Programmation en assembleur

Architecture des Ordinateurs
Module M14 Semestre 4

Printemps 2008

Coordinateur du module M14: Younès EL Amrani

Formats d'opérandes en assembleur IA32

Modes d'adressage

Type	Forme	Valeur d'opérandes	Nom
Immédiate	Imm= 12q,10,0xA,Ah,$0A,1010b	Imm (val de Imm)	Immédiate
Registre	REG (REG = EAX , ESP ...)	Contenu de REG (val du registre)	Registre
Mémoire	[Imm]	Contenu de la Mémoire à l'@ Imm	Absolu
Mémoire	[REG] (REG = EAX, ESP...)	Contenu de la Mémoire à l'@ REG	Indirect
Mémoire	[REGb + Imm])	Mémoire (REGb+Imm)	Base + déplacement
Mémoire	[REGb + REGi]	Mémoire (REGb + REGi)	Indexé
Mémoire	[Imm + REGb + REGi]	Mémoire(Imm + REGb + REGi)	Indexé
Mémoire	[REGi * s]	Mémoire (REGi* s)	Indexé, pondéré
Mémoire	[Imm+ REGi * s]	Mémoire (Imm + (REGi* s))	Indexé, pondéré
Mémoire	[REGb+ REGi * s]	Mémoire (REGb + (REGi* s))	Indexé, pondéré
Mémoire	[Imm + REGb + REGi * s]	Mémoire (Imm+REGb+(REGi* s))	Indexé, pondéré

Mémoire

Adresse	Valeur
Ox100	OxFF
0x104	OxAB
0x108	Ox13
0x10C	Ox11

Registres

Registre	Valeur
EAX	Ox100
ECX	Ox1
EDX	Ox3

Que valent les opérandes suivantes ?

EAX	
[Ox104]	
0x108	
[EAX]	
[EAX+4]	
[9+EAX+EDX]	
[260+ECX+EDX]	
[0xFC+ECX*4]	
[EAX+EDX*4]	

Exemples:

Soit la mémoire
Et les registres
Suivants:

Adresse	Valeur
Ox100	OxFF
0x104	OxAB
0x108	Ox13
0x10C	Ox11

Registre	Valeur
EAX	Ox100
ECX	Ox1
EDX	Ox3

Que valent les opérandes suivantes ?

% eax	Ox100	
[Ox104]	OxAB	
0x108	Ox108	
[EAX]	OxFF	
[EAX+4]	OxAB	
[9+EAX+EDX]	Ox11	9+0x100+0x3=0x10C
[260+ECX+EDX]	Ox13	260=16*16+4=0x104
[0xFC+ECX*4]	OxFF	0xFC+0x1*4 = 0x100
[EAX+EDX*4]	0x11	0x100+0x3*4=0x10C

Principaux registres du processeur

1. Le compteur de programme EIP, Ce compteur indique (contient) l'adresse de la prochaine instruction à exécuter. Il ne peut être manipulé directement par programme.
2. Le fichier des registres d'entiers: il contient huit registres. Chaque registre peut contenir une valeur de 32 bits pour une architecture 32 bits comme IA32. Ces registres sont:
    i. EAX , EBX , ECX , EDX (généraux)
    ii. ESI , EDI (Utilisés par les chaînes)
    iii. ESP , EBP (Utilisés par la pile)

Typage niveau assembleur

En assembleur on ne fait pas de distinction entre

(1) Les entiers signés et les entiers non signés

(2) Les adresses (pointeurs) entres elles

(3) Entre les adresses et les entiers

(4) Entre les entiers et les caractères

(5)Entre structures, unions, classes, tableaux, etc

Les entiers et les pointeurs: même chose!

Typage niveau assembleur

QUESTION: Mais alors, qui effectue le typage de programmes en assembleur ??

RÉPONSE: C'est le programmeur qui assure le typage... DANS SA TÊTE !!!

Le typage est effectué par le programmeur

Structures de données coté assembleur

Les tableaux, les structures et les unions sont perçus pareillement au niveau de l'assembleur.

Les structures de données sont perçues comme

« des *collections d'octets contigüs* (côte à côte)»

Tableaux ≡ structures ≡ unions ≡ " collections d'octets contigüs"

Les instructions = opération-code + opérandes: OPCODE *(op)**

- Une instruction à un code d'opération noté opcode + une ou plusieurs opérandes.
- Les opérandes (si nécessaires) contiennent
  - les valeurs qui constituent les données de l'opération à effectuer et / ou
  - l' adresse de la destination du résultat.
- Syntaxe: Les « **( )** » signifient optionnel. Le « ***** » signifie 0 ou plusieurs. Le « **|** » signifie ou.

