Assembleur

Licence informatique – semestre 4

Section 1

Présentation de l'unité

Anne Parrain - anne.parrain@univ-artois.fr
Bureau C304

- Responsable de l'unité Assembleur
- Responsable du semestre 4 de la licence informatique

Semestre 4

- l'emploi du temps est disponible sur **`http://edtlens.univ-artois.fr/`**
- les TPs d'assembleur commenceront en semaine 3

Objectifs de l'unité

Dans cette unité, vous allez :

- revenir sur la représentation de l'information
- étudier le langage assembleur :
  - comprendre la notion de registre
  - connaître les principales instructions
  - comprendre l'utilisation de la pile des appels
  - comprendre comment se fait le passage de paramètres

Crédits et charge horaire

ECTS	4
Heures hebdomadaires	3h15 / semaine le plus souvent
CM et TD	1h45/sem $\times$ 12 sem (Semaines 1–12)
TP	1h30/sem $\times$ 10 sem (Semaines 3–12)

Modalités de contrôle de connaissance

- contrôle continu : au minimum deux contrôles sur le semestre en TD ou en TP
- un examen terminal en session 1 comme en session 2
- la formule de calcul de la note de l'unité est :

$$\max(\text{examen}, (\text{examen} + \text{cc})/2)$$

L'unité en ligne sur la plateforme Moodle de l'université

Moodle : **`http://foad.univ-artois.fr/`**

Vous trouverez :

- support des cours ;
- exercices ;
- contrôles continus ;
- notes.

Clé d'inscription : assembleur

Section 2

Rappels sur la représentation et le stockage de l'information

La représentation de l'information – À vous !

Un bit

- un bit : unité élémentaire d'information, qui ne peut prendre que deux valeurs
  - 0 ou 1
  - Vrai ou Faux
  - Noir ou Blanc
  - ...
- *deux bits permettent de coder combien de valeurs ?*
- *trois bits ?*

La représentation de l'information – À vous !

Un bit

- un bit : unité élémentaire d'information, qui ne peut prendre que deux valeurs
  - 0 ou 1
  - Vrai ou Faux
  - Noir ou Blanc
  - ...
- *deux bits permettent de coder combien de valeurs ?*
  $2^2 = 4$ valeurs de 0 à $2^2 - 1$
- *trois bits ?*

La représentation de l'information – À vous !

Un bit

- un bit : unité élémentaire d'information, qui ne peut prendre que deux valeurs
  - 0 ou 1
  - Vrai ou Faux
  - Noir ou Blanc
  - ...
- *deux bits permettent de coder combien de valeurs ?*
  $2^2 = 4$ valeurs de 0 à $2^2 - 1$
- *trois bits ?*
  $2^3 = 8$ valeurs de 0 à $2^3 - 1$

La représentation de l'information (2) – À vous !

Un octet

- la brique élémentaire pour accéder à l'information est l'octet, qui contient 8 bits
  - *combien de valeurs permet de coder un octet ?*
  - *quelle est la valeur en binaire de 135 ?*
  - *quelle est la valeur en décimal de 00110101 ?*
  - *quel est l'algorithme pour passer de la base 10 à la base 2 ? vous retournerez le résultat sous la forme d'une liste ou d'une chaîne de caractères, au choix*

La représentation de l'information (2) – À vous !

Un octet

- la brique élémentaire pour accéder à l'information est l'octet, qui contient 8 bits
  - *combien de valeurs permet de coder un octet ?*
    $2^8 = 256$ valeurs de 0 à $2^8 - 1$
  - *quelle est la valeur en binaire de 135 ?*
  - *quelle est la valeur en décimal de 00110101 ?*
  - *quel est l'algorithme pour passer de la base 10 à la base 2 ? vous retournerez le résultat sous la forme d'une liste ou d'une chaîne de caractères, au choix*

La représentation de l'information (2) – À vous !

