Rappels d'assembleur

V. Nicomette, E. Alata

INSA Toulouse

Introduction

Rappels sur les appels de fonction

Examples

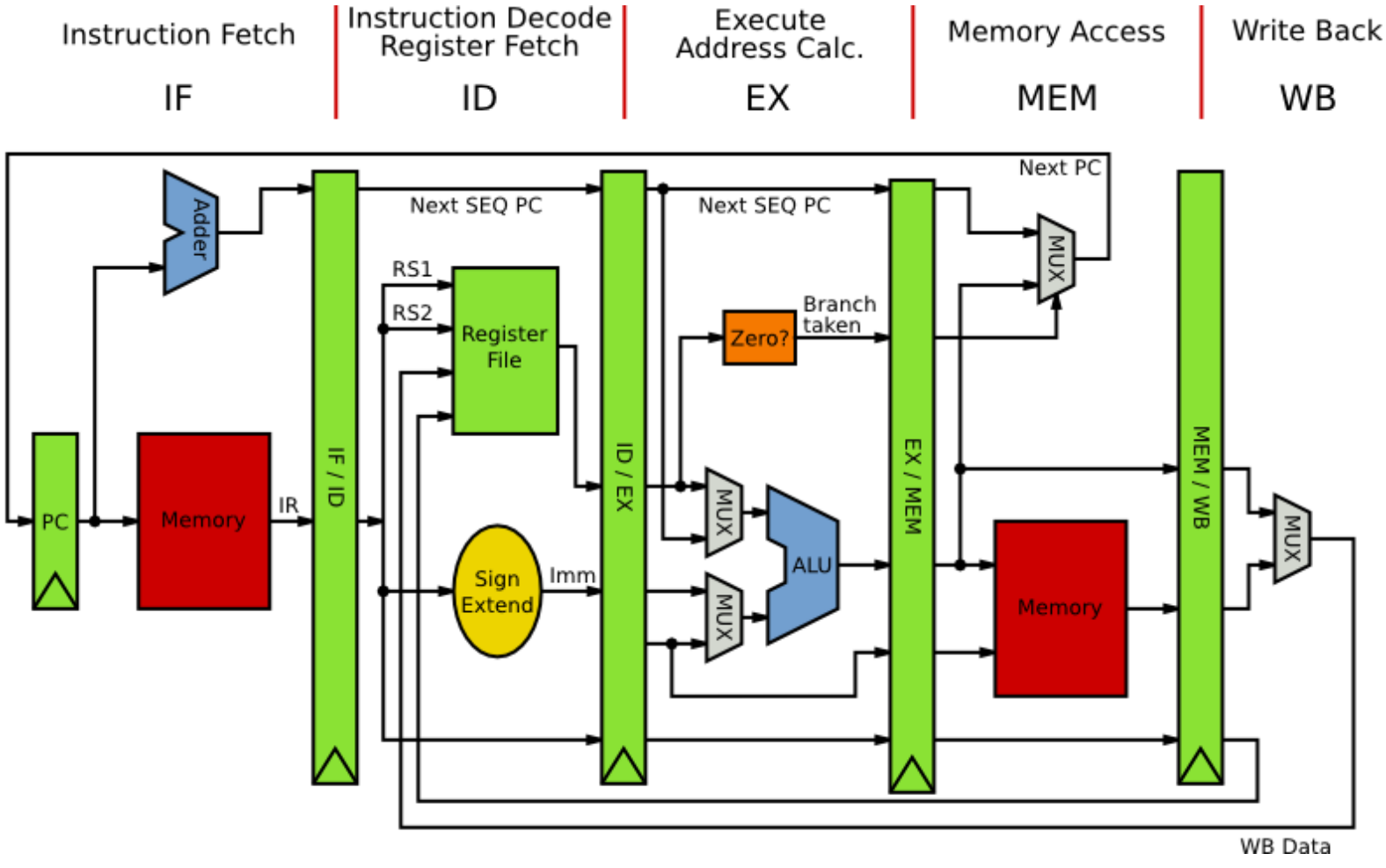
Introduction

Rappels sur les appels de fonction

Examples

Les processeurs

- Plusieurs types : amd, intel, arm, atmel, ...
- Architectures plus ou moins complexes (pipeline, memoire, registres, ...)
- Exemple d'une architecture MIPS :



Les outils à connaître

- `gcc` – pour la compilation
- `ddd` – pour le *debuggage*
- `gdb` – pour le *debuggage*
- `objdump` – pour le désassemblage

