Logiciel de base, filière apprentissage
Introduction

Ensimag 1A Apprentissage

Matthieu Moy

Matthieu.Moy@imag.fr

2013

Du C au langage d'assemblage

- C (ou Ada, Java, C++, ...) ...)
  - Langage de haut niveau d'abstraction
  - Permet de s'abstraire des détails techniques du processeur
- Langage machine
  - Format des instructions qui seront combinées selon des règles imposées par le processeur
- Entre les deux : langage d'assemblage
  - Les mêmes instructions décrites sous une forme plus compréhensible pour un humain

Grenoble INP ensimag

Assembleur

- Programme qui traduit un code du langage d'assemblage au langage machine
- Code source (langage de haut niveau ou d'assemblage)
  - Instructions d'un programme écrit dans 1 langage non directement exécutable par le processeur
  - Peut être interprété ou compilé
- Code objet (langage machine)
  - Suite de bits, peu lisible (impression en hexadécimal sur les listings), difficile à utiliser

Grenoble INP ensimag

Interprétation

Un programme appelé interprète[ur] lit le programme source et exécute ses instructions au fur et à mesure
Caractéristiques :

- Facile à réaliser
- Lent
- Bien adapté à l'interactivité (débogage)

Compilation

Un programme appelé compilateur traduit le programme source en un programme objet exécutable sur une machine donnée
Caractéristiques :

- Complexe à réaliser
- Exécution beaucoup plus rapide
- Les programmes exécutés fréquemment sont stockés sous format objet (bibliothèques)

Exemples

- Langages interprétés :
  - Langages shell
  - PostScript
  - Python, SQL, HTML, XML, JavaScript, PERL, ...
- Langages compilés :
  - Ada
  - C et C++
- Langage semi-interprété :
  - Java, Scala
  - C#
  - JavaScript selon l'implémentation

Logiciel de base

- Regroupe ces programmes (interpréteurs + compilateurs) permettant d'exécuter un programme source
- A mi-chemin entre les aspects matériel et logiciel :
  - Architecture
  - Algorithmique et programmation

Grenoble INP ensimag

Le reste de la séance

- Quelques mots sur les entrées/sorties
- Gestion de la mémoire
- Exécution des instructions (langage d'assemblage)
- Exemple : traduction de quelques structures de C en langage d'assemblage

Entrées-sorties

Échange de données et programmes depuis l'extérieur vers le couple processeur-mémoire

- Réalisées par :
  - Le processeur lui-même
  - Un processeur auxiliaire (processeur vidéo par ex.)
- Accès à plusieurs types de support :
  - Mémoire secondaire (disque)
  - Périphériques : clavier, écran, support amovible (disquette, bande), imprimante, ...

Mémorisation (1)

Distinction entre emplacements et adresses :

- Adresse d'un emplacement :
  - Accès et manipulation de son contenu
  - Lui est attachée une fois pour toutes
- Contenu d'un emplacement :
  - Modifiable
  - Suite de bits sans signification intrinsèque
  - Sens donné par l'instruction qui le manipule

Mémorisation (2)

Taille d'un emplacement (en général multiple de 8 bits 1 octet)

Octet (byte) : suffixe b dans les instructions (movb, addb)

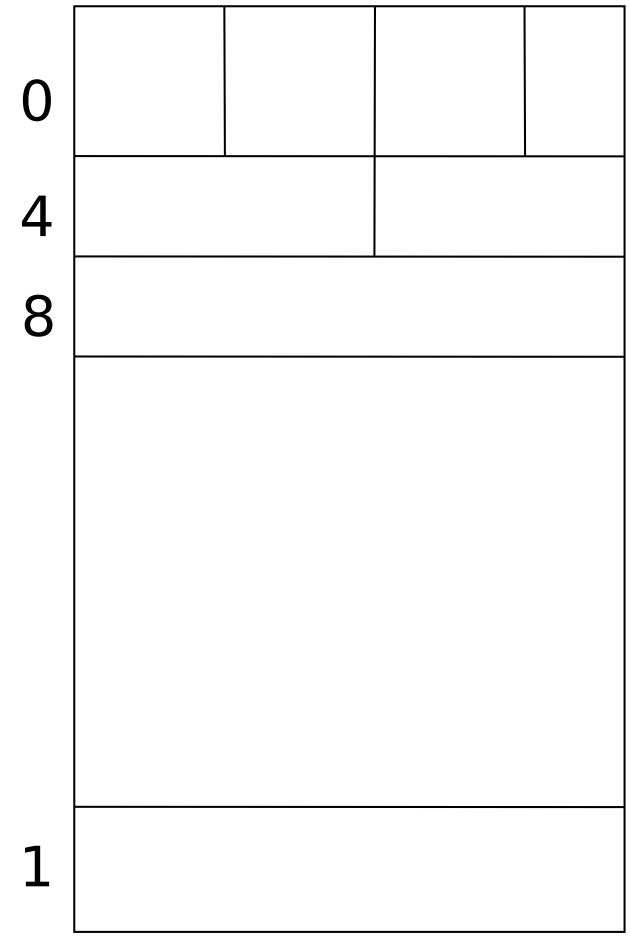
Mot (word) : 2 octets, suffixe w dans les instructions (movw, addw)