Un octet

- la brique élémentaire pour accéder à l'information est l'octet, qui contient 8 bits
  - *combien de valeurs permet de coder un octet ?*
    $2^8 = 256$ valeurs de 0 à $2^8 - 1$
  - *quelle est la valeur en binaire de 135 ?*
    10000111
  - *quelle est la valeur en décimal de 00110101 ?*
  - *quel est l'algorithme pour passer de la base 10 à la base 2 ? vous retournerez le résultat sous la forme d'une liste ou d'une chaîne de caractères, au choix*

La représentation de l'information (2) – À vous !

Un octet

- la brique élémentaire pour accéder à l'information est l'octet, qui contient 8 bits
  - *combien de valeurs permet de coder un octet ?*
    $2^8 = 256$ valeurs de 0 à $2^8 - 1$
  - *quelle est la valeur en binaire de 135 ?*
    10000111
  - *quelle est la valeur en décimal de 00110101 ?*
    53
  - *quel est l'algorithme pour passer de la base 10 à la base 2 ? vous retournerez le résultat sous la forme d'une liste ou d'une chaîne de caractères, au choix*

Conversion base 10 vers vase 2

```
1 def conversion_decimal_vers_binaire(n) :
2   '''int -> str'''
3   res = ''
4   while n > 0 :
5     res = str(n%2)+res
6     n = n//2
7   return res
```

```
>>> conversion_decimal_vers_binaire(56)
'111000'
>>> conversion_decimal_vers_binaire(135)
'10000111'
```

La base 16 – À vous !

- Une autre base couramment utilisée est la base hexadécimale ou base 16
  - son alphabet est 0, 1, ... 9, A, B, ... F
  - *combien de bits un chiffre en base 16 représente-t-il ?*
  - *combien faut-il de chiffres en base 16 pour représenter un octet ?*
  - *quelle valeur en décimal représente F9E8 ? en binaire ?*
- les bits de poids faible sont les bits représentés à droite
- les bits de poids fort sont les bits représentés à gauche
- de la même façon, on parle de partie haute et de partie basse de n bits d'un nombre pour désigner les n bits de poids fort ou les n bits de poids faible

La base 16 – À vous !

- Une autre base couramment utilisée est la base hexadécimale ou base 16
  - son alphabet est 0, 1, ... 9, A, B, ... F
  - *combien de bits un chiffre en base 16 représente-t-il ?*
    $16 = 2^4$ donc 4 bits
  - *combien faut-il de chiffres en base 16 pour représenter un octet ?*
  - *quelle valeur en décimal représente F9E8 ? en binaire ?*

- les bits de poids faible sont les bits représentés à droite
- les bits de poids fort sont les bits représentés à gauche
- de la même façon, on parle de partie haute et de partie basse de n bits d'un nombre pour désigner les n bits de poids fort ou les n bits de poids faible

La base 16 – À vous !

- Une autre base couramment utilisée est la base hexadécimale ou base 16
  - son alphabet est 0, 1, ... 9, A, B, ... F
  - *combien de bits un chiffre en base 16 représente-t-il ?*
    $16 = 2^4$ donc 4 bits
  - *combien faut-il de chiffres en base 16 pour représenter un octet ?*
    un octet, c'est 8 bits, donc 2 chiffres en base 16 représentent un octet
  - *quelle valeur en décimal représente F9E8 ? en binaire ?*

- les bits de poids faible sont les bits représentés à droite
- les bits de poids fort sont les bits représentés à gauche
- de la même façon, on parle de partie haute et de partie basse de n bits d'un nombre pour désigner les n bits de poids fort ou les n bits de poids faible

La base 16 – À vous !