(label:) OPCODE (operande)* ; commentaire

Les instructions = opération-code + opérandes: OPCODE *(op)**

(label:) OPCODE (operande)* ; commentaire

- **<u>Exemples</u>:**

Ecrit la valeur décimale 1 dans le registre EAX

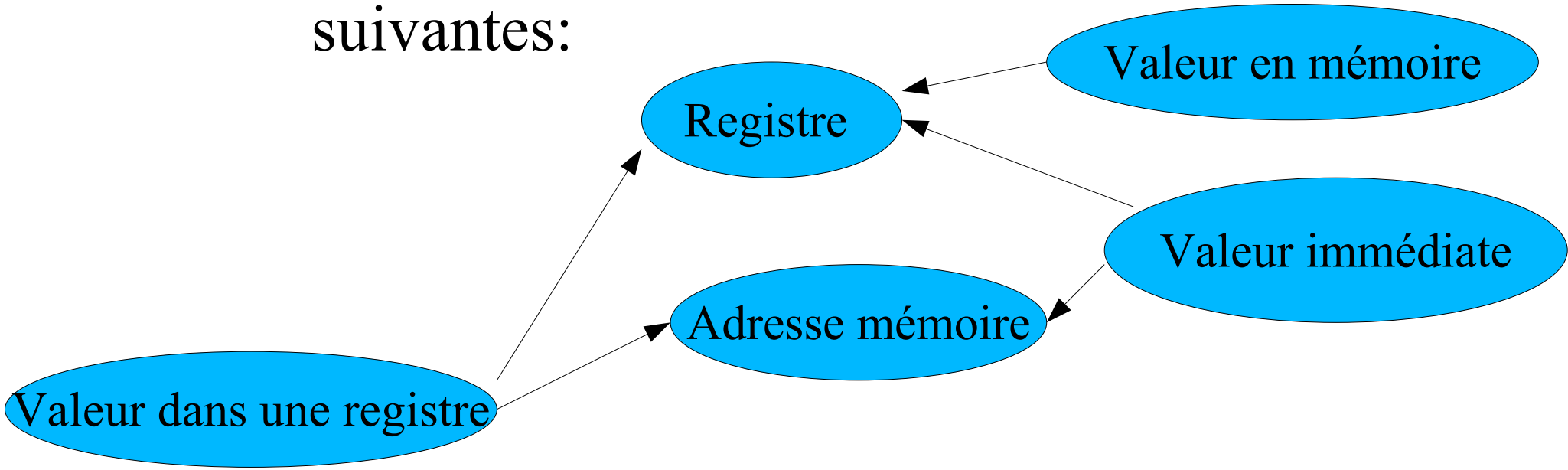
```
mov EAX , 1  ; écrit 1 dans eax
```

Lit la valeur à l'adresse EBP + 8 et l'écrit dans ESP

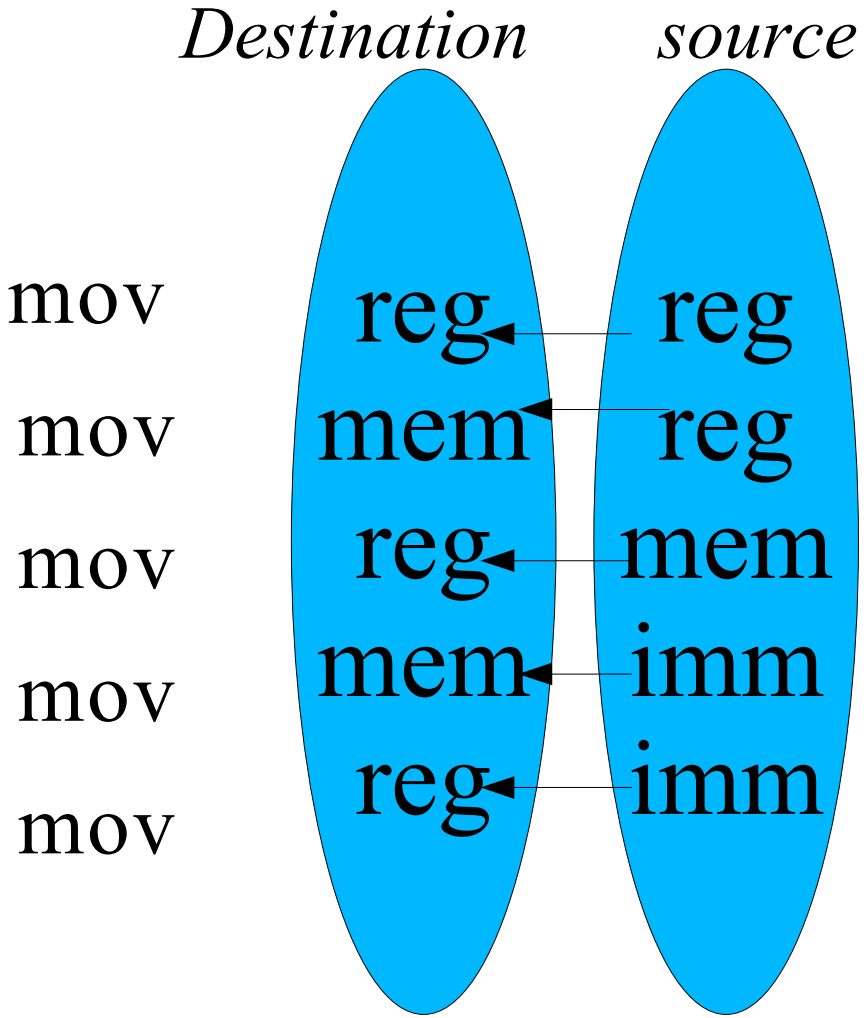
```
; lit mem(ebp+8) et l'écrit dans esp
mov ESP , [ EBP + 8 ]
```

