CIRC2 : Assembleur, Numération, et Circuits

Licence informatique – semestre 4

Section 1

Utilisation de la pile

Utilisation de la pile

- La pile est un tableau de cases mémoires contigües
- L'instruction `pushq operande` permet d'empiler
- `popq operande` permet de dépiler.
- Le processeur dispose d'un registre spécial `rsp` pour Re-extended Stack Pointer, pointeur de pile étendue.
- `%rsp` pointe toujours sur le sommet de la pile (le dernier élément empilé)

- Les opérations se font toujours par 64 bits sur les processeurs 64 bits.
- Sur un système 32 bits, la pile est `esp` et les instructions sont `pushl` et `popl`.
- Sur un système 16 bits, c'est `sp`, `pushw` et `popw`.

Exemples utilisation de la pile

Permuter le contenu de registres `rax` et `rbx`

```
1 pushq   %rax
2 movq    %rbx, %rax
3 popq    %rbx
```

On peut accéder à une zone particulière de la pile :

```
1 pushq   $10             # valeur qui va etre ajoutee a rax
2 pushq   %rax
3 movq    %rbx, %rax
4 addq    8(%rsp), %rax # 8(%rsp) est l'emplacement de $10
5 popq    %rbx
```

Afficher des étoiles

```
*****
 ***
  *
```

Pour `lig` lignes à afficher :

- `rsi` contient le numéro de ligne courant (de `lig` à `1`) : initialisé à `lig`
- `rdi` contient le nombre d'étoiles : initialisé à `2*lig - 1`
- pour chaque ligne faire :
  - empiler `rsi` (qui contient `i` le numéro de ligne courant)
  - empiler `rdi`
  - afficher `lig - i` espaces en utilisant le compteur `rsi`
  - afficher `2*i - 1` étoiles en utilisant le compteur `rdi`
  - afficher un retour à la ligne
  - dépiler `rdi` et le diminuer de `2`
  - dépiler `rsi` et le diminuer de `1`

Afficher des étoiles – Solution (1)

```
1           .data
2   star:   .byte   '*'
3   nl:     .byte   '\n'
4   esp:    .byte   ' '
5   lig:    .quad   7
6
7           .text
8           .globl  main
9   main:   movq    lig, %rsi
10                  # %rsi contient le num. ligne courant
11          movq    %rsi, %rdi
12                  # nbre d'etoiles : 2*%rsi -1
13          shlq    %rdi     # x2
14          decq    %rdi
15                  # %rdi contient le nbre d'etoiles
```

Afficher des étoiles – Solution (2)

```
1  ligne:  pushq   %rsi
2                  # sur la pile : le num. ligne courant
3          pushq   %rdi     # et le nombre d'etoiles
4          # afficher une ligne
5          # afficher des espaces
6          movq    lig, %rbx # nbre d'espaces : lig-num.ligne
7          subq    %rsi, %rbx
8          movq    %rbx, %rsi
9                  # %rsi contient maintenant le nbre d'espa
10         cmpq    $0, %rsi
11         je      etoile
12 space:  ...                # ecrire ' '
13         decq    %rsi
14         jnz     space
15         # afficher des etoiles
16 etoile: ...                # ecrire '*'
17         decq    %rdi
18         jnz     etoile
```

Afficher des étoiles – Solution (3)

```
1           # aller a la ligne
2 nvligne:
3           ...                 # ecrire '\n'
4           popq    %rdi
5           subq    $2, %rdi
6           popq    %rsi
7           decq    %rsi
8           jnz     ligne
9
10 fin:
11          movl    $1,%eax
12          movl    $0,%ebx
13          int     $0x80
```

Section 2

Les procédures

L'instruction `call`

- `call nom_proc` est l'instruction pour appeler la procédure à l'adresse symbolique (étiquette d'instructions) `nom_proc`
- Les paramètres de la procédure sont généralement sur la pile.
- À l'appel de l'instruction :
  - `call`, l'adresse au sommet du registre des adresses d'instructions `rip` est mis sur la pile
  - l'adresse dans le registre `rip` est remplacée par l'adresse associée à `nom_proc`.

L'instruction `ret`

- `ret` est l'instruction pour quitter la procédure et revenir à l'instruction suivant l'appel de la procédure.
- Lorsque `ret` est appelé, l'adresse au sommet de la pile est dépilée et posée sur `rip`.
- `ret` peut-être appelé avec une opérande `n` :
  - `ret` dépile l'adresse de l'instruction suivante
  - puis dépile `n` autres éléments de la pile.

