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

```
+-------+ 0xff
| Stack |
+-------+


+-------+
| Heap  |
+-------+ 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

```
+----------+ %rbp
|  @ret    |
+----------+
+----------+
|  orbp    |
+----------+ %rsp
+----------+
|          |
+----------+
+----------+
|          |
+----------+
+----------+
|          |
+----------+
```

pushq %rbp movq %rsp *prologue*

```
+----------+ %rbp
|  @ret    |
+----------+
+----------+
```

```
|  orbp    |
+----------+
+----------+
|          |
+----------+
+----------+
|          |
+----------+
+----------+
|          |
+----------+ %rsp
```

subq $24, %rsp

```
+----------+ %rbp
|  @ret    |
+----------+
+----------+
|  orbp    |
+----------+
+----------+
|          |
+----------+
+----------+
|          |
+----------+
+----------+
|  str     |
+----------+ %rsp
```

movq %rdi, -24(%rbp)

```
+----------+ %rbp
|  @ret    |
+----------+
+----------+
|  orbp    |
+----------+
+----------+
|          |
+----------+
+----------+
| len = 0  |
```

```
+----------+
+----------+
|  str     |
+----------+ %rsp
```

movq $0, -16(%rsp)

```
+----------+ %rbp
|  @ret    |
+----------+
+----------+
|  orbp    |
+----------+
+----------+
|  i = 0   |
+----------+
+----------+
| len = 0  |
+----------+
+----------+
|  str     |
+----------+ %rsp
```

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

```
int foo()
{
  return 0;
}
```

```
+----------+ %rbp
|  @ret    |
+----------+ %rsp
+----------+
|          |
+----------+
+----------+
|          |
+----------+
+----------+
|          |
+----------+
+----------+
```

```
|           |
+-----------+
```

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

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.