Langage assembleur sous Windows

Structure d’un système informatique

- Système d’exploitation
- Processeur(s)
- Interface unifiée vers la mémoire
- Périphériques

Microprocesseur

- Microprocesseur
  - Circuit électronique complexe
  - Identifie et exécute des instructions
- Instruction
  - Code opérateur (opcode) : action à effectuer
  - Opérandes (s)
- Registre
  - Zone de mémoire à accès ultra rapide hébergée dans le microprocesseur

Code machine

- Suite d’octets interprétée par le processeur
- Langage natif du matériel
  - Composé d’instructions et de données
  - Chaque processeur possède son propre langage machine
- Du fait de posséder quasiment les mêmes instructions, le langage le plus facile à convertir en code machine est l’assembleur.

Langage assembleur

- Langage de bas niveau
- Permet de rendre le langage machine plus compréhensible.
- Différents « dialectes »
  - NASM
  - Gas : assembleur GNU utilisé notamment avec gcc
  - MASM : Assembleur Microsoft

Opcode

- Chaque instruction commence par un nombre appelé opcode (ou code opération) qui détermine la nature de l'instruction.

Exemples :

push	0x10	6A 10
mov	eax,0x4	B8 04000000
mov	ecx,0x12345678	B9 78563412
cmp	eax,0	83F8 00
cmp	ebx,0	83FB 00

Syntaxes

- Les 2 principales syntaxes de l'assembleur sont la syntaxe Intel et la syntaxe AT&T. Les deux varient en ce qui concerne l'emploi des préfixes, la direction des opérandes et la sémantique lié aux contenus mémoire.

Intel

```
mov     eax,1
mov     ebx,0ffh
mov     eax,[ecx]
mov     eax,[ebx+3]
```

AT&T

```
movl    $1,%eax
movl    $0xff,%ebx
movl    (%ecx),%eax
movl    3(%ebx),%eax
```

RFLAGS

- Le registre rflags contient des informations d'états (masque binaire) mises à jour lors de l'exécution de la plupart des instructions de sorte à indiquer le résultat de l'instruction.

- Certains bits du registre (flags) ont une signification particulière.

- Seuls les 32 bits de la partie eflags sont utilisés.

Flags RFLAGS

- Indicateurs usuels
  - **CF** Carry Flag ou retenue
  - **PF** Parity Flag ou parité
  - **AF** Auxiliary Carry Flag ou retenue auxiliaire
  - **ZF** Zero Flag ou indicateur de zéro
  - **SF** Sign Flag, ou indicateur de signe
  - **DF** Direction Flag ou indicateur de direction
  - **OF** Overflow Flag ou indicateur de débordement de capacité
- Indicateurs de programmation système
  - **TF** Trap Flag ou indicateur de trappe
  - **IF** Interrupt Enable Flan ou indicateur d'autorisation d'interruption
  - …

Registres

- Les données dont le processeur a besoin sont stockées des registres.
- Plusieurs types de registres, chacun disposant d'une utilité particulière.
- **Registres généraux** Servent à manipuler des données, à transférer des paramètres lors des appels de fonctions et à stocker des résultats intermédiaires.
- **Registres d’offset ou pointeur** Contiennent une valeur représentant un offset à combiner avec une adresse de segment.
- **Registres de segment** Utilisés pour stocker l’adresse de départ d’un segment. Il peut s’agir de l’adresse du début des instructions du programme, du début des données, du début de la pile. etc
- **Registre de flags** Masque binaire servant à décrire le comportement d'ensemble du processeur ainsi que le résultat des instructions exécutées par lui.

Registres généraux

- **rax** accumulateur - sert à effectuer des calculs arithmétiques
- **rbx** registre auxiliaire de base - sert à effectuer des calculs arithmétiques ou des calculs sur les adresses.
- **rcx** registre auxiliaire (compteur) - sert généralement comme compteur dans des boucles.
- **rdx** registre auxiliaire (données) - sert à stocker des données destinées à des routines.
- **r8** registre général
- **r9** registre général
- …
- **r15** registre général

Registres d’offset

- **RIP** Pointeur d'instruction - est associé au registre de segment CS (CS:IP) pour indiquer la prochaine instruction à exécuter.

- **RSI** Index de source - est principalement utilisé lors d’opérations sur des chaînes de caractères ; il est associé au registre de segment DS.

