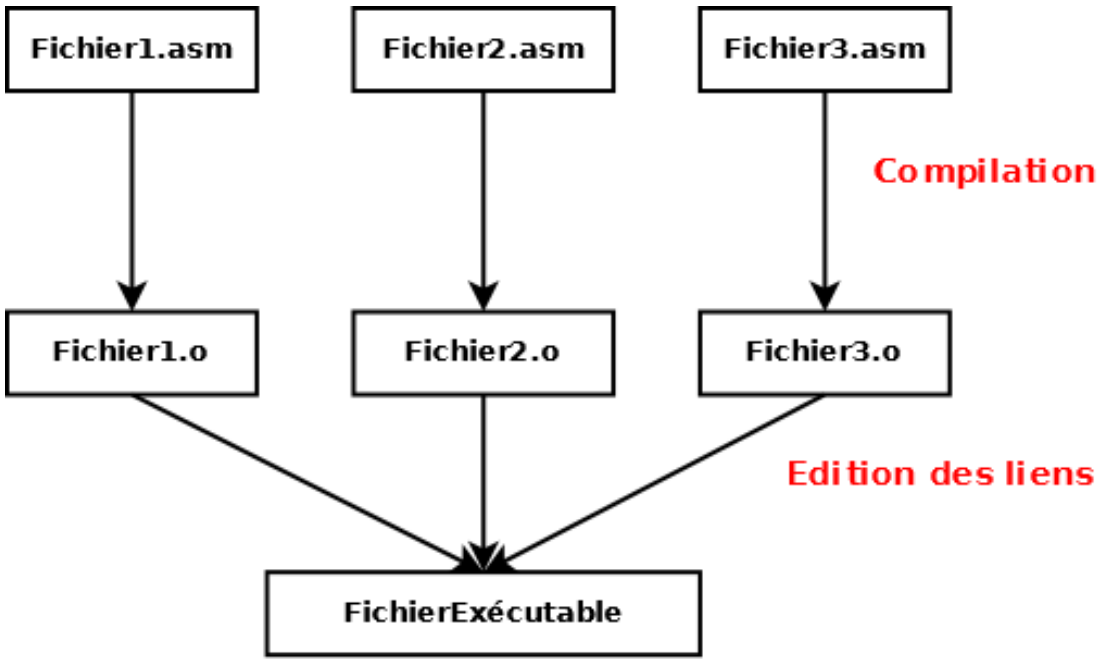
- L'assembleur est beaucoup plus facile à décrypter !

L'Assembleur (compilateur)

- *Assembleur* = compilateur du langage Assembleur
- « *Assemblage* » = compilation
- But : transformer le fichier source en code machine binaire
  - Mnémoniques traduites en instructions binaires (*opcodes*, voir chap. 7)
  - Labels traduits en adresses (ou décalages)
  - Variables traduites en adresses

Assemblage et édition des liens

- Utilisation possible de plusieurs fichiers sources pour produire un exécutable
- Phase 1 : compilation de chaque fichier source $\Longrightarrow$fichiers objets
- Phase 2 : édition des lien entre fichiers objets $\Longrightarrow$fichier exécutable



Avantages de l'assembleur

- Contrôle fin du comportement du processeur
- Permet une optimisation fine du code « à la main »
  - Programmes de petite taille
  - Programmes rapides
- Mais requiert expertise importante du programmeur

Désavantages de l'assembleur

- Clarté du code : le code ASM est moins lisible qu'un langage de plus haut niveau (C, C++, Java, ...)
- Temps de développement : plus long car programmation plus complexe qu'en langages de haut niveau
- Portabilité : moins portable car jeu d'instruction différent pour chaque processeur

Exemple : if-else

Code ASM

```
        or dword[a], 0
        jnz else
        mov dword[a], 6
        jmp endif
else : mov dword[a], 7
endif :
```

Code Java

```
if(a == 0)
   a=6;
else
   a=7;
```

Exemple : while

Code ASM

```
tant_que : or dword[a], 0
           jnz fin_tq
           inc dword[a]
           jmp tant_que
fin_tq :
```

Code Java

```
while(a ≠ 0)
   a=a+1 ;
```

Dialecte AT&T et dialecte Intel

- Deux dialectes différents pour l'assembleur x86 : AT&T et Intel
- Différences : ordre des opérateurs, spécifications de taille, ...
- Exemples :

Intel	AT&T
mov eax, 5	movl $5, %eax
mov ax, 5	movw $5, %ax
mov al, 5	movb $5, %al

L'assembleur inline

- Certains compilateurs permettent d'insérer du code ASM à l'intérieur d'un programme en langage de haut niveau
- Exemple en C avec le compilateur gcc (syntaxe AT&T) :

```
int main(){
        int i;
        for (i=0; i< 1000; ++i)
                printf("%dn\n", i);
        asm("movl␣$1,␣%eax");
        return 0;
}
```

Assembleur et performances

- On dit souvent que les programmes en assembleur sont plus rapides
- Pas toujours vrai :
  - Dépend fortement des capacités du programmeur ASM
  - Les compilateurs actuels permettent de fortement optimiser le code
    - $\Longrightarrow$ souvent code tout aussi rapide que l'assembleur
- Exemple d'omptimisation en C avec le compilateur gcc :
  - gcc -o2 fichier_source.c

Applications classiques de l'assembleur

- Code système :
  - Drivers, handlers d'interruptions, BIOS, ...
  - Certaines portions du kernel Linux sont encore en assembleur
- Micro-contrôleurs et systèmes embarqués :
  - Mais de plus en plus remplacé par le langage C
- Applications Multimedia :
  - Applications nécessitant de lourds calculs (graphiques pour jeux, etc...)
  - Nécessité de tirer parti des instructions les plus spécialisées du processeur (ex. SIMD, voir chap. 11)
- Virus

L'assembleur aujourd'hui

- Plus employé pour le développement d'applications complètes
- Employé pour certaines portions de code très précises nécessitant un optimisation fine
- Intérêt pédagogique :
  - Permet de mieux comprendre le fonctionnement du processeur

L'assembleur aujourd'hui

- Etude de Steve McIntyre sur l'ASM en packages UBUNTU et FEDORA (2013)
  - `https://wiki.linaro.org/LEG/Engineering/OPTIM/Assembly`
- 6% des packages UBUNTU contiennent de l'assembleur (pur ou inline)
- Conclusion de McIntyre : dans la majorité de cas, l'emploi de l'ASM n'y est pas indispensable