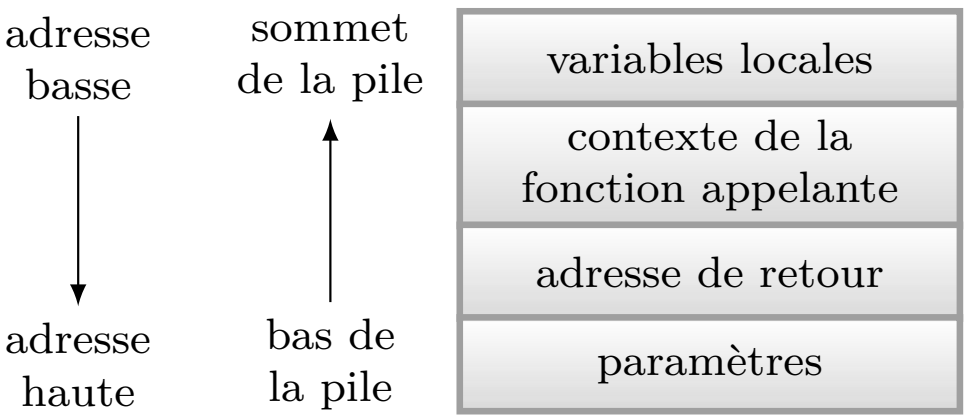
Introduction

Rappels sur les appels de fonction

Examples

Appel de fonction

- Lors de l'appel d'une fonction en langage C, on empile dans l'ordre :
  - Les paramètres de la fonction invoquée
  - L'adresse de retour de la fonction appelante
    adresse de l'instruction qui suit l'invocation
  - Une sauvegarde du contexte d'exécution de la fonction appelante
    dépend de l'architecture matérielle (état de la pile, etc.)
  - Les variables locales




```
void f(int a, char* str) {
  char ch[8];
  int var;
}
void main(int argc, char** argv) {
  f(4, argv[1]);
}
```

Appel de fonction

- Lors de l'appel d'une fonction en langage C, on empile dans l'ordre :
  - Les paramètres de la fonction invoquée
  - L'adresse de retour de la fonction appelante
    adresse de l'instruction qui suit l'invocation
  - Une sauvegarde du contexte d'exécution de la fonction appelante
    dépend de l'architecture matérielle (état de la pile, etc.)
  - Les variables locales

- Lors du retour, on dépile dans l'ordre :
  - Les variables locales
  - La sauvegarde du contexte d'exécution de la fonction appelante
    le contexte est restauré
  - L'adresse de retour de la fonction appelante
    retour au niveau de l'instruction qui suit l'invocation
  - Les paramètres de la fonction invoquée

Appel de fonction sur x86 1/3

- Le registre `esp` est le sommet de la pile
  - Il évolue avec les `push`/`pop`
  - `push` : un élément est ajouté en sommet de pile et `esp` est décrémenté
  - `pop` : un élément est retiré du sommet de pile et `esp` est incrémenté

  $\Rightarrow$ la taille de la pile évolue au cours de l'exécution

  NB `push` *et* `pop` *peuvent être remplacés par des* `mov`, `add` *et* `sub`
- Le registre `ebp` est le pointeur de base de pile
  - Difficile de localiser les paramètres avec `esp` (taille de pile variable)
  - Utilisation d'un registre qui pointe sur le début d'une zone de la pile réservée à la fonction en cours d'exécution
  - Lors de l'appel d'une fonction, ce registre doit donc être sauvegardé pour permettre à la fonction appelée de disposer de sa propre zone de pile
- La pile *grandit* vers le bas, en opposition au tas qui *grandit* vers le haut

Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

```
f :          08048cb0  push  %ebp
             08048cb1  mov   %esp,%ebp
             08048cb3  sub   $0x10,%esp
             08048cb6  leave
             08048cb7  ret
main : →     08048cb8  push  %ebp
             08048cb9  mov   %esp,%ebp
             08048cbb  sub   $0x8,%esp
             08048cbe  mov   Oxc(%ebp),%eax
             08048cc1  add   $0x4,%eax
             08048cc4  mov   (%eax),%eax
             08048cc6  mov   %eax,Ox4(%esp)
             08048cca  movl  $0x4,(%esp)
             08048cd1  call  8048cb0
             08048cd6  leave
             08048cd7  ret
```

Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

```
f :       08048cb0  push  %ebp
          08048cb1  mov   %esp,%ebp
          08048cb3  sub   $0x10,%esp
          08048cb6  leave
          08048cb7  ret
main :    08048cb8  push  %ebp
        → 08048cb9  mov   %esp,%ebp
          08048cbb  sub   $0x8,%esp
          08048cbe  mov   Oxc(%ebp),%eax
          08048cc1  add   $0x4,%eax
          08048cc4  mov   (%eax),%eax
          08048cc6  mov   %eax,Ox4(%esp)
          08048cca  movl  $0x4,(%esp)
          08048cd1  call  8048cb0
          08048cd6  leave
          08048cd7  ret
```