- Une autre base couramment utilisée est la base hexadécimale ou base 16
  - son alphabet est 0, 1, ... 9, A, B, ... F
  - *combien de bits un chiffre en base 16 représente-t-il ?*
    $16 = 2^4$ donc 4 bits
  - *combien faut-il de chiffres en base 16 pour représenter un octet ?*
    un octet, c'est 8 bits, donc 2 chiffres en base 16 représentent un octet
  - *quelle valeur en décimal représente F9E8 ? en binaire ?*
    en décimal : $8.16^0 + 14.16^1 + 9.16^2 + 15.16^3 = 63976$

- les bits de poids faible sont les bits représentés à droite
- les bits de poids fort sont les bits représentés à gauche
- de la même façon, on parle de partie haute et de partie basse de n bits d'un nombre pour désigner les n bits de poids fort ou les n bits de poids faible

La base 16 – À vous !

- Une autre base couramment utilisée est la base hexadécimale ou base 16
  - son alphabet est 0, 1, ... 9, A, B, ... F
  - *combien de bits un chiffre en base 16 représente-t-il ?*
    $16 = 2^4$ donc 4 bits
  - *combien faut-il de chiffres en base 16 pour représenter un octet ?*
    un octet, c'est 8 bits, donc 2 chiffres en base 16 représentent un octet
  - *quelle valeur en décimal représente F9E8 ? en binaire ?*
    en décimal : $8.16^0 + 14.16^1 + 9.16^2 + 15.16^3 = 63976$
    en binaire : 1111 1001 1110 1000
- les bits de poids faible sont les bits représentés à droite
- les bits de poids fort sont les bits représentés à gauche
- de la même façon, on parle de partie haute et de partie basse de n bits d'un nombre pour désigner les n bits de poids fort ou les n bits de poids faible

Exemple en base 16 – À vous !

Soit le nombre `F5D3221A`

- son bit de poids le plus fort est :
- son bit de poids le plus faible est :
- sa partie haute de 8 bits est :
- sa partie basse de 16 bits est :

Exemple en base 16 – À vous !

Soit le nombre F5D3221A

- son bit de poids le plus fort est :
  1
- son bit de poids le plus faible est :
- sa partie haute de 8 bits est :
- sa partie basse de 16 bits est :

Exemple en base 16 – À vous !

Soit le nombre `F5D3221A`

- son bit de poids le plus fort est :
  1
- son bit de poids le plus faible est :
  `A` est représenté par 1010 en binaire, donc 0
- sa partie haute de 8 bits est :
- sa partie basse de 16 bits est :

Exemple en base 16 – À vous !

Soit le nombre `F5D3221A`

- son bit de poids le plus fort est :
  1
- son bit de poids le plus faible est :
  `A` est représenté par 1010 en binaire, donc 0
- sa partie haute de 8 bits est :
  `F5`
- sa partie basse de 16 bits est :

Exemple en base 16 – À vous !

Soit le nombre `F5D3221A`

- son bit de poids le plus fort est :
  `1`
- son bit de poids le plus faible est :
  `A` est représenté par 1010 en binaire, donc 0
- sa partie haute de 8 bits est :
  `F5`
- sa partie basse de 16 bits est :
  `221A`

La taille des cases mémoire

- la plus petite taille d'une case mémoire est l'octet (8 bits)
- mais avec l'évolution des processeurs, on peut accéder à des cases mémoire :
  - de 16 bits (2 octets) : un mot
  - de 24 bits (3 octets) : plutôt dans le cadre du traitement d'images
  - de 32 bits (4 octets) : un mot double, ou mot long
  - de 64 bits (8 octets) : un mot quadruple

L'adressage de la mémoire – À vous !

L'adressage de la mémoire

Chaque mot en mémoire a une adresse. La taille de la mémoire peut être limitée par les possibilités d'adressage de la mémoire.

- Une adresse sur 8 bits peut permettre d'accéder à 2^8 cases mémoire.
- Si on considère des mots de 2 octets, quel espace (en octets) peut-on adresser ?

- Les processeurs actuels manipulent des adresses sur 64 bits (8 octets).
- Si on considère des mots de 1 octet, quel espace (en octets) peut-on adresser ?