Double mot (long word) : 4 octets, suffixe l dans les instructions (movl, addl)

Mémorisation (3)

Exemple Intel (> 80386)

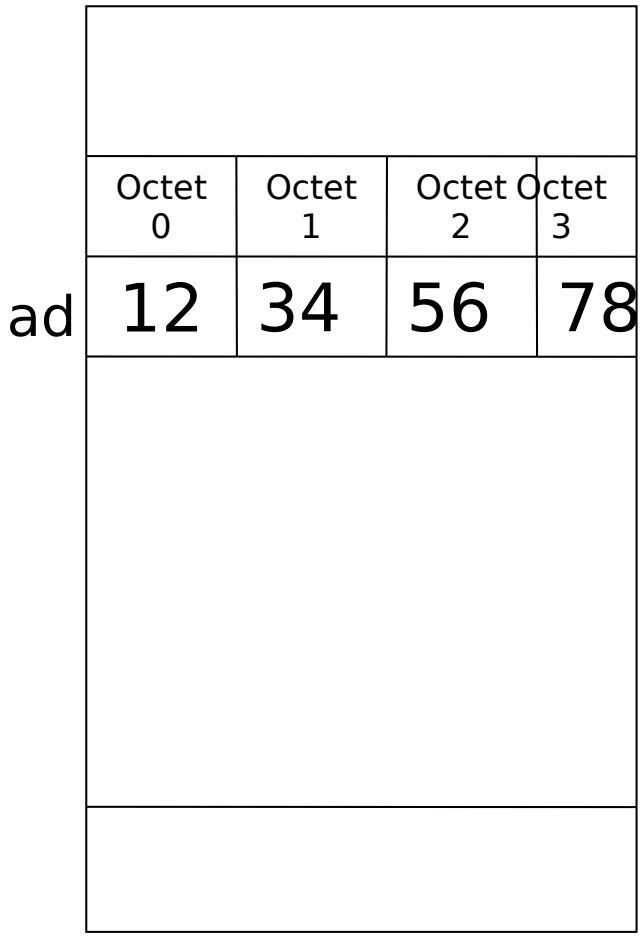
- Deux espaces mémoires séparés : données et programmes
- Espace ~ tableau de 4Goctets (4 x 2^30) indexés de 0 à 2^32 - 1
- Index dans le tableau = adresse = entier non signé sur 32 bits
- Emplacements adressables : octets, mots (16 bits), double mots (32 bits)



Mémorisation (4)

Exemple Intel (> 80386) :

- Adresse d'un mot ou double mot = octet de poids faible (little endian)
- Contenus du :
  - mot d'adresse ad : ??
  - double mot d'adresse ad : ??

