

Assembleur

EXERCICE 1 – Compilation de C vers l'assembleur

- 1) Compilez le code suivant avec la ligne commande : `gcc -S -Wall -o exercice.s exercice.c`

```
#include <stdio.h>
int main()
{
    int i;
    for (i=0; i<4; i++)
        if (i==2) printf("%i:Hello World !\n", i);
    return 0;
}
```

- 2) Regardez le contenu de `exercice.s`.

- 3) Recompilez le tout avec la ligne commande suivante : `gcc -Wall -o exercice exercice.c` et utilisez `objdump -d` sur l'exécutable ainsi généré et essayez d'identifier les différentes fonctions de chacun des éléments du binaire.

EXERCICE 2 – Désassemblage

Que font les codes assembleurs suivants :

- 1) Programme 1 :

```
movl    %esp, %eax
movl    $0, (%eax)
movl    $1, 4(%eax)
movl    $2, 8(%eax)
```

- 2) Programme 2 :

```
movl    $1, %eax
addl    $3, %eax
addl    $5, %eax
addl    $7, %eax
```

- 3) Programme 3 :

```
movl    %edx, %eax
addl    %ecx, %edx
addl    %eax, %ecx
```

- 4) Programme 4 :

```
pushl   %ebp
movl    %esp, %ebp
movl    12(%ebp), %eax
addl    8(%ebp), %eax
movl    %ebp, %esp
popl    %ebp
ret
```

EXERCICE 3 – Utilisation des modes d'adressage

Que fait le code assembleur suivant :

```

pushl %ebp
movl  %esp, %ebp
pushl %ebx
movl  12(%ebp), %ecx
movl  8(%ebp), %edx
movl  (%ecx), %eax
movl  (%edx), %ebx
movl  %eax, (%edx)
movl  %ebx, (%ecx)
movl  -4(%ebp), %ebx
movl  %ebp, %esp
popl  %ebp
ret

```

EXERCICE 4 — Utilisation de lea pour du calcul
Que fait le code assembleur suivant :

```

pushl %ebp
movl  %esp, %ebp
movl  8(%ebp), %eax
movl  12(%ebp), %edx
leal  (%edx,%eax), %ecx
leal  (%edx,%eax,2), %edx
sall  $4, %edx
addl  16(%ebp), %ecx
leal  4(%edx,%eax), %eax
imull %ecx, %eax
movl  %ebp, %esp
popl  %ebp
ret

```

EXERCICE 5 — Instructions Shift et Rotate
Que fait le code assembleur suivant :

```

pushl %ebp
movl  %esp, %ebp
movl  8(%ebp), %eax
xorl  12(%ebp), %eax
sarl  $17, %eax
andl  $8185, %eax
movl  %ebp, %esp
popl  %ebp
ret

```

EXERCICE 6 — Programmation par sousroutines en assembleur

- 1) Faites un programme assembleur qui calcule $x = 2x - y + z$ (avec x , y et z les arguments du programme).
- 2) Écrivez un programme minimal qui contient un appel à la fonction `printf` et compilez-le avec l'option `-S` pour regarder l'assembleur généré.
- 3) Écrivez en assembleur une fonction qui calcule la factorielle de 5 et qui affiche le tout grâce à la fonction `printf`.

EXERCICE 7 — Position des variables locales en mémoire

Comparer les programmes assembleurs des deux programmes suivants lorsqu'ils sont compilés avec `gcc -S -O0 -Wall -o <src>.s <src>.c`.

– Programme 1 :

```

int main() {
    static int i;
    int j;

```

```

        i=i+j;
        return i;
    }

```

– Programme 2 :

```

int main() {
    int i,j;
    i=i+j;
    return i;
}

```

- 1) Quelles différences notables constatez-vous au niveau de la pile mémoire ?
- 2) Que constatez-vous si vous ajoutez le mot-clef static devant la déclaration de j ?

EXERCICE 8 – Niveaux d’optimisation

Compilez le programme suivant avec les différents niveaux d’optimisation suivants : -O0, -Os, -O1, -O2, -O3.

```

int foo(int i) {
    while (i) i--;
    return i;
}

int main() {
    int i;
    for (i=0; i<10; i++)
        foo(i);
    return i;
}

```

- 1) Quelles différences constatez-vous sur le code assembleur des différentes sorties ?
- 2) Essayez de représenter les états de la pile pour les premières itérations de la fonction main.

EXERCICE 9 – C vs. C++

Comparez le code assembleur généré par les deux programmes "Hello World!" suivants :

– Le programme C :

```

/* hello.c */
#include <stdio.h>
int main() {
    printf("Hello World !\n");
    return 0;
}

```

– Le programme C++ :

```

/* hello.cpp */
#include <iostream>
using namespace std;
int main() {
    cout << "Hello World !" << endl;
    return 0;
}

```

EXERCICE 10 – Code de retour sur un struct

Comment se passe le passage du code de retour lorsque la fonction retourne un struct ? Par exemple, pour le programme suivant :

```

#include <stdio.h>
#define LENGTH 10

typedef struct {
    char str[LENGTH];
    int value; } record_t;

```

```
record_t read_record() {
    record_t record;
    scanf("%s", record.str);
    return record;
}

int main() {
    record_t record;
    record = read_record();
    printf(record.str);
    return 0;
}
```

EXERCICE 11 – Les programmes mystères

Téléchargez les binaires suivants, analysez les pour savoir ce qu'ils font et tentez de reconstituer le programme C original.

- Programme 1 : <http://www.labri.fr/~fleury/courses/SS07/download/exercises/mystery-1>
- Programme 2 : <http://www.labri.fr/~fleury/courses/SS07/download/exercises/mystery-2>
- Programme 3 : <http://www.labri.fr/~fleury/courses/SS07/download/exercises/mystery-3>