Appel de fonction sur x86 2/3

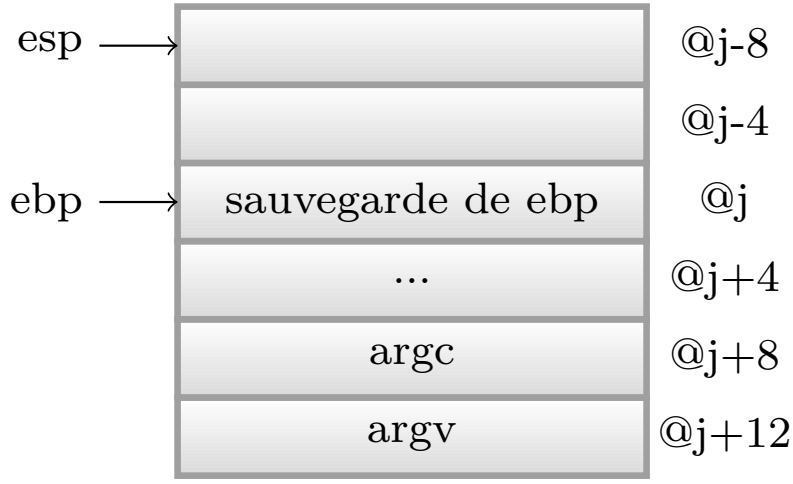
Dćompilation du binaire : `objdump -d program` 2

```
f :      08048cb0  push  %ebp
         08048cb1  mov   %esp,%ebp
         08048cb3  sub   $0x10,%esp
         08048cb6  leave
         08048cb7  ret
main :   08048cb8  push  %ebp
         08048cb9  mov   %esp,%ebp
      →  08048cbb  sub   $0x8,%esp
         08048cbe  mov   Oxc(%ebp),%eax
         08048cc1  add   $0x4,%eax
         08048cc4  mov   (%eax),%eax
         08048cc6  mov   %eax,Ox4(%esp)
         08048cca  movl  $0x4,(%esp)
         08048cd1  call  8048cb0
         08048cd6  leave
         08048cd7  ret
```

Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

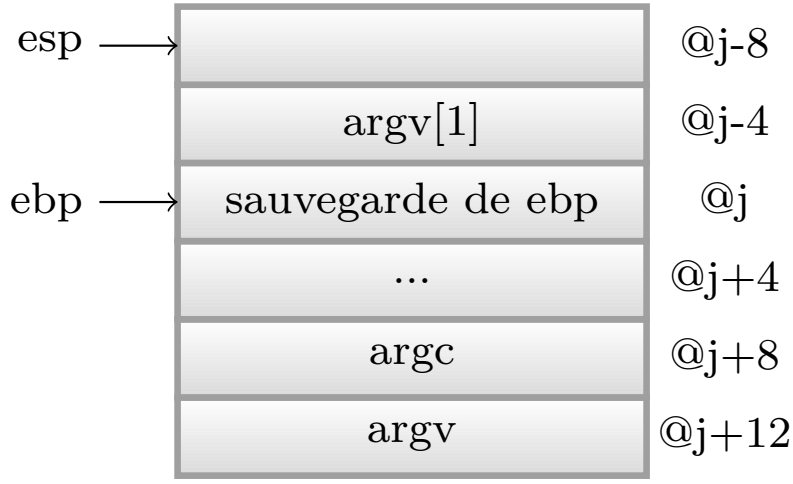
```
f :      08048cb0  push  %ebp
         08048cb1  mov   %esp,%ebp
         08048cb3  sub   $0x10,%esp
         08048cb6  leave
         08048cb7  ret
main :   08048cb8  push  %ebp
         08048cb9  mov   %esp,%ebp
         08048cbb  sub   $0x8,%esp
      →  08048cbe  mov   Oxc(%ebp),%eax
      →  08048cc1  add   $0x4,%eax
      →  08048cc4  mov   (%eax),%eax
      →  08048cc6  mov   %eax,Ox4(%esp)
         08048cca  movl  $0x4,(%esp)
         08048cd1  call  8048cb0
         08048cd6  leave
         08048cd7  ret
```



Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

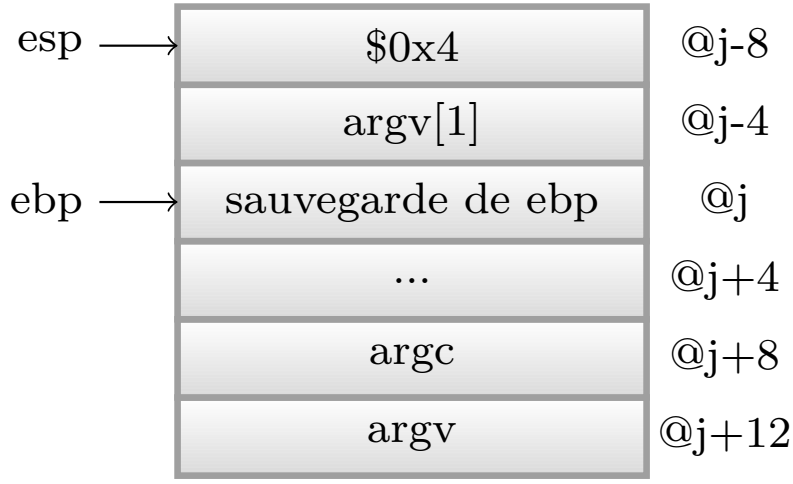
```
f :        08048cb0  push  %ebp
           08048cb1  mov   %esp,%ebp
           08048cb3  sub   $0x10,%esp
           08048cb6  leave
           08048cb7  ret
main :     08048cb8  push  %ebp
           08048cb9  mov   %esp,%ebp
           08048cbb  sub   $0x8,%esp
           08048cbe  mov   Oxc(%ebp),%eax
           08048cc1  add   $0x4,%eax
           08048cc4  mov   (%eax),%eax
           08048cc6  mov   %eax,Ox4(%esp)
      →    08048cca  movl  $0x4,(%esp)
           08048cd1  call  8048cb0
           08048cd6  leave
           08048cd7  ret
```



Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

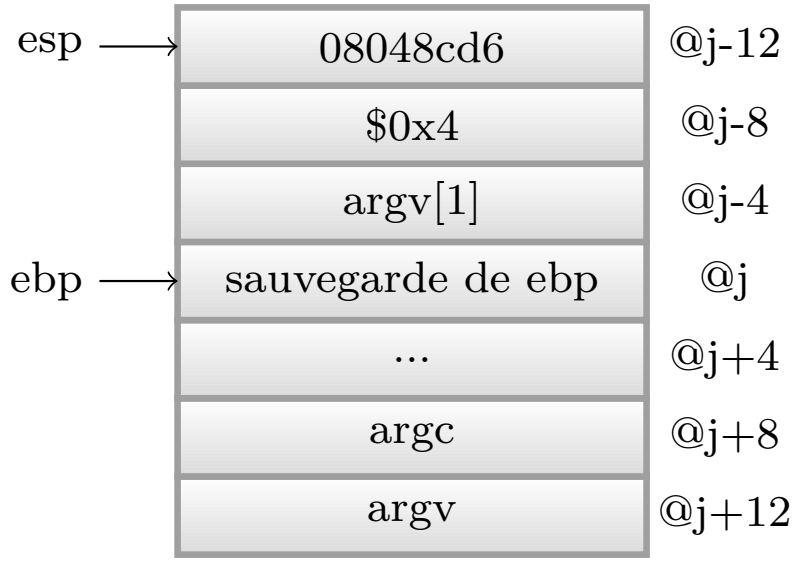
```
f :      08048cb0  push  %ebp
         08048cb1  mov   %esp,%ebp
         08048cb3  sub   $0x10,%esp
         08048cb6  leave
         08048cb7  ret
main :   08048cb8  push  %ebp
         08048cb9  mov   %esp,%ebp
         08048cbb  sub   $0x8,%esp
         08048cbe  mov   Oxc(%ebp),%eax
         08048cc1  add   $0x4,%eax
         08048cc4  mov   (%eax),%eax
         08048cc6  mov   %eax,Ox4(%esp)
         08048cca  movl  $0x4,(%esp)
    →    08048cd1  call  8048cb0
         08048cd6  leave
         08048cd7  ret
```



Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

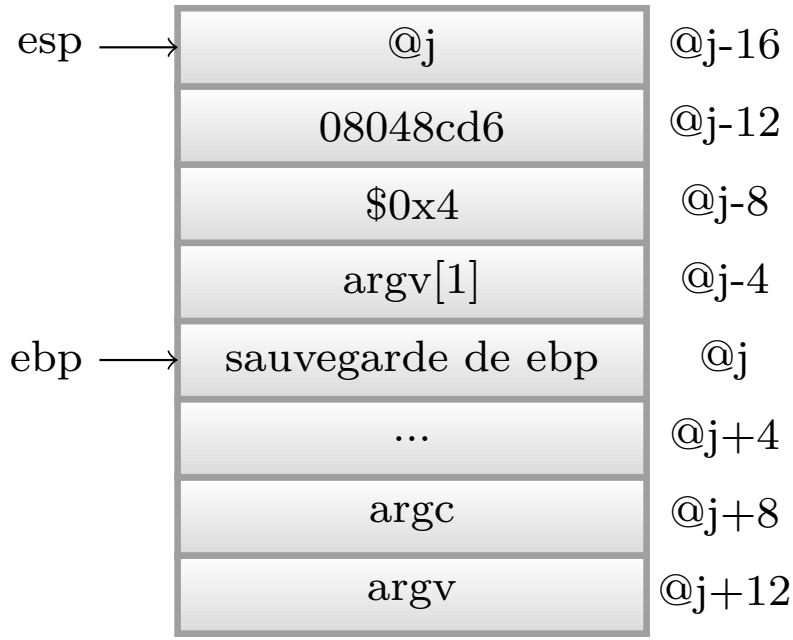
```
f :     → 08048cb0  push  %ebp
          08048cb1  mov   %esp,%ebp
          08048cb3  sub   $0x10,%esp
          08048cb6  leave
          08048cb7  ret
main :    08048cb8  push  %ebp
          08048cb9  mov   %esp,%ebp
          08048cbb  sub   $0x8,%esp
          08048cbe  mov   Oxc(%ebp),%eax
          08048cc1  add   $0x4,%eax
          08048cc4  mov   (%eax),%eax
          08048cc6  mov   %eax,Ox4(%esp)
          08048cca  movl  $0x4,(%esp)
          08048cd1  call  8048cb0
          08048cd6  leave
          08048cd7  ret
```



Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

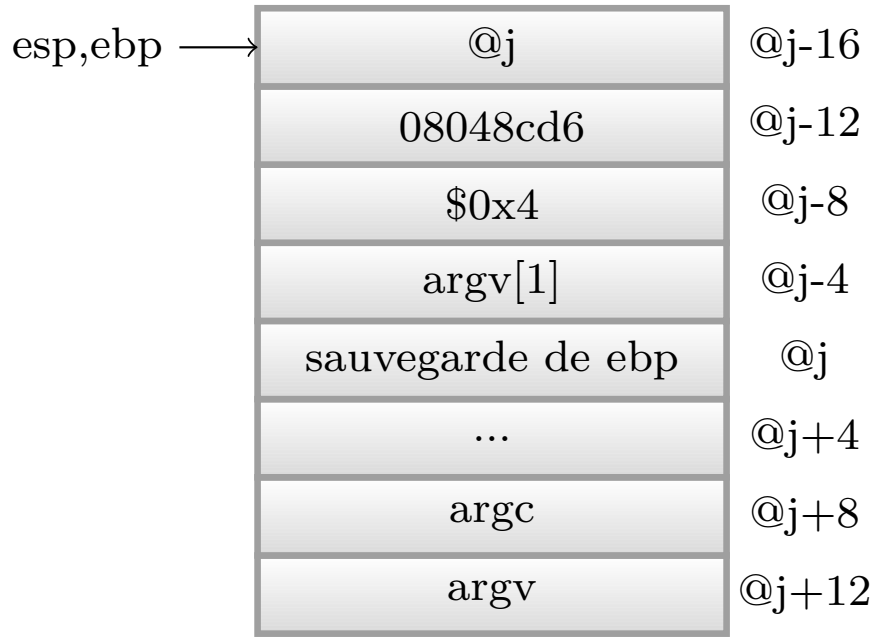
```
f :     08048cb0  push  %ebp
      → 08048cb1  mov   %esp,%ebp
        08048cb3  sub   $0x10,%esp
        08048cb6  leave
        08048cb7  ret
main :  08048cb8  push  %ebp
        08048cb9  mov   %esp,%ebp
        08048cbb  sub   $0x8,%esp
        08048cbe  mov   Oxc(%ebp),%eax
        08048cc1  add   $0x4,%eax
        08048cc4  mov   (%eax),%eax
        08048cc6  mov   %eax,Ox4(%esp)
        08048cca  movl  $0x4,(%esp)
        08048cd1  call  8048cb0
        08048cd6  leave
        08048cd7  ret
```



Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

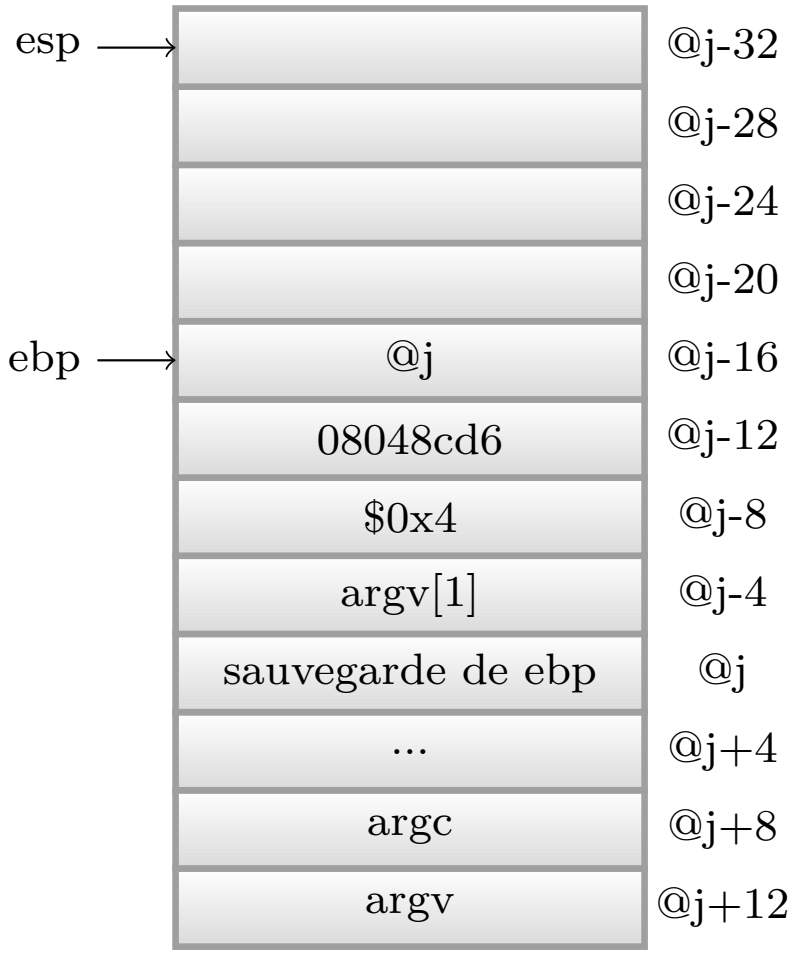
```
f :        08048cb0  push  %ebp
           08048cb1  mov   %esp,%ebp
      →    08048cb3  sub   $0x10,%esp
           08048cb6  leave
           08048cb7  ret
main :     08048cb8  push  %ebp
           08048cb9  mov   %esp,%ebp
           08048cbb  sub   $0x8,%esp
           08048cbe  mov   Oxc(%ebp),%eax
           08048cc1  add   $0x4,%eax
           08048cc4  mov   (%eax),%eax
           08048cc6  mov   %eax,Ox4(%esp)
           08048cca  movl  $0x4,(%esp)
           08048cd1  call  8048cb0
           08048cd6  leave
           08048cd7  ret
```



Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

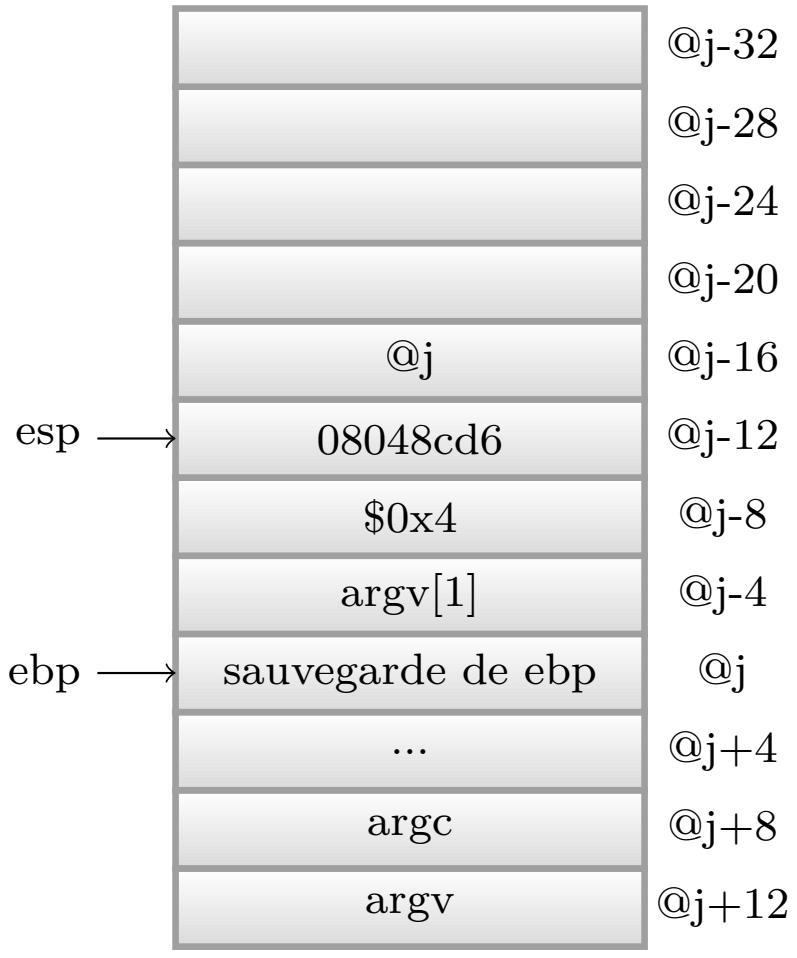
```
f :       08048cb0  push  %ebp
          08048cb1  mov   %esp,%ebp
          08048cb3  sub   $0x10,%esp
        → 08048cb6  leave
          08048cb7  ret
main :    08048cb8  push  %ebp
          08048cb9  mov   %esp,%ebp
          08048cbb  sub   $0x8,%esp
          08048cbe  mov   Oxc(%ebp),%eax
          08048cc1  add   $0x4,%eax
          08048cc4  mov   (%eax),%eax
          08048cc6  mov   %eax,Ox4(%esp)
          08048cca  movl  $0x4,(%esp)
          08048cd1  call  8048cb0
          08048cd6  leave
          08048cd7  ret
```



Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

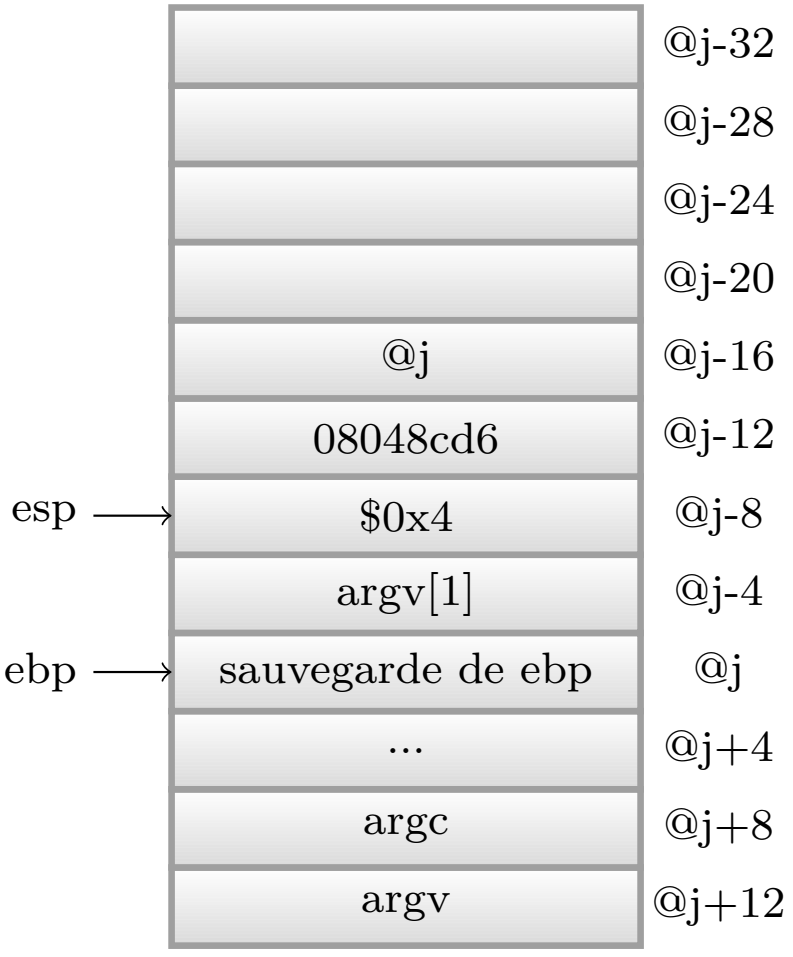
```
f :       08048cb0  push  %ebp
          08048cb1  mov   %esp,%ebp
          08048cb3  sub   $0x10,%esp
          08048cb6  leave
        → 08048cb7  ret
main :    08048cb8  push  %ebp
          08048cb9  mov   %esp,%ebp
          08048cbb  sub   $0x8,%esp
          08048cbe  mov   Oxc(%ebp),%eax
          08048cc1  add   $0x4,%eax
          08048cc4  mov   (%eax),%eax
          08048cc6  mov   %eax,Ox4(%esp)
          08048cca  movl  $0x4,(%esp)
          08048cd1  call  8048cb0
          08048cd6  leave
          08048cd7  ret
```



