

## System

- Numero: 3
- Prof: Laskar Gabriel
- Date: 18 Octobre 2017

## Cours d'assembleur

Makefile: -save-temps : sauvegarde tous les fichiers intermediaires de la compilation Dans le .s: code assembleur traduisible par la machine (à vérifier) Symbole local : static dans la table de symbole locale (à vérifier) Mnémonique : instructions (movq, etc...), suivi d'opérandes call +name\_func : appel à la fonction Stockage des variables locales: pile, registres (%rax): déréférence %rax. Exemple: movq -8(%rbp) -> déréférence %rbp - 8 call \*%rax = appelle fonction à l'adresse %rax (pointeur sur fonction)

## Méta-données

Méta-données: utiles pour le linkeur, donnent diverses infos Exemples :

- .file +name\_file : indique le fichier source, informatif
- .glob(a)l +nom\_du\_symbole : met le symbole dans la table <=> extern
- .type +symbole, @function : dans la table de symboles, déclare les fonctions
- .text : section code
- .size main, .-main : taille du symbole main (.-... > adresses courantes)
- .string "toto": directive pour mettre des strings en mémoire
- .section +name: passe dans la section voulue, par exemple .rodata pour les variables constantes
- **ATTENTION** : ".name:" == label

## Conventions de nommages

Paramètres: **%rdi, %rsi, %rdx**, %rcx, %r8, %r9, dans cet ordre S'il n'y a pas assez de registres : push les paramètres dans la pile %xmm(0-6) pour les flottants Retour: %rax

À l'origine, %rdi et %rsi sont pour les strings (d -> destination, s -> source)

## Pile

Dans la pile: variables locales, paramètres, valeur de retour, adresse de retour **Tête de pile** : %rsp

```

1  +-----+ 0xff
2  | Stack |
3  +-----+
4
5
6  +-----+
7  | Heap  |
8  +-----+ 0x0

```

Ajout en pile: décrémente %rsp, inverse pour retirer en pile Manipuler la pile plus simplement : 1. push : ajout valeur en pile et décrémente 2. pop : retire valeur en pile et incrémente call: met l'adresse de retour en pile ret: rappelle l'adresse dans la pile **ATTENTION** On veut à ret le même état qu'à l'arrivée Pour cela, on utilise l'adresse de la base de pile %rbp pour créer une stack frame

```
pushq %rbp movq %rsp prologue
```

```
popq %rbp epilogue
```

### Exemple pour my\_strlen avec 2 size\_t i et len

```

1  +-----+ %rbp
2  | @ret  |
3  +-----+
4  +-----+
5  |  orbp  |
6  +-----+ %rsp
7  +-----+
8  |        |
9  +-----+
10 +-----+
11 |        |
12 +-----+
13 +-----+
14 |        |
15 +-----+

```

```
pushq %rbp movq %rsp prologue
```

```

1  +-----+ %rbp
2  | @ret  |
3  +-----+
4  +-----+

```

5		orbp	
6	+	-----	+
7	+	-----	+
8			
9	+	-----	+
10	+	-----	+
11			
12	+	-----	+
13	+	-----	+
14			
15	+	-----	+

%rsp

subq \$24, %rsp

1	+	-----	+	%rbp
2		@ret		
3	+	-----	+	
4	+	-----	+	
5		orbp		
6	+	-----	+	
7	+	-----	+	
8				
9	+	-----	+	
10	+	-----	+	
11				
12	+	-----	+	
13	+	-----	+	
14		str		
15	+	-----	+	%rsp

movq %rdi, -24(%rbp)

1	+	-----	+	%rbp
2		@ret		
3	+	-----	+	
4	+	-----	+	
5		orbp		
6	+	-----	+	
7	+	-----	+	
8				
9	+	-----	+	
10	+	-----	+	
11		len = 0		

```

12 +-----+
13 +-----+
14 |  str   |
15 +-----+ %rsp

```

movq \$0, -16(%rsp)

```

 1 +-----+ %rbp
 2 |  @ret   |
 3 +-----+
 4 +-----+
 5 |  orbp   |
 6 +-----+
 7 +-----+
 8 |  i = 0   |
 9 +-----+
10 +-----+
11 |  len = 0  |
12 +-----+
13 +-----+
14 |  str     |
15 +-----+ %rsp

```

movq %rax, -8(%rsp) *%rax provient d'un appel à foo*

```

1 int foo()
2 {
3     return 0;
4 }

```

```

 1 +-----+ %rbp
 2 |  @ret   |
 3 +-----+ %rsp
 4 +-----+
 5 |         |
 6 +-----+
 7 +-----+
 8 |         |
 9 +-----+
10 +-----+
11 |         |
12 +-----+
13 +-----+

```

14		
15	+-----+	

leave <=> move %rbp, %rsp pop %rbp *épilogue*

### Infos diverses

x86\_32 = 6 registres => paramètres mis de base dans la pile x86\_64 = paramètres mis en registre en premier movzbl == mov + z (déréférencement) + b (de byte) + l (à long) Jeu d'instructions plus récent contient moins d'instructions car optimisation automatiques par la machine de l'assembleur.