- **RDI** Index de destination - est principalement utilisé lors d’opérations sur des chaînes de caractères ; il est associé au registre de segment ES.

- **RSP** Pointeur de pile - est associé au registre de segment SS (SS :SP) pour indiquer le dernier élément de la pile.

- **RBP** Pointeur de base - est associé au registre de segment SS (SS :BP) pour accéder aux données de la pile lors d’appels de sous-routines.

Registres de segment

- **CS** segment de code - indique l'adresse du début des instructions d'un programme ou d'une sous-routine
- **DS** segment de données - contient l'adresse de début des données de programmes.
- **SS** segment de pile - pointe sur la pile.
- **ES** segment supplémentaire - est utilisé par certaines instructions de copie de bloc et de traitement de chaînes de caractères.
- **FS** segment supplémentaire - réservé au système d'exploitation.
- **GS** segment supplémentaire - réservé au système d'exploitation.

Registres généraux

- Structure imbriquée des registres
  - Uniquement les registres généraux : rax, rbx, rcx, rdx
  - Partie haute et partie basse
- Exemple avec l'accumulateur :
  - **al** 8 bits b7…b0
  - **ah** 8 bits b15…b8
  - **ax** 16 bits b15…b0
  - **eax** 32 bits b31…b0
  - **rax** 64 bits b63…b0

Accès à la mémoire

- Opérateur []
  - [cetteAdresse] représente la valeur stockée à l'adresse cetteAdresse.
  - [ceRegistre] représente la valeur stockée à l'adresse contenue dans le registre ceRegistre.
  - Il est possible d'associer une étiquette (label) ceLabel à une adresse mémoire et utiliser [ceLabel].

Directives

- Instructions qui s'adressent au programme d'assemblage et qui lui permettent de produire un code exécutable compatible avec le système d'exploitation ciblé.

.data demande la création d’une zone de données

.code demande la création d’une zone de code machine exécutable

Directives de données

- Permettent de réserver de l'espace de mémoire dans les segments de données
  - **d**x données initialisées
  - **res**x données non initialisées
- La valeur du caractère x dépend de la taille des données.
  - **b** 1 octet (byte)
  - **w** 1 mot (word)
  - **d** 2 mots (double word)
  - **q** 4 mots (quadruple word)

db et directives affiliées

- Les directives db, dw, dd, et dq sont utilisées pour déclarer les données initialisées dans le fichier de sortie.

db	0x55	1 octet initialisé à la valeur 0x55
db	0x55,0x56	2 octets l'un à la suite de l'autre
db	'a',0x55	Caractère ascii 'a' (0x41) suivi de 0x55
db	'abc'	Sequence de caracteres
dw	0x1234	0x34 0x12
dw	'a'	0x41 0x00
dw	'ab'	0x41 0x42
dw	'abc'	0x41 0x42 0x43 0x00
dd	0x12345678	0x78 0x56 0x34 0x12
dq	0x1122334455667788	0x88 0x77 0x66 0x55 0x44 0x33 0x22 0x11

Définition de constantes

- La pseudo-directive EQU sert à associer un symbole à une valeur déterminée.
- Ces valeurs dépendent du compilateur et sont donc calculés lors de la compilation et non lors de l'exécution.

```
foo equ 1
mov eax,foo
← mov eax,1

message db 'hello world!'
msglen  equ $-message
mov eax,msglen
← mov eax,12
```

- Les constantes peuvent être testées à l'intérieur du programme pour permettre une compilation conditionnelle. (Attention, c'est une compilation et non une éxécution conditionnelle). Les tests se font par les pseudo-commandes IF - ENDIF. Une condition peut être égale (EQ), >= (GE), >(GT), <= (LE), <(LT), ou différente (NE).

Opérateurs et expressions

- Les opérateurs dans la plupart des assembleurs sont similaires au niveau de la forme et de la fonction à ceux utilisées en C.
- Opérateurs de calcul
    - **+** opérateur d'addition
    - **-** opérateur de soustraction
    - ***** opérateur de multiplication
    - **/** opérateur de division
    - **%** opérateur modulo
- Opérateurs bit-à-bit
    - **&** ET bit-à-bit
    - **|** OU bit-à-bit
    - **^** OU bit-à-bit exclusif
- Opérateurs de décalage de bit
    - **<<** décalage à gauche
    - **>>** décalage à droite

Expressions spéciales

- Deux opérateurs spéciaux, $ et $$, permettent des calculs à associer à la position des instructions dans le fichier de sortie.