Appel de fonction sur x86 2/3

Dćompilation du binaire : `objdump -d program` 2

```
f :      08048cb0  push  %ebp
         08048cb1  mov   %esp,%ebp
         08048cb3  sub   $0x10,%esp
         08048cb6  leave
         08048cb7  ret
main :   08048cb8  push  %ebp
         08048cb9  mov   %esp,%ebp
         08048cbb  sub   $0x8,%esp
         08048cbe  mov   Oxc(%ebp),%eax
         08048cc1  add   $0x4,%eax
         08048cc4  mov   (%eax),%eax
         08048cc6  mov   %eax,Ox4(%esp)
         08048cca  movl  $0x4,(%esp)
         08048cd1  call  8048cb0
    →    08048cd6  leave
         08048cd7  ret
```

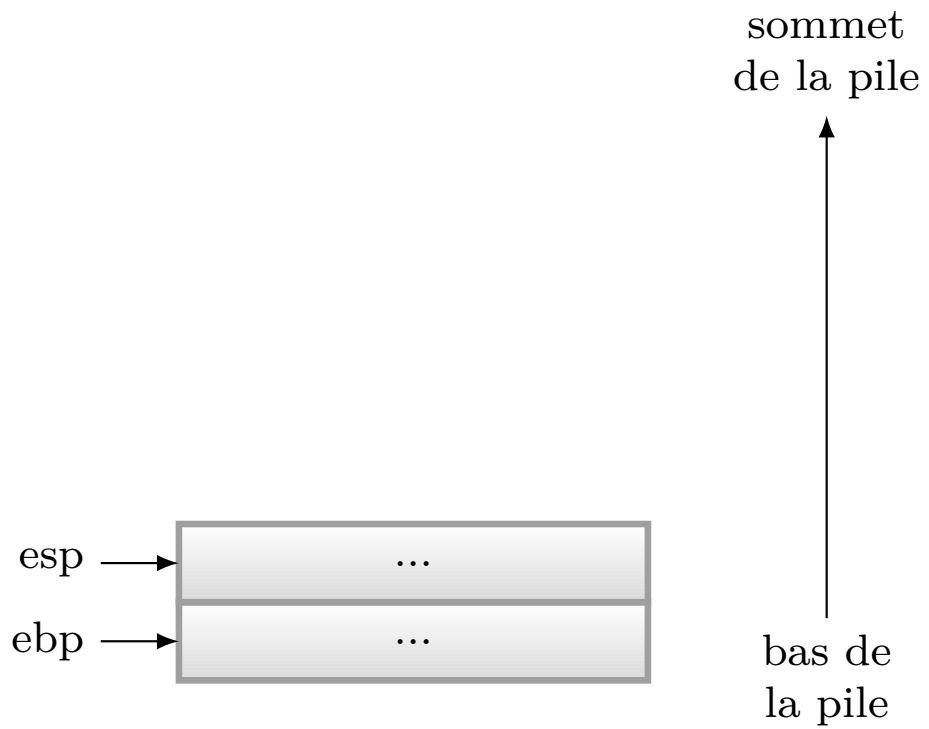


Appel de fonction sur x86 3/3

- Etat de la pile avant et durant l'invocation de la fonction

```
void f(int a, char* str) {
   char ch[8];
   int var;
}
void main(int argc, char** argv) {
   f(4, argv[1]);
}
```