L'adressage de la mémoire – À vous !

L'adressage de la mémoire

Chaque mot en mémoire a une adresse. La taille de la mémoire peut être limitée par les possibilités d'adressage de la mémoire.

- Une adresse sur 8 bits peut permettre d'accéder à 2^8 cases mémoire.
- Si on considère des mots de 2 octets, quel espace (en octets) peut-on adresser ?
  $2*2^8$ octets = 2^9 octets = 512 octets
- Les processeurs actuels manipulent des adresses sur 64 bits (8 octets).
- Si on considère des mots de 1 octet, quel espace (en octets) peut-on adresser ?

L'adressage de la mémoire – À vous !

L'adressage de la mémoire

Chaque mot en mémoire a une adresse. La taille de la mémoire peut être limitée par les possibilités d'adressage de la mémoire.

- Une adresse sur 8 bits peut permettre d'accéder à 2^8 cases mémoire.
- Si on considère des mots de 2 octets, quel espace (en octets) peut-on adresser ?
  $2*2^8$ octets = 2^9 octets = 512 octets
- Les processeurs actuels manipulent des adresses sur 64 bits (8 octets).
- Si on considère des mots de 1 octet, quel espace (en octets) peut-on adresser ?
  on pourra donc adresser 2^{64} octets, soit 16.2^{60} octets, soit plus de 16.10^{18} octets, soit 16 000 peta-octets, soit 16 millions de tera-octets, soit 16 milliards de giga-octets.

Section 3

Programmer en langage assembleur

Programme en langage machine

- langage machine : langage natif d'un processeur
- c'est une suite d'instructions machine à exécuter
- les instructions sont en binaire
- les instructions machine sont de taille différentes : elles contiennent toutes les informations nécessaires (paramètres) à leur exécution
- le programme contient donc la suite des adresses où chaque instruction commence

Programme en langage assembleur

- premier langage au-dessus du langage machine compréhensible par un humain
- structure un peu plus simple des instructions
- structure d'un programme en langage assembleur :
  - une ligne = une instruction
  - adresse d'une instruction = numéro de la ligne
- on peut nommer des adresses : adresses symboliques ou liens
- on peut nommer des valeurs (numériques, chaînes de caractères, etc. . .) : constantes
- une instruction : *mnémonique suite-d-opérandes*
  - le mnémonique correspond à une instruction : ADD, MOV, JP, . . .
  - il est suivi, le cas échéant, de ses paramètres
- il existe plusieurs syntaxes : Intel, AT&T . . .
- nous allons travailler en syntaxe AT&T, avec l'assembleur gas qui est inclus dans le compilateur gcc