- Le registre `rbp` pour *Re-extended Base Pointer* sert souvent pour mémoriser les différentes adresses importantes sur la pile (lorsqu'on veut accéder directement à un élément de la pile par exemple).
- Il pointe généralement sur l'adresse de retour de la procédure : en-deça on trouvera les paramètres, au-delà les variables locales...

Exemples (1)

On peut maintenant passer les instructions pour écrire un caractère ou une chaîne de caractères dans une procédure :

```
1 space:  pushq   $esp
2         call    ecr_car          # ecrire ' '
3         ...
4 etoile: pushq   $star
5         call    ecr_car          # ecrire '*'
6         ...
7         pushq   $nl
8         call    ecr_car          # ecrire '\n'
```

Exemples (2)

Et la procédure `ecr_car` :

```
1  ecr_car:
2    movq   %rsp, %rbp          # %rbp contient l'adresse de
3                               # retour et l'adresse au-dessus
4                               # duquel il y a les parametres
5    movl   $4, %eax
6    movl   $1, %ebx
7    movq   $1, %rdx            # on ecrit 1 caractere
8    movq   8(%rbp), %rcx       # on recupere l'adresse du car
9                               # a ecrire
10   int    $0x80               # appel de l'interruption
11   retq   $8                  # on depile l'argument
12                              # de la fct avant de quitter
```

Effets de bord...

Que pensez-vous de ce code ?

```
1         .data
2 plus:   .byte   '+'
3 deux:   .byte   '2'
4 un:     .byte   '1'
5
6 movq    $deux, %rax
7 movq    deux, %rcx
8 subq    $'0', %rcx
9 pushq   %rax
10 call   ecr_car
11 pushq  $plus
12 call   ecr_car
13 pushq  $un
14 call   ecr_car
15 incq   %rcx
```

La pile pour savegarder les registres

- les procédures ne doivent pas modifier l'état des registres
- en début de procédure, il faut empiler les registres pour les sauvegarder
- en fin de procédure, il faut dépiler les registres pour remettre dans l'état initial

- `rbp` pointe sur l'adresse du retour de la fonction
- les registres sont sur la pile entre `rbp` et `rsp`
- et pour permettre les appels de fonctions dans des fonctions, il faut d'abord sauvegarder `rbp`

Afficher des étoiles (4)

```
1  ecr_car:
2    pushq    %rbp
3    movq     %rsp, %rbp      # %rbp contient l'adresse de
4                             # retour et l'adresse au-dessus
5                             # duquel il y a les parametres
6    pushq    %rax            # sauvegarde des registres
7    pushq    %rbx
8    pushq    %rcx
9    pushq    %rdx
10
11   ...      ...             # ecriture d'un caractere
12   movq     8
13
14 (%rbp), %rcx
15                            # on recupere l'adresse du car
16                            # a ecrire
17
18   popq     %rdx            # remise en etat des registres
19   popq     %rcx
```

Exemples (3)

```
def f(a,b) :
  return a + 2*b
```

En assembleur (cas d'un retour sur la pile) :

```
1   f:   pushq    %rbp
2        movq     %rsp, %rbp
3        pushq    %rax
4        pushq    %rbx
5        movq     8(%rbp), %rax    # param b dans rax
6        movq     16(%rbp), %rbx   # param a dans rbx
7        shlq     %rax
8        addq     %rbx, %rax
9        movq     %rax, 16(%rbp)
10       popq     %rbx
11       popq     %rax
12       popq     %rbp
13       retq     $8
```

Exemples (3)

```
1 main:
2       pushq   $5
3       pushq   $23
4       call    f
5       popq    %rax                # on recupere le resultat
6       ...
```

ou bien la fonction retourne le résultat dans un registre défini.

Variables locales

Les variables locales sont stockées sur la pile, dans le contexte local, entre `rsp` et `rbp`.

```
def f(a) :
  if a%2 == 0 :
     c = a/2
  else :
     c = 3*a + 1
  return c
```

- le paramètre sera fourni dans le registre `rax`
- et on retournera le résultat dans le registre `rax`.

Variables locales

```
1  f:   pushq %rbp
2       movq  %rsp, %rbp
3       pushq %rbx
4       pushq %rdx
5       subq  $8, %rsp     # on reserve une case memoire sur l
6       movq  $0, %rdx
7       movq  %rax, -8(%rsp)
8       movq  $2, %rbx
9       divq  %rbx
10      cmpq  $0, %rdx
11      je    ffin
12      movq  -8(%rsp), %rax
13      movq  $3, %rbx
14      movq  $0, %rdx
15      mulq  %rbx
16      incq  %rax
17      movq  %rax, -8(%rsp)
18 ffin:
19      popq  %rax
```