  - **$** évalué à la position de la ligne contenant l'expression

  - **$$** évalué au début de la section courante

Outils de programmation

- **MASM** Programme d'assemblage
- **OllyDbg** Outil d'analyse dynamique de binaires exécutés en mode utilisateur
- **WinDbg** Outil d'analyse dynamique de binaires mode utilisateur et mode noyau.

La pile

- La plupart des microprocesseurs gèrent nativement une pile. Elle correspond alors à une zone de la mémoire, et le processeur retient l'adresse du dernier élément.

- Fondée sur le principe « dernier entré, premier sorti » (LIFO, *Last In, First Out*).

Rôle de la pile

- Sauvegarde des registres
- Passage de paramètres auprès des sous-routines
- Stockage des variables locales
- Stockage des adresses de retour

Instructions de pile

- push ajoute une donnée sur la pile. L'élément se trouve au *sommet* de la pile.

- push met à jour automatiquement le registre pointeur de pile : rsp - 8 (64 bits), esp - 4 (32 bits).

- pop retire l'élément au sommet de la pile.

- Comme pour push, l'instruction pop actualise d'elle-même la valeur du registre pointeur de pile: rsp + 8 (64 bits), esp + 4 (32 bits).

Instructions arithmétiques

- **add** op1,op2 op1 <- op1 + op2
- **sub** op1,op2 op1 <- op1 - op2
- **neg** reg reg <- -reg
- **inc** reg reg <- reg + 1
- **dec** reg reg <- reg - 1

Opérations sur les bits

- **and** op1,op2 op1 ← op1 & op2
- **or** op1,op2 op1 ← op1 | op2
- **xor** op1,op2 op1 ← op1 ^ op2
- **not** reg reg ← ~reg
- **shl** reg,immédiat reg ← reg << immédiat
- **shr** reg,immédiat reg ← reg >> immédiat
- **sal** reg,immédiat reg ← reg << immédiat
- **sar** reg,immédiat reg ← reg >> immédiat signé
- **rol** reg,immédiat reg ← reg decalageCirculaireGauche(imm)
- **ror** reg,immédiat reg ← reg decalageCirculaireDroite(imm)

Branchements

- Lors d'une comparaison, par exemple cmp a,b, le processeur effectue la soustraction a – b et positionne les indicateurs en fonction du résultat de l'opération.
  - ZF = zero flag = 1 si le résultat est nul, sinon ZF = 0
  - CF = carry flag =1 s'il y a une retenue, sinon CF = 0
  - SF = sign flag =1 si le résultat est négatif, sinon SF = 0
  - OF = overflow flag =1 si débordement de capacité, sinon OF = 0
- En assembleur standard, il n'y a pas de distinction entre un nombre signé ou non signé lors de sa déclaration. Ce n'est que l'instruction utilisée pour le branchement qui détermine si on le considère comme signé ou non.

Branchements inconditionnels

- call
- jmp
- ret,retn,retf (near, far)
- iret (interrupt return)

Branchements conditionnels simples

- **je** jump if equal (x = y) ZF = 1
- **jne** jump if not equal (x ≠ y) ZF = 0
- **jz** jump if zero ZF = 1
- **jnz** jump if not zero ZF = 0

Branchements conditionnels non signés

- **ja** jump above ($x > y$) CF = 0 & ZF = 0
- **jna** jump not above ($x \leq y$) CF = 1 | ZF = 1
- **jae** jump above or equal ($x \geq y$) CF = 0
- **jnae** jump not above or equal ($x < y$) CF = 1
- **jb** jump below ($x < y$) CF = 1
- **jnb** jump not below ($x \geq y$) CF = 0
- **jbe** jump below or equal CF = 1 | ZF = 1
- **jnbe** jump not below or equal ($x > y$) CF = 0 & ZF = 0

Branchements conditionnels signés

- **jg** jump greater (x > y) SF = OF & ZF = 0
- **jng** jump not greater (x ≤ y) SF ≠ OF & ZF = 1
- **jge** jump greater or equal (x ≥ y) SF = OF
- **jnge** jump not greater or equal (x < y) SF ≠ OF
- **jl** jump less (x < y) SF ≠ OF
- **jnl** jump not less (op1 ≥ op2) SF = OF
- **jle** jump less or equal (x ≤ y) SF ≠ OF
- **jnle** jump not less or equal (x > y) SF = OF & ZF = 0