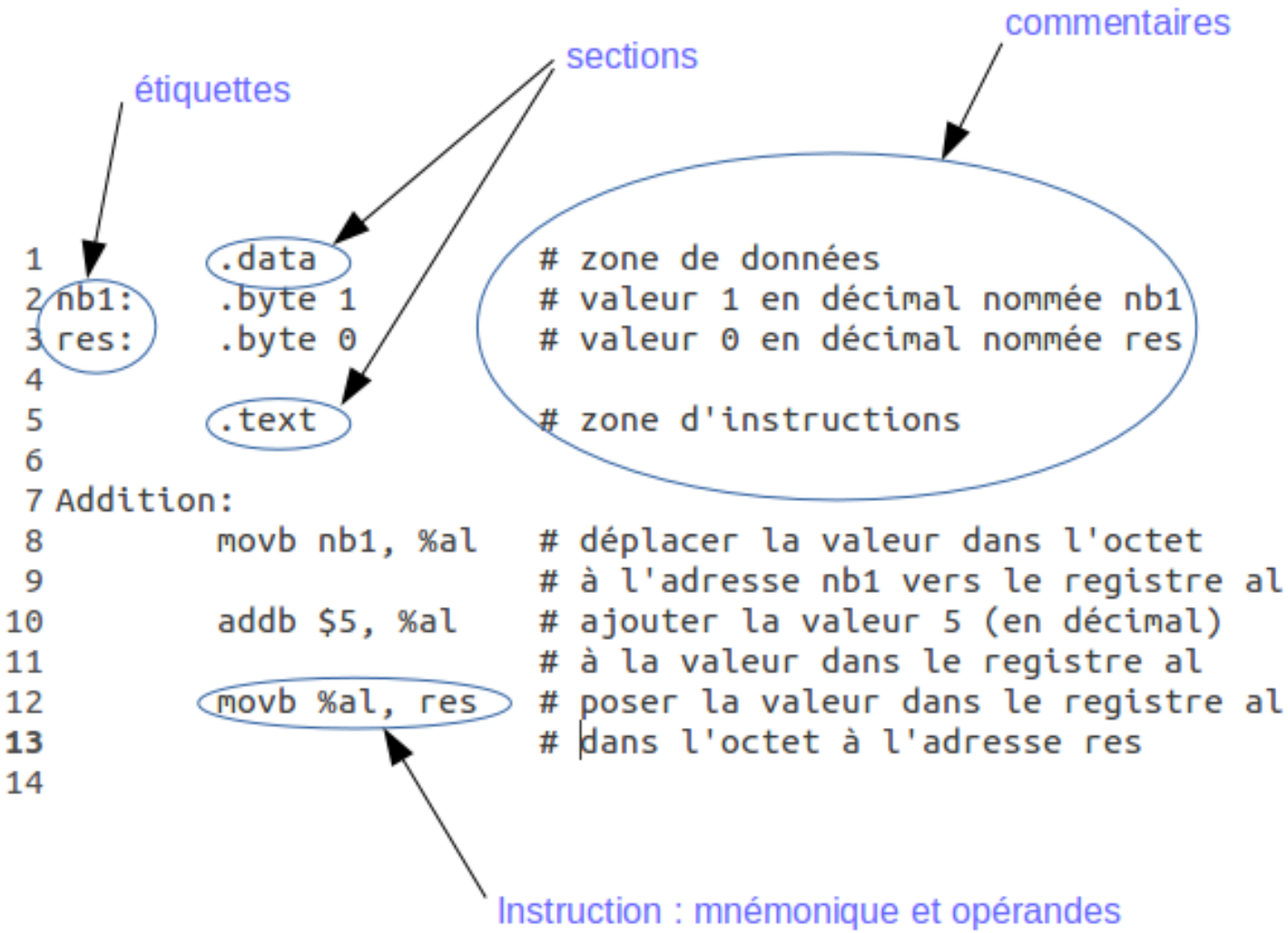
Un premier exemple en assembleur

Le code en assembleur : `src/premier.s`

```
1         .data             # zone de données
2 nb1:    .byte 1           # valeur 1 en décimal nommée nb1
3 res:    .byte 0           # valeur 0 en décimal nommée res
4
5         .text             # zone d'instructions
6
7 Addition:
8         movb nb1, %al     # déplacer la valeur dans l'octet
9                           # à l'adresse nb1 vers le registre al
10        addb $5, %al      # ajouter la valeur 5 (en décimal)
11                          # à la valeur dans le registre al
12        movb %al, res     # poser la valeur dans le registre al
13                          # dans l'octet à l'adresse res
14
```

Un premier exemple en assembleur

Le code en assembleur : `src/premier.s`



Section 4

Les déclarations de constantes et de variables

Les représentations des valeurs numériques – À vous !

- `13` est une constante entière par défaut en décimal
- `0x13` est une constante en hexadécimal
  et cela fait combien en décimal ?

- `0b1101` est une constante en binaire
  et cela fait combien en décimal ?

- `0137` est une constante en octal
  et cela fait combien en décimal ?

Les représentations des valeurs numériques – À vous !