Instructions de transfert de données

- Parmi les instruction les plus utilisées figurent celle qui déplacent les données (on dit aussi qui transfèrent les données). C'est l'instruction MOV
- Les transferts peuvent se faire dans les direction suivantes:



L'instruction move: jamais de mémoire à mémoire !



Instructions de transfert de donnée

- Une opération *MOV* ne peut avoir deux opérandes qui sont toutes les deux des adresses mémoires. Autrement dit, il n'existe pas en IA32 de transfert direct de la mémoire vers la mémoire: il faut passer par un registre.

Un mot sur l'assembleur de GCC: GAS

- Le format de l'assembleur *GNU* (noté GAS) est très différent du format standard utilisée dans la documentation d'*Intel* ainsi que d'autres compilateurs (y compris *Microsoft*) une différence majeure est dans l'inversion de l'ordre des opérandes source et destination. Ainsi que le suffixe % dont sont affublés les registres.
- Dans GAS les registres EAX, EBX, etc se notent:

%EAX

%EBX etc

GNU Assembleur Versus Assembleur INTE

- GAS documente ses propres différences avec les notations standard d'Intel.
- La commande ***info as*** dans l'environnement Linux permet d'obtenir la documentation sur GAS dans l'environnement Linux.
- Une sous-section est réservée à la comparaison de GAS avec la notation standard d'Intel.
- Dans la pratique: l'assembleur GAS est utilisé uniquement pour le code généré de gcc.

L'assembleur sous LINUX

- Observation 1:

  Il est interessant de connaître GAS pour s'inspir

  du code produit pour les programmes C et C++

- Observation 2:

  Cependant, il est plus intéressant de connaître le
  instructions de la documentation d'Intel qui
  s'imposent comme un standard.

Comparaison entre GAS et INTEL

- Différence 1

en GAS nous avons

opération source, destination

en IA32 on a

opération destination , source

- Différence 2

Pour chaque opération de GAS un caractère en suffixe indique la taille de l'opérande:

- b pour byte (1 octet)
- w pour word (16 bits) et
- L pour long words (32 bits)

Format des instructions dans un assembleur

(label:) opcode (opérandes) ; commentaire*

- Généralement on distingue deux types d'instructions:

  1- les instructions directives donnent des directives qui ne sont pas des instructions exécutable. Ex: ***extern _printf***

  2- les instructions exécutables. Ex: PI: DD 3.14 ; le réel pi

- Exemples d'instructions *directive:*

  extern _printf ; printf sera importée

  global _main ; le _main sera exporté

Allocation de données initialisées

Nombre d'octets	Mnémonique	Description	Attribut
1	D B	Définit un byte (un octet)	Byte
2	D W	Définit un mot (un " Word " en anglais)	Word
4	DD	définit un double mot (Double Word)	Double word
8	DQ	définit un mot quadruple (quadruple word)	Quad word
10	DT	Définit dix octets (dix bytes)	Ten bytes

Données initialisées en NASM

```
;;;Ce programme affiche "Hello, World"
SEGMENT .data ; données initialisées
textHello:          DB "Hello, World !" , 0
integerFormat:      DB "%d" , 0
stringFormat:       DB "%s" , 0
characterFormat:    DB "%c" , 0
charVal1:     DB  'C'
intVal1:      DD 15
intVal2:      DD 20
```

Allocation de données non initialisées

Nombre d'octets	Mnémonique	Description	Nom
1 * n	RESB n	Réserve n octet(s)	Byte
2 * n	RESW n	Réserve n mots (1 mot = 2 octets)	Word
4 * n	RESD n	Réserve n double mot(s) (1 Double mot = 4 octets)	Double word
8 * n	RESQ n	Réserve un quadruple mot (quadruple word)	Quad word
10 * n	REST n	Réserve dix octets (dix bytes)	Ten bytes