Branchements conditionnels sur indicateurs

- **jc** jump if carry CF = 1
- **jnc** jump if not carry CF = 0
- **jo** jump if overflow OF = 1
- **jno** jump if not overflow OF = 0
- **jp** jump if parity PF = 1
- **jnp** jump if not parity PF = 0
- **jpo** jump if parity odd PF = 0
- **js** jump if sign SF = 1
- **jns** jump if no sign SF = 0

Branchements conditionnels sur compteurs

- jcxz jump if cx = 0
- jecxz jump if ecx = 0

Tableaux

- Collection d'octets contigus
- Réunit des données de même type et de même taille.
- L'adresse mémoire de chaque élément du tableau est calculée en fonction de
  - l'adresse du tableau (autrement dit du premier élément du tableau)
  - le nombre d'octets de chaque élément
  - l'indice de l'élément (sa position au sein du tableau)
- Chaîne de caractères : tableau d'octets dont le dernier élément est 0.

Exemples de définition de tableau

.data

; définit un tableau de 10 doubles mots initialisés

a1 **dd** 1,2,3,4,5,6,7,8,9,10

; définit un tableau de 10 quadruples mots initialisés

a1 **dq** 0,0,0,0,0,0,0,0,0,0

; idem que précédemment avec dup

a1 **dq** 10 **dup**(0)

Structures de controle

- if
- switch
- while
- do while
- for

if

```
if (a > b) {
  . . .
}
else {
  . . .
}
```

```
if1:
  cmp eax,ebx
  jng else1
  . . .
  jmp endif1
else1:
  . . .
endif1:
```

switch

```
switch (i) {
 case 1:
  . . .
  break;
 case 2:
  . . .
  break;
 default:
  . . .
}
```

```
switch1:
 cmp ecx,1
 jne case2
 . . .
 jmp endswitch1
case2:
 cmp ecx,2
 jne default1
 . . .
 amp endswitch1
default1:
 . . .
endswitch1:
```

while, do while

```
while (x) {
  . . .
 x--;
}
```

```
while1:
 cmp eax,0
 jle endwhile1
 . . .
 dec eax
 jmp while1
```

```
do {
  . . .
 x--;
} while (x);
```

```
do2:
 . . .
 dec eax
 cmp eax,0
 jg  do2
```

for

```
for (i = 1;
     i < 10;
     i++) {
  . . .
}
```

```
for1:
 mov ecx,1
 jmp test1
next1:
  . . .
 inc ecx
test1:
 cmp ecx,10
 jl  next1
```

Microsoft Windows

- A l'origine, un environnement graphique au dessus de MS-DOS. Par la suite, un système d'exploitation complet.

- Fonctionne à l'heure actuelle sur une large gamme de machines :
  - ordinateurs personnels
  - serveurs
  - tablettes
  - smartphones
  - consoles (version spécialisée pour Xbox)

API Windows

- Interface de programmation (API, *Application Programming Interface*)

- Ensemble normalisé de fonctions

- Permettent aux logiciels applicatifs de se servir des fonctionnalités de Microsoft Windows

- Appellables depuis le mode utilisateur

- Rendues visibles via diverses bibliothèques (DLL)

DLL fondamentales de Windows

- **Kernel32.dll** Met en oeuvre la plus grande partie de l'API Windows : gestion des processus et des threads, interactions de niveau fichier, etc.

- **Kernelbase.dll** Refactorisation de Kernel32 à partir de Windows Vista.

- **User32.dll** Fonctions d'interaction avec l'interface graphique mode utilisateur.

- **Gdi32.dll** Manipulation des dispositifs d'impression et d'affichage (écran).

- **Ntdll.dll** Bibliothèque générale de support ; contient les appels systèmes.

ABI Windows

- Interface binaire-programme
- Convention d’appel
- Format de fichier pour les exécutables et bibliothèques

Convention d’appel

- Définit la manière d'appeler une fonction.
- Varie selon le langage de programmation, le compilateur, et parfois l'architecture du processeur cible.
- Au niveau de l’assembleur, les conventions diffèrent d’un programme d’assemblage à l’autre.
- Pour s’interfacer avec un langage de plus haut niveau (par exemple C), l’assembleur doit suivre la convention d’appel associée à ce langage.