- `13` est une constante entière par défaut en décimal
- `0x13` est une constante en hexadécimal
  et cela fait combien en décimal ?
  19
- `0b1101` est une constante en binaire
  et cela fait combien en décimal ?

- `0137` est une constante en octal
  et cela fait combien en décimal ?

Les représentations des valeurs numériques – À vous !

- `13` est une constante entière par défaut en décimal
- `0x13` est une constante en hexadécimal
  et cela fait combien en décimal ?
  19
- `0b1101` est une constante en binaire
  et cela fait combien en décimal ?
  13
- `0137` est une constante en octal
  et cela fait combien en décimal ?

Les représentations des valeurs numériques – À vous !

- `13` est une constante entière par défaut en décimal
- `0x13` est une constante en hexadécimal
  et cela fait combien en décimal ?
  19
- `0b1101` est une constante en binaire
  et cela fait combien en décimal ?
  13
- `0137` est une constante en octal
  et cela fait combien en décimal ?
  $1 \times 8^2 + 3 \times 8^1 + 7 \times 8^0 = 64 + 24 + 7 = 95$

Les directives

Dans la section `.data` on peut déclarer (réserver de l'espace) et nommer des valeurs :

```
1         .data
2 nb1:    .byte  1
3 res:    .byte  0
4 bonj:   .ascii "Salut !\n"
5 tab:    .int   0xFFFFAAAA, 350000, 0x10, 2
6         .equ   MAX, 100
```

Les étiquettes sont des adresses mémoires auxquelles on peut trouver la valeur définie, ou de manière consécutive les valeurs définies.

`.equ` associe une valeur à un nom.

Les directives (2)

Les directives permettent de typer les valeurs pour adapter l'espace réservé :

- `.byte` : entier sur un octet

```
1 mini:     .byte   0
2 maxi:     .byte   0xFF
```

- `.word` : entier sur un mot (2 octets)

```
1 maxiw:    .word   65535
```

- `.int` ou `.long` : entier sur un mot double (4 octets)

```
1 unLong:   .long   70000
2 unInt:    .int    700000
```

- `.quad` : entier sur un mot quadruple (8 octets)

```
1 unTresLong:
2           .quad   0xAABC340101000000
```

Les directives (3)

- `.ascii` : chaîne de caractères

```
1 bonjour:  .ascii  "Bonjour !\n"
```

qui est équivalent à :

```
1 bonjour:  .ascii  "Bon", "jour !\n"
```

- `.asciz` : chaîne de caractères terminée par l'octet `0`

```
1 bonjour2: .asciz  "Bonjour !\n"
```

qui est équivalent à :

```
1 bonjour3:  .ascii  "Bonjour !\n\0"
```

Chaque caractère d'une chaîne est stocké sur un octet.

`bonjour` est stocké sur 10 octets, `bonjour2` sur 11 octets.

Pour des listes de valeurs...

- un tableau de quatre entiers :

```
1 tab:    .int    0xFFFFAAAA, 350000, 0x10, 2
```

- un tableau de chaînes de caractères :

```
1 bonjour:  .asciz  "une", "suite", "de", "mots"
```

- pour initialiser un tableau de `nb` éléments à la valeur `val` :
  `.space <nb>,<val>`

```
1 tab:      .space  5, 10
2 tabStar:  .space  5, '*'
```

- pour initialiser un tableau de `nb` éléments à la valeur `val` codée sur `taille` octets :
  `.fill <nb>,<taille>,<val>`

```
1 tab:      .fill   5, 2, 10
2 tabD:     .fill   5, 4, 0xFFFFAA99
```

Section 5

Les registres

Les registres

- Les registres sont des emplacements mémoire internes au processeur.
- Les opérations arithmétiques ne peuvent pas se faire directement sur des emplacements mémoire externes.

Les registres généraux

Il y a 4 registres généraux sur les processeurs récents : a, b, c, d.