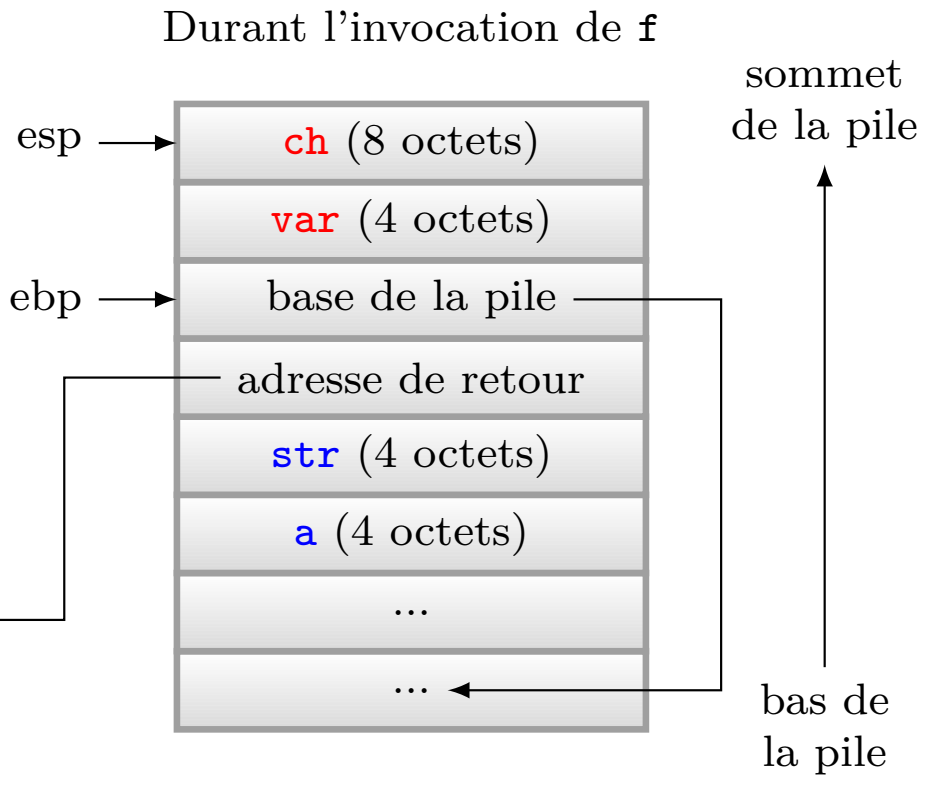
Avant l'invocation de `f`



Appel de fonction sur x86 3/3

- Etat de la pile avant et durant l'invocation de la fonction

```
void f(int a, char* str) {
   char ch[8];
   int var;
}
void main(int argc, char** argv) {
   f(4, argv[1]);
}
```



Récapitulatif

- Une fonction a besoin d'un espace pour stocker ses variables locales
- Une fonction peut être récursive (même indirectement)

$\Rightarrow$ A un instant, il peut y avoir deux contextes pour cette fonction

- L'espace dédié aux variables locales doit être propre à chaque exécution de la fonction
- Une fonction doit pouvoir localiser ses paramètres fournis par la fonction appelante
- A la fin d'une fonction, le processeur doit pouvoir déterminer l'adresse à laquelle poursuivre l'exécution – dans la fonction appelante
- Une fonction peut être invoquée par différentes fonctions appelante $\Rightarrow$ l'adresse de retour n'est pas unique
- Une fonction doit être indépendante des implémentations des autres fonctions
- Une fonction doit savoir comment invoquer les autres fonctions sans connaître leur implémentation
- Une fonction doit pouvoir récupérer ses paramètres sans connaître l'implémentation des fonctions appelantes

Introduction

Rappels sur les appels de fonction

Examples

Edition de liens

- Compilation pour 32 bits
  `gcc -m32 ld.c`
- Obtention du script d'édition de liens
  `gcc -m32 ld.c -Wl,-verbose`
- Décompilation
  `objdump -D a.out`
- Compilation pour 64 bits
  `gcc ld.c`
- Obtention du script d'édition de liens
  `gcc ld.c -Wl,-verbose`
- Décompilation
  `objdump -D a.out`

```c
int main(void) {
  return 0;
}
```

Calculs arithmétiques

- Compilation pour 32 bits
  `gcc -m32 -mpreferred-stack-boundary=2 math.c`
- Décompilation
  `objdump -D a.out`

```
int f(int argc,
      char** argv) {
  int i = atoi(argv[1]);
  int j = i + 123;
  return j;
}

int main(int argc,
      char** argv) {
  f(argc, argv);
}
```

Chaînes de caractères

- Compilation pour 32 bits
  `gcc -O0 -m32 -mpreferred-stack-boundary=2 string.c`
- Décompilation
  `objdump -D a.out`
- Tester différentes longueurs de chaînes

```
int f(void) {
  char s[16] =
    "aaaaaaaaaaaaaaaa";
}

int main(int argc,
    char** argv) {
  f();
}
```

Chaînes de caractères (suite)

- Compilation pour 32 bits
  `gcc -O0 -m32 -mpreferred-stack-boundary=2 string.c`
- Décompilation
  `objdump -D a.out`

```
void f(void) {
  char a[] = "AAA";
  char *b = "BBB";
  a[0] = 'C';
  b[0] = 'D';
}
int main(void) {
  f();
  return 0;
}
```

Invocations

- Compilation pour 32 bits
  `gcc -O0 -m32 -mpreferred-stack-boundary=2 string.c`
- Décompilation
  `objdump -D a.out`

```c
#include <stdio.h>

int f1(int i) {
  i = 12;
}

int f2(int *i) {
  *i = 12;
}

void p(int n) {
  printf("a: %d\n", n);
}

int main(void) {
  int a = 21;
  p(a);
  f1(a);
  p(a);
  f2(&a);
  p(a);
  return 0;
}
```