Contenu de la convention d’appel

- **Méthode de récupération des paramètres** Où le code appelant place-t-il les paramètres : sur la pile ou dans des registres ?

- **Ordre d'empilage** Quand les paramètres sont placés sur la pile, dans quel ordre y sont-ils placés : du premier au dernier ou du dernier au premier ?

- **Procédure de nettoyage** Quand les paramètres sont placés sur la pile, qui doit nettoyer la pile suite à l’appel : le code appelant ou le code du sous programme ?

- **Registres préservées ou non** Quels sont les registres préservés (callee save) que le sous programme doit mémoriser avant d’utiliser et restaurer à la fin ? Quels registres l'appelant (caller save) a sauvegardé et peuvent en l'occurence être utilisés par le sous programme à volonté ?

- **Méthode de récupération de la valeur retournée** Où se trouve la valeur de retour d’une fonction ?

Conventions d’appel sous Windows

- **cdecl** Convention d'appel par défaut du C
- **stdcall** Utilisée dans les versions (32 bits) x86 de Windows
- **fastcall** Employée dans les déclinaisons 64 bits de Windows
- **thiscall** Fonctions membres C++

stdcall

- Les paramètres sont passés par la pile, de droite à gauche
- Les registres EAX, ECX et EDX sont réservés à l'usage de la fonction
- La valeur de retour se trouve dans le registre EAX
- Le sous programme doit retirer les paramètres de la pile.

Exemple d’appel stdcall

```
int WINAPI MessageBox(
 _In_opt_ HWND    hWnd,
 _In_opt_ LPCTSTR lpText,
 _In_opt_ LPCTSTR lpCaption,
 _In_     UINT    uType
);

push MB_OK              ; uType
push offset szWndTitle ; lpCaption
push offset szWndText  ; lpText
push 0                 ; hWnd
call MessageBoxA
```

Appel de sous programme

- De façon générique, un **sous-programme** est un sous-ensemble du programme dans sa hiérarchie fonctionnelle.

- A la fin de l'exécution d'un sous programme, l'exécution doit se poursuivre avec l'instruction qui suit l'appel.

- Contrôle du flux d'exécution

  - **call** op Mémorisation de l'adresse de l'instruction suivante (par le biais de la pile) et saut inconditionnel à op

  - **ret** Retourne à l'adresse identifiée par le pointeur de pile courant et ajuste ce pointeur en conséquence

- Attention à la gestion de la pile !

CALL / JMP / RET

- Sur le plan fonctionnel, les instructions suivantes sont identiques.

```
call l1
  . . .
l1:
 ret
```

```
push l1
jmp  l2
  . . .
l1:
 ret
l2:
```

Variables locales

- On alloue l'espace requis par les variables locales en diminuant le registre pointeur de pile (rsp ou esp).
- Chaque emplacement de pile correspond alors potentiellement à la valeur d'une variable :
  - [rsp], [rsp+8], ...
  - [esp], [esp+4], ...
- A la fin du sous programme, on libère l'espace mémoire correspondant :
  - mov rsp,rbp
  - mov esp,ebp

Valeur de retour des fonctions

- Elles sont passées par des registres :
  - accumulateur (rax/eax) pour un pointeur ou un type entier
  - couple rdx:rax (edx:eax) dans le cas d'une valeur 128 bits
  - xmm0 pour une valeur flottante ou xmm1:xmm0 si besoin

Un programme MASM simple

```
; msgbox.asm

.386
.model flat, stdcall
option casemap:none

      include \masm32\include\windows.inc
      include \masm32\include\user32.inc
      include \masm32\include\kernel32.inc

      includelib \masm32\lib\user32.lib
      includelib \masm32\lib\kernel32.lib

.data
      szWndTitle  db    "Wnd Title",0
      szWndText   db    "Wnd Text",0

.code

start:

      push  MB_OK
      push  offset szWndTitle
      push  offset szWndText
      push  0
      call  MessageBoxA

      push  0
      call  ExitProcess

end   start
```

Transformation asm vers exe

- Assemblage

  \masm32\bin\ml /c /coff /nologo msgbox.asm

- Edition des liens

  \masm32\bin\link /SUBSYSTEM:WINDOWS msgbox.obj

- Attention aux permissions du fichier !