Prise de note

intel et amd = cpu x86-64

téléphone = cpu ARM

unité de contrôle : lis le code instruction (101101111)

flags = ca indique l’état de la dernière opération, sans historique comme les logs

registre = 64 bits

mme traduit les adresses, cela a était rajouté au moment du passage des cpu a 32 bits

a chaque démarage les cpu refonds l’histoire de 16 bits a 64 bits en passant pour bootloader, le kernel démare en 64 bits (systemd ou pib 1)

uefi démarre a la place du bios “classique”

grub legacy : permet de boot sur un certain os

ne pouvais avoir que 4 partitions (annés 1970-80)

et pour envoir lus en devais définir une mbr pour créer des sous partitions

architecture ARM pas de rétrocompatibité, utilisé sur les raspberry py

w11 permet plus la rétrocoptibité, mais w10 oui

intel n’est pas réellement un vrais cisc

axel.mura@linux:~/Desktop$ size script

text data bss dec hex filename

1509 604 4 2117 845 script

bss = réserver les octets

data = stocke la variable

voir la traduction de C en ARM

axel.mura@linux:~/Desktop$ objdump -S script

GDB :

axel.mura@linux:~/Desktop$ gcc -o -g script script.c

axel.mura@linux:~/Desktop$ gdb ./script

n = next pas a pas

c= continue

run = lancer le programme

delete breakpoints

désactiver une breakpont = enable ou disable p

show breakpoints

Breakpoint 1, main () at script.c:6

6 for (i=0;i<10; i++){

(gdb) n

7 printf("hello \n");

(gdb) n

hello

6 for (i=0;i<10; i++){

(gdb) n

7 printf("hello \n");

(gdb) n

hello

6 for (i=0;i<10; i++){

(gdb) print i

$1 = 1

(gdb) n

7 printf("hello \n");

(gdb) n

hello

6 for (i=0;i<10; i++){

(gdb) n

7 printf("hello \n");

(gdb) print i

$2 = 3

(gdb) print &i

$3 = (int \*) 0x555555755018 <i>

(gdb) print i+42

$4 = 45

(gdb)

http://www.gdbtutorial.com/tutorial/how-install-gdb