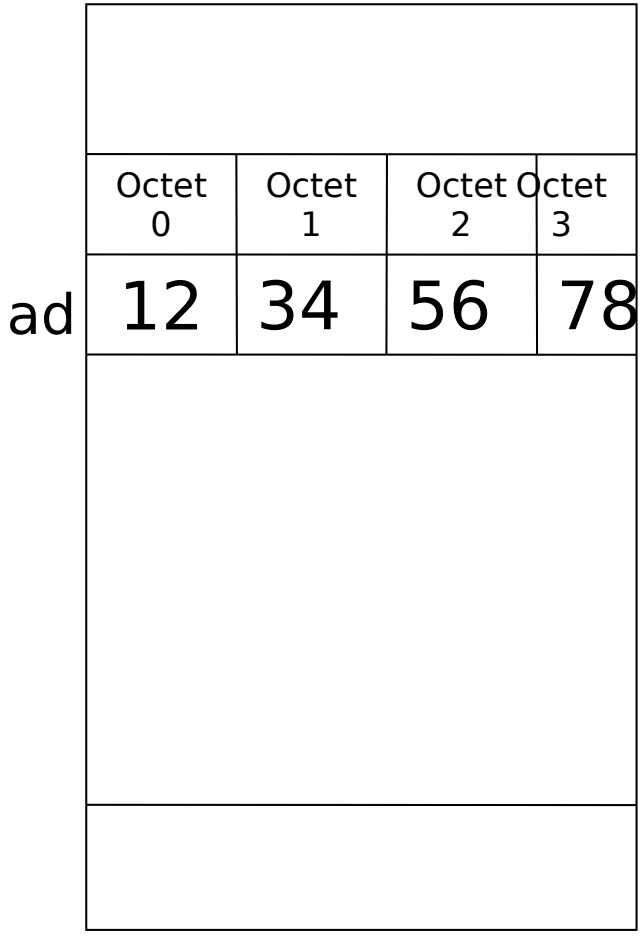


Mémorisation (4)

Exemple Intel (> 80386) :

- Adresse d'un mot ou double mot = octet de poids faible (little endian)
- Contenus du :
  - mot d'adresse ad : 3412
  - double mot d'adresse ad : 78563412
  - Rq : gdb présente les données de façon plus intelligible

Attention : signification du contenu dépend du contexte : instruction, donnée (de quel type ?)



Séquencement et exécution des instructions

- Exécution d'une instruction :
  - Chargement du Registre Instruction (RI)
    - ★ Contient l'instruction en cours
  - Décodage et exécution de l'instruction :
    - ★ Détermination de l'adresse des opérandes (si nécessaire)
    - ★ Chargement des opérandes depuis la mémoire (si...)
    - ★ Exécution de l'instruction
    - ★ Rangement du résultat en mémoire (si...)
    - ★ Incrémentation du pointeur d'instructions (EIP)

  Exécution simultanée possible (mode pipe-line)

Séquencement et exécution des instructions

Exemple :
Ici : `movl $Ici, %eax`
$\rightsquigarrow$ copie l'adresse d'Ici dans le registre EAX

- Détermination de l'adresse des opérandes (si nécessaire)
- Chargement des opérandes depuis la mémoire (si...)
- Exécution de l'instruction
- Rangement du résultat en mémoire (si...)
- Incrémentation du pointeur d'instructions (EIP)

Registres des processeurs Intel

- Sur 8, 16 ou 32 bits

Notation	31 16	15 0		Nom usuel
EAX		AX		Accumulator
		AH	AL	
EBX		BX		Base Index
		BH	BL	
ECX		CX		Count
		CH	CL	
EDX		DX		Data
		DH	DL	
ESP				Stack Pt.
EBP				Base Pt.
EDI		DI		Dest. Index
ESI		SI		Source Index
EIP				Instruction Pt.
EFLAGS		FLAGS		Flag

Registres spécialisés

EIP *Extended Instruction Pointer* = @ de l'instruction suivante

- Modifié par les sauts, appels/retours de sous-programmes

ESP *Extended Stack Pointer* = @ de la pile (mémoire données)

EFLAGS = registre d'état

- Bit 0 : retenue (C : carry)
- Bit 6 : zéro (Z)
- Bit 7 : signe (S)
- Bit 11 : débordement (O : overflow)

Les autres registres

- Utilisés comme des « variables » dans les programmes assembleurs
- Fonctions multiples :
  - Adressage de données en mémoire données
  - @ d'un emplacement en mémoire programme
  - Spécialisation pour certaines instructions ou modes d'adressage
  - cf. cours suivants

Traduction C -> langage d'assemblage (1)

```
x = x + 42;
```

- Charger x dans un registre
- Ajouter 42 au registre
- Stocker le résultat en mémoire

```
movl x, %eax
addl $42, %eax
movl %eax, x
```

Traduction C $\rightsquigarrow$ langage d'assemblage (2)

Conditionnelle

if (<*expression*>) <*instr1*> else <*instr2*>

```
  <calcul valeur expression, res dans \%eax>
  cmpl $0,%eax
  je else
  <instr1>
  jmp finsi
else:
  <instr2>
finsi:
```

Traduction C $\rightsquigarrow$ langage d'assemblage (3)

Boucle

while (<*expression*>) <*instructions*>

```
while:
   <calcul valeur de l'expression, res dans %eax>
   cmpl $0, %eax
   je finwhile
   <instructions>
   jmp while
finwhile:
```

Traduction C $\rightsquigarrow$ langage d'assemblage (4)

Boucle `for`

for (<*instr1*> ; <*expr*> ; <*instr2*>) <*instrs*>

```
  <instr1>
iter:
  <calcul valeur de l'expression, res dans %eax>
  cmpl $0, %eax
  je finfor
  <instrs>
  <instr2>
  jmp iter
finfor:
```