Données non-initialisées en NASM

```
;;;segment de données non initialisées
SEGMENT .bss
textHello:      RESB 15
textSalam:      RESB 5
tableauInt1:    RESD 10
tableauInt2:    RESD 15
charVal1:       RESB  1
charVal2:       RESB  1
```

Programme proprement dit

.text ;marque le début du segment de code

global _main ; *symboles exportés vers l'extérieur*

; *fonctions importées*

extern _printf , _scanf , _f1 , _f2

_main:

;;;*la directive **proc** annonce le début de la*
;;;procédure main. Cette procédure est
;;;fondamentale: elle marque le début de l'entrée
;;;du code dans le protocol C

Programme Hello World utilise la librairie C

```
_main: enter 0, 0
        pusha   ; EMPILER TOUS LES REGISTRES
        pushf   ; EMPILER LE REGISTRE D'ETAT
        mov eax , hello ; ADRESSE DU TEXTE HELLO
        push EAX        ; EMPILER ADRESSE HELLO
        push integerFormat ; EMPILER FORMAT
        call   printf           ; APPEL DE FONCTION
        pop    ecx      ; DEPILER UNE FOIS <=> ADD ESP, 4
        pop    ecx      ; dépiler une seconde fois  <=> ADD ESP, 4
```

Fin Du Programme Hello World

```
mov eax , 0   ;;; Signifie la fin de programme pour l'OS
leave         ;;; libère la zone utilisée sur la pile
ret           ;;; restaure le registre EIP
```

Les Macros en NASM

%macro nomMacro n ; n = nombre d'arguments

OPCODE1 <operandes>

...

OPCODEp <operandes>

%endmacro

Dans le corps de la macro, %1 dénote le premier argument en ligne arg1 ,..., %n référence argn

APPEL: nomMacro arg1 , ... , argn

RISC versus CISC

- En fait, avant les années 80, la notion de RISC était inexistante. La tendance était de rajouter autant d'instructions que possible au processeur.
- En fait, c'est dans les années 80, que des chercheurs d'**IBM** sous la direction de **John Cocke** se sont convaincus qu'un ensemble réduit d'instructions " rapides" valait mieux qu'un grand ensemble d'instructions parfois plus lentes.

RISC versus CISC

- Les processus RISC seraient moins chers, moins complexes et ne comporteraient que des instructions très rapides toutes codées sur un même nombre d'octets.
- Les professeurs David Patterson (ami Berkeley et John Hennessy (Stanford) sont ceux qui apposèrent les noms de CISC et RISC aux deux philosophies.

RISC	CISC
Peu d'instructions (~100)	Beaucoup d'instructions (~ 1000)
Instructions rapides: 1 cycle = 1 instruction	Des instructions parfois lentes (> 1 cycle)
Instructions toute codées sur 4 octets	Instructions codées sur 1 à 15 octets
Une base + 1 déplacement pour l'adressage	Format complexe car plusieurs formats utilisés pour l'adressage mémoire
Opérations arithmétiques et logiques sur les registres uniquement	Opération arithmétiques et logiques à la fois sur des registres et de la mémoire
Contraintes d'implémentation: séquence d'instructions interdites	Implémentation transparente
Seuls des tests, dont le résultat va dans des registres, sont utilisés lors des branchements conditionnels	Des drapeaux sont positionnés et utilisés pour lors des branchements conditionels
Utilisation uniquement des registres pour les arguments des fonctions ainsi que pour l'adresse de retour	Utilisation intensive de la pile pour les arguments et pour l'adresse de retour

RISC versus CISC

- Dans les années 80 la communauté scientifique a longuement débattu de l'avantage de l'une et l'autre philosophie.
- Dix ans plus tard, il est apparu que l'une et l'autre avait des avantages. Ainsi les processus RISC ont tendance à devenir de + en + CISC et vice-versa.
- La technologie CISC domine le marché des ordinateurs de bureau et des ordinateurs portables. La technologie RISC domine le marché des microprocesseurs embarqués.

Documentation

- Sur le site d'intel **http://download.intel.com/**

On trouve les documents suivants:

- Volume I volume 1 Basic Architecture, 1999 donne un panorama de l'architecture du point de vue d'un programmeur en assembleur.
- *download.intel.com/design/pentiumII/manuals/24319002.PDF*

- Volume II Intel Architecture Software Developer's Manual volume 2

Ensemble des Instructions Volume

Instruction Set Architecture (ISA), contenu dans le

Volume 2. Donne une description détaillée des différentes instructions disponibles sur le microprocesseur.

http: //download.intel.com/design/pentiumII/

manuals/24319102.PDF

Manuel du développeur
Volume III

Intel Architecture Software Developer's Manual

download.intel.com/design/pentiumII/manuals

/24319202.PDF

Module M14. Resp. Younès EL AMRANI.