

# Assembleurs et langages d'assemblages

Eliott Duverger   Paul Monti

11 mai 2017

# Table des matières

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

jeu d'instructions  $\left\{ \begin{array}{ll} 0011\ 0110 & \Rightarrow \text{MOV AH, AL} \\ 0101\ 1110 & \Rightarrow \text{CMP BX, DX} \\ 1100\ 0001 & \Rightarrow \text{JMP exit} \end{array} \right\}$  mnémoniques

Figure 1 – Exemples imaginaires de correspondances binaire-assembleur

# Assembleur(s), assembleur(s) ?

- Assembleur : logiciel d'assemblage
  - MASM : Microsoft
  - NASM, YASM : Linux
  - Xcode : Apple
- assembleur : langage d'assemblage

Fichier C  $\Rightarrow$  Fichier asm  $\Rightarrow$  Fichier binaire

Figure 2 – Processus de compilation d'un code haut niveau

Fichier binaire  $\Rightarrow$  Fichier asm

Figure 3 – Processus de désassemblage

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux



- Marché des processeurs : **Intel** - 80% > **AMD** - 20% > Autres - <0,1%
- Architectures **von Neumann** et **Harvard** : incompatibles
- Familles **x86** et **m68k** : incompatibles
- MASM ne fonctionne qu'avec la famille x86
- Compatibilité ascendante à partir du **Intel 8086 (aka x86)** (1978)

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

# Composition d'un processeur

- CPU = UAL + Registres
- Registres : peu mais rapides d'accès

# Composition d'un registre x86

- Mémoire courte : Quelques dizaines de registres, 16 puis 32 puis 64 bits
- Chacun a une fonction
- Accessibles en lecture ou lecture écriture

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	0	0	1	1	1	1	0	1	0

Figure 4 – Registre 16 bits

# Résumé des registres de l'architecture x86

Type	Nom	16 bits	32 bits	64 bits
de travail	Accumulateur	<b>AX</b>	<b>EAX</b>	<b>RAX</b>
	Auxiliaire de base	<b>BX</b>	<b>EBX</b>	<b>RBX</b>
	Auxiliaire (compteur)	<b>CX</b>	<b>ECX</b>	<b>RCX</b>
	Auxiliaire de données	<b>DX</b>	<b>EDX</b>	<b>RDX</b>
d'index	Index de source	<b>SI</b>	<b>ESI</b>	<b>RSI</b>
	Index de destination	<b>DI</b>	<b>EDI</b>	<b>RDI</b>
	Pointeur de pile	<b>SP</b>	<b>ESP</b>	<b>RSP</b>
	Pointeur de base	<b>BP</b>	<b>EBP</b>	<b>RBP</b>
de segments	De code	<b>CS</b>	-	-
	De données	<b>DS</b>	-	-
	De pile	<b>SS</b>	-	-
	Supplémentaire	<b>ES</b>	-	-
	Pointeur d'instruction	<b>IP</b>	<b>EIP</b>	<b>RIP</b>
	Registre de flags	<b>FLAGS</b>	<b>EFLAGS</b>	<b>RFLAGS</b>

Table 1 – Registres 16 bits originaux de l'architecture x86 et leur extension

# Registres de travail

- Registres de calcul :
  - **AX** : calculs arithmétiques, paramètre d'interruption
  - **BX** : calculs arithmétiques, calculs sur des adresses
  - **CX** : compteur de boucle
  - **DX** : données destinées à des fonctions
- Manipulables par l'utilisateur
- Arguments des appels systèmes

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	1	0	0	1	1	1	0	1	0
AH								AL							

Figure 5 – Décomposition d'un registre de travail en bits de poids fort et faible

# Registres d'index, de segments

- Registres d'index : pointent vers des adresses
  - **DI, SI** : pointent vers la **D**estination et la **S**ource des manipulations
  - **SP, BP** : pointent vers la pile (**S**tack) et une position dedans (**B**ase)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	1	0	0	1	1	1	0	1	0
								<b>DIL</b>							

Figure 6 – Décomposition d'un registre d'index en bits de poids fort et faible

- Registres de segments : pointent vers des segments particuliers
  - **CS, DS** : pointent vers le programme (**C**ode) et ses **D**onnées
  - **ES, SS** : pointent vers des données et la pile
- **SS :SP** permettent de connaître le dernier élément de la pile

- Appelé aussi compteur ordinal, il s'incrémente sans cesse
- Pointe en permanence sur l'adresse de la prochaine instruction à exécuter
- **CS :IP** délimitent la prochaine instruction à exécuter



# Registre de flags

- Manipulé bit par bit et non dans son ensemble
  - **CF**, Carry Flag : retenue du résultat du calcul
  - **PF**, Parity Flag : parité du résultat du calcul
  - **ZF**, Zero Flag : égalité à 0 du résultat du calcul
  - **SF**, Sign Flag : positivité du résultat du calcul
  - **OF**, Overflow Flag : dépassement de mémoire du résultat du calcul
  - ...