Ces registres se décomposent (suivant l'historique technologique) :

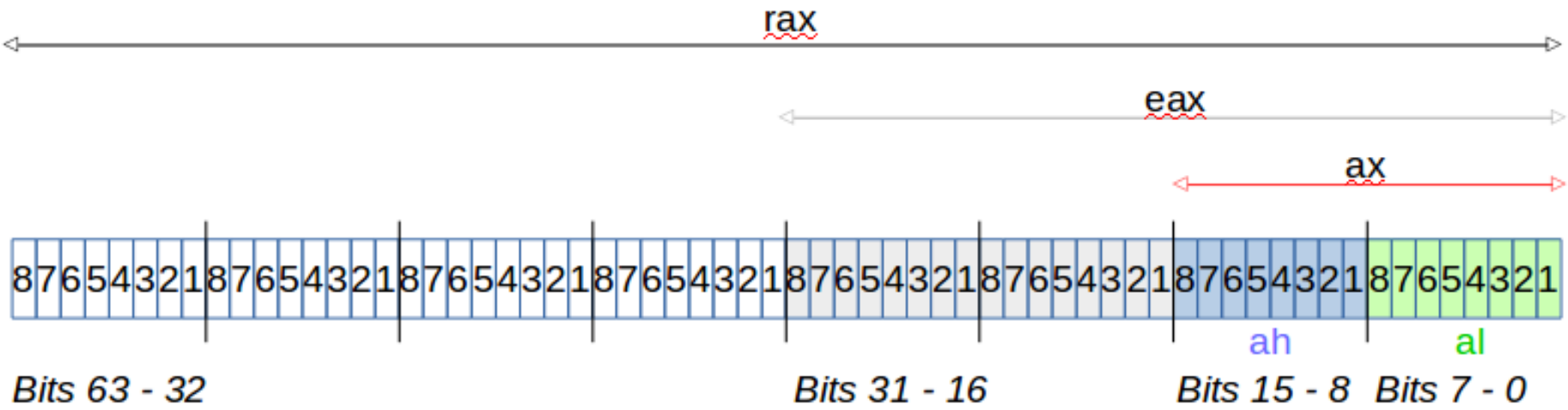
- Accumulator eXtended - deux octets : ax, bx, cx, et dx

  on peut n'adresser que la partie haute ou basse de chaque registre
  - Accumulator High - partie haute, 8 bits de poids fort : ah, bh, ch, et dh
  - Accumulator Low - partie basse, 8 bits de poids faible : al, bl, cl, et dl

- Extended Accumulator eXtended - registres étendus à 32 bits avec 2 octets supplémentaires en partie haute : eax, ebx, ecx, et edx

- Re-extended Accumulator eXtended - dernière extension à 64 bits : rax (et respectivement rbx, rcx, rdx), dont les 32 bits de poids faibles sont constitués par eax etc. . .)

Les registres généraux (2)



Par convention :

- a est utilisé comme accumulateur, il sert pour les paramètres et pour le résultat
- b est utilisé comme base, il sert comme base d'adressage (cf plus tard !)
- c est utilisé comme compteur, pour les boucles
- d est utilisé comme paramètre pour des divisions ou des multiplications

Les registres généraux (3) – À vous !

Supposons que rax contient la valeur `0X8734FDAB3AC2789C`
Que contiennent (réponses en binaire) :

- le registre ah ?
- le registre al ?
- le registre ax ?
- le registre eax ?

Les registres généraux (3) – À vous !

Supposons que rax contient la valeur `0X8734FDAB3AC2789C`
Que contiennent (réponses en binaire) :

- le registre ah ?
  `0x78 = 0b01111000`
- le registre al ?
- le registre ax ?
- le registre eax ?

Les registres généraux (3) – À vous !

Supposons que rax contient la valeur `0X8734FDAB3AC2789C`
Que contiennent (réponses en binaire) :

- le registre ah ?
  `0x78 = 0b01111000`
- le registre al ?
  `0x9C = 0b10011100`
- le registre ax ?
- le registre eax ?

