Mécanismes de bas niveau

1 Architecture du processeur x64

1.1 Registres

Attention : le rangement en mémoire est petit-boutiste (*little-endian*), ce qui signifie que

-	*nom*	*nb. bits*	*signification*	*rôle*
-	`rip`			
-	`rflags`			
-	`rsp`			
-	`rbp`			
14 registres généraux	`rax` ...`rdx`, `r8` ...`r15`		numérotés	
8 registres nombres flottants	`fpr0` ...`fpr7`			
16 registres étendus	`xmm0` ...`xmm15`			
autres	-	-	-	réservés au système d'exploitation

Remarque : les registres sont souvent subdivisés en 2 : `eax`, `edi`...

1.2 Assembleur

— le format est : `instruction destination, source;`
— en réalité, le code est sous forme binaire illisible par un humain.

instruction	*signification*	*rôle*
`nop`		
`mov`		
`push`, `pop`	-	
`add`, `sub`, `mul` ,`div`	-	
`inc`		
`cmp`		
`jmp`, `jle`, `jge`...		
`call`	-	
`ret`		
etc.	-	-

1.3 Exemple

```c
#include <stdio.h>

int main(void) {
        for (int i = 0; i < 10; i++)
                printf("Bonjour !\n");
        return 0;
}
```

```
00000000004004f6 <main>:
  4004f6:       55                      push   rbp
  4004f7:       48 89 e5                mov    rbp,rsp
  4004fa:       48 83 ec 10             sub    rsp,0x10
  4004fe:       c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
  400505:       eb 0e                   jmp    400515 <main+0x1f>
  400507:       bf c0 05 40 00          mov    edi,0x4005c0
  40050c:       e8 df fe ff ff          call   4003f0 <puts@plt>
  400511:       83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
  400515:       83 7d fc 09             cmp    DWORD PTR [rbp-0x4],0x9
  400519:       7e ec                   jle    400507 <main+0x11>
  40051b:       b8 00 00 00 00          mov    eax,0x0
  400520:       c9                      leave
  400521:       c3                      ret
  400522:       66 2e 0f 1f 84 00 00    nop    WORD PTR cs:[rax+rax*1+0x0]
  400529:       00 00 00
  40052c:       0f 1f 40 00             nop    DWORD PTR [rax+0x0]
```

2 Débogueur (gdb)

Compiler avec l'option [ ], puis lancer [ ] :

```
gcc -g bonjour.c -o bonjour.out
gdb ./bonjour.out
```

instruction	*raccourci*	*rôle*
`list`	`-`	
`disassemble f`	`disass f`	
`breakpoint f`	`b f`	
`breakpoint 18`	`b 18`	
`run <params>`	`r <params>`	
`continue`	`c`	
`next`	`n`	
`nexti`	`ni`	
`step`	`s`	
`stepi`	`si`	
`backtrace`	`bt`	
`info registers`	`i r`	
`info register rip`	`i r rip`	
`x 0x7fffffffdd36`	`-`	
`quit`	`q`	

Options d'examen de la mémoire :

— nombre de blocs + taille du bloc + format
— exemple : `x/12gx $rsp`

taille	signification	traduction	nombre de bits
`b`			
`w`			
`g`			

format	signification
`x`	
`d`	
`f`	
`s`	
`i`	

3 Langage C

3.1 Quelques types de données

type	*printf*	*taille en bits*	*contenu*
`char`			
`int`			
`long`			
`long long`			
`float`			
`double`			
`void *`, `char *`, `int *`, `double *`...			
`char *`			

$\Longrightarrow$ utiliser `sizeof(int)` pour connaître la taille en octets d'un `int`, etc.

$\Longrightarrow$ %d pour décimal
$\Longrightarrow$ %x pour hexadécimal

3.2 Pointeurs

```
int x = 18;
int *p = &x;                                //
int y = *p;                                 //
++p;                                        //

int *tab = malloc (1000 * sizeof(int));     //
*(tab + 3) = -5                             //
free(tab);                                  //
tab = NULL;                                 //
```