14	13	12	11	10	9	8	7	6	4	2	0
NT	IOPL		OF	DF	IF	TF	SF	ZF	AF	PF	CF

Figure 7 – Décomposition du registre de flags 16 bits

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

- **Pile** : structure de donnée LIFO
- **PUSH** Registre ou Valeur
- **POP** Registre
- |      |     |
|------|-----|
| push | EAX |
| push | 4h  |
| pop  | EAX |

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

# Comparaisons

- **CMP** Registre, Opérande
- Effectue une soustraction
- Modifie les flags

mov AH, 2 mov AL, 4 cmp AL, AH	⇒	Bit	Indicateur	Valeur
		0	CF	0
		7	ZF	0
		8	SF	0

Figure 8 – Exemple de modification des flags par cmp

# Sauts (in)conditionnels

- Saut inconditionnel : **JMP** label
- Sauts conditionnels :

Indicateur	Valeur	Saut	Indicateur	Valeur	Saut
CF	1	JB	ZF	0	JNBE
CF	1	JBE	ZF	0	JNE
CF	1	JC	ZF	0	JNZ
CF	1	JNAE	PF	1	JP
CF	0	JA	PF	1	JPE
CF	0	JAE	PF	0	JNP
CF	0	JNB	PF	0	JPO
CF	0	JNC	OF	1	JO
ZF	1	JE	OF	0	JNO
ZF	1	JNA	SF	1	JS
ZF	1	JZ	SF	0	JNS

## Sauts conditionnels

```
mov AH, 2
mov AL, 4
cmp AL, AH
je exit ; Si AL=AH, ce qui est faux
jpo exit ; Si AL-AH=2 est impair, ce qui est faux
jns exit ; Si AL-AH=2 est non signe, ce qui est vrai

exit:
; instructions
```



- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

# Opérations arithmétiques

- **MOV** Contenant, Contenu
- **ADD** Destination, Source
- **SUB** Destination, Source
- **MUL** Opérande :  $AX = AX * \text{Opérande}$
- **IMUL** pour des nombres signés
- **DIV** Opérande :  $AX = AX / \text{Opérande}$
- **IDIV** pour des nombres signés
- **NEG** Opérande
- Exemple linéaire du calcul  $[(3 * 5) + 1]/2$  :

```
mov AX, 3
mul 5
add AX, 1
div 2
```

# Traiter les nombres à virgules et négatifs

- Nombres à virgule : Eliminer les virgules du calcul

- Multiplication 20 par 0.25 :

```
mov AX,25 ; AX ← 25 (car 0.25 * 100)
mul 20 ; AX ← 500 (car 25 * 20)
div 100 ; AX ← 5 (car 500 / 100)
```

- Nombres signés : complément à deux

# Opérations booléennes

- **AND** Destination, Source
- **OR** Destination, Source
- **XOR** Destination, Source
- **NOT** Destination, Source

AND	0	1	1	0	0	1	0	1
	1	1	1	0	1	1	0	0
	=	0	1	1	0	0	1	0

Figure 9 – Application de **AND**

- Exemple d'utilisation de **AND** :
- `mov AL, 27 ; 27 = Ob 0001 1011`  
`and AL, 31 ; 30 = Ob 0001 1110`  
`; AL = Ob 0001 1010 = 26`

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux

- Traitées par l'Assembleur avant la compilation : ce n'est pas de l'assembleur
- Exemples :
  - MASM : `.486` : Processeur Intel 80486
  - MASM : Structure `IF [ELSE] ENDIF`
  - NASM : `extern _printf` : va chercher le code assembleur du `printf` de la libc
  - NASM : `global _main` : définit le point d'entrée du programme par `main` :
  - NASM : `section .text` ou `.data` ou ... : découper son programme en sections
  - Enregistrer des chaînes dans des variables

- Utile pour réutiliser des blocs de texte
- Le compilateur réécrira le contenu de la macro à l'endroit de son appel
- Une macro doit être déclarée avant son premier appel

- 1 Qu'est-ce que l'assembleur ?
- 2 Processeurs et architectures
  - Différences et compatibilités
  - Registres Intel x86
- 3 Instructions des langages assembleurs
  - La pile d'instruction
  - Comparaisons et sauts
  - Arithmétique
  - Directives et macros
  - Appels systèmes Linux



- Abregés **syscalls**
- Fonctions du noyau même de l'OS
- Appelés par les processus executés
- Manipulations de fichiers, allocation de mémoire, gestion des périphériques : tout ce dont l'OS a la charge
- Section 2 du `man`

# Appels systèmes en assembleur

- Utiliser des appels systèmes en assembleur
- Liste établie des appels systèmes numérotés
  - `/usr/src/linux-headers-4.4.0-75/include/uapi/asm-generic/unistd.h`
- Numéro de l'appel système voulu en (E)AX
- Arguments si nécessaires en (E)BX, (E)CX, (E)DX, (E)SI, (E)DI
- Interruption du noyau pour appliquer : Sur Linux : `int 80h`

# Appels systèmes en assembleur

- Appel système **sys\_tee** de duplication de pipe :
  - `ssize_t tee(int fd_in, int fd_out, size_t len, unsigned int flags);`
  - `eax = 0x13b, ebx <- fd_in, ecx <- fd_out, edx <- len, esi <- flags`
- Appel système **sys\_write** :
  - `ssize_t write(int fd, const void *buf, size_t count);`
  - Salut db 'Salut !', 10 ; directive d'alias  
`mov eax, 4h ; num du syscall write (cf doc Linux)`  
`mov ebx, 1 ; std_out`  
`mov ecx, Salut`  
`mov edx, 13 ; len`  
`int 80h`

Daemon. *L'assembleur*. [Lien](#). juillet 2003.

Exit (*system call*). [Lien](#).

GeO. *Prise en main de nasm et découverte du langage d'assemblage x86 / x64*. [Lien](#). avril 2015.

*Les boucles et conditions en assembleur*. [Lien](#).

*Linux manual*. [Lien](#).

*Linux syscall reference*. [Lien](#).

*Linux system call table for x86 64*. [Lien](#).

Yontoryu. *En profondeur avec l'assembleur*. [Lien](#). Jan. 2016.