Les registres généraux (3) – À vous !

Supposons que rax contient la valeur `0X8734FDAB3AC2789C`
Que contiennent (réponses en binaire) :

- le registre ah ?
  `0x78 = 0b01111000`
- le registre al ?
  `0x9C = 0b10011100`
- le registre ax ?
  `0x789C = 0b0111100010011100`
- le registre eax ?

Les registres généraux (3) – À vous !

Supposons que rax contient la valeur `0X8734FDAB3AC2789C`
Que contiennent (réponses en binaire) :

- le registre ah ?
  `0x78 = 0b01111000`
- le registre al ?
  `0x9C = 0b10011100`
- le registre ax ?
  `0x789C = 0b0111100010011100`
- le registre eax ?
  `0x3AC2789C = 0b00111010110000100111100010011100`

Section 6

Premiers programmes

Les instructions

```
mnemonique <op1>, <op2>
```

- Les mnémoniques sont les noms associés aux instructions : `MOV`, `ADD`, `SUB`, `MUL`...
- Les instructions ont entre 0 et 2 opérandes ;
- Les opérandes peuvent être soit une étiquette, soit une valeur numérique, soit un registre, soit un accès à la mémoire ;
- En général, il n'y a pas plus d'un accès mémoire pour une instruction ;
- En général, toutes les opérandes doivent avoir la même taille (en nombre d'octets).

Instruction de copie de valeur

On a la section `data` suivante :

```
1 nb1:    .byte 1
2 res:    .byte 0
```

`MOV <source>, <destination>`
copie le contenu de `<source>` dans `<destination>`

```
1 movb nb1, %al
```

déplace la valeur dans l'octet à l'adresse nb1 vers le registre al

- `MOV` transfère des valeurs entre deux registres, entre un registre et la mémoire, ou transfère une valeur dans un registre ou dans la mémoire.
- `MOV` ne transfère pas des valeurs entre deux adresses mémoire.

Quelques instructions arithmétiques standards

`ADD <op1>, <op2>`

additionne le contenu de <op1> et de <op2> et met le résultat dans <op2>

```
1 addb $5, %al
```

additionne 5 et la valeur contenue dans le registre al et stocke le résultat dans al

```
1 addb nb1, %al
```

additionne la valeur contenue dans nb1 et celle dans al et stocke le résultat dans al

`INC <op>`

incrémente de 1 le contenu de l'opérande (adresse mémoire ou registre)

```
1 incb nb1
2 incl %eax
```

Quelques instructions arithmétiques standards (2)

`SUB <op1>, <op2>` `# <op2> = <op2> - <op1>`

soustrait le contenu de <op1> à <op2> et met le résultat dans <op2>

```
1 subb $5, %al
```

soustrait 5 à la valeur contenue dans le registre al et stocke le résultat dans al

```
1 subb %bl, %al
```

soustrait la valeur contenue dans bl à celle dans al et stocke le résultat dans al

`DEC <op>`

décrémente de 1 le contenu de l'opérande (adresse mémoire ou registre)

```
1 decb nb1
2 decl %eax
```

La taille des opérandes

La dernière lettre des mnémoniques indique la taille des opérandes :

- b (comme movb, addb) : byte - les opérandes sont des adresses sur un octet
- w : word - les opérandes sont des adresses de la taille d'un mot $\Rightarrow$ 2 octets
- l ou d : long (word) ou double (word) - 4 octets
- q : quad word, ou quadruple word - 8 octets (64 bits)

Quelques modes d'adressage

Dans les exemples précédents :

- adressage immédiat : quand on utilise une constante (`$5, $0xF3`...), ou pour faire un calcul sur une adresse mémoire (`$nb1`)
- adressage registre : le symbole % distingue un registre (`%eax, %bl`)
- adressage direct : quand on utilise une adresse mémoire (`nb1, res`...)