
EPITECH.
L'ECOLE DE L'EXPERTISE INFORMATIQUE
Quand un passionné devient un expert reconnu
European Institute of Information Technology
Titre homologué par l'Etat - Niveau I (CNCP)
medega_j

LE MODULE

Déroulement :
- 1 Cours
- 4 TP
- Un projet (ASM Mini LibC)

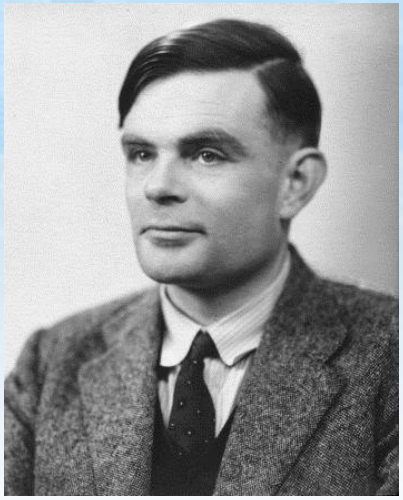
Objectifs :
- Compréhension des mécanismes de bas niveau

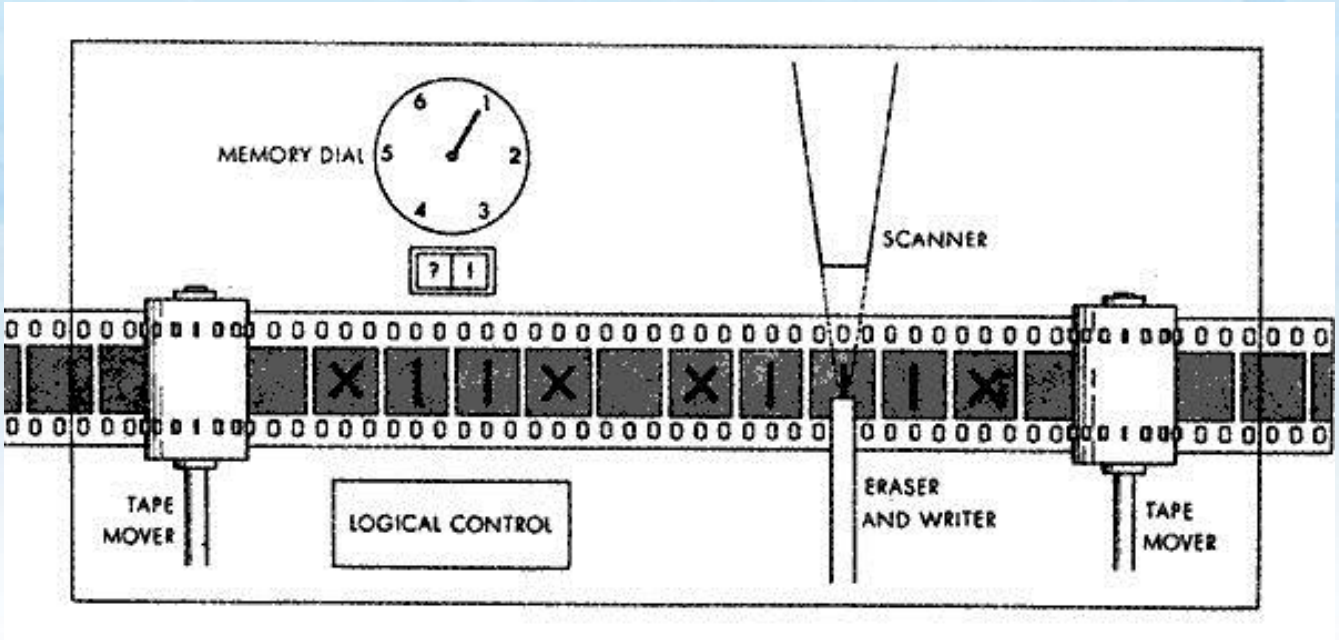
Environnement :
- Processeurs x86_64 (64 bits)
- Jeu d'instructions Intel i386 (http://www.intel.com/products/processor/manuals)
- Syntaxe Intel (vs AT&T)

Outils :
- Pour transformer l'assembleur en code binaire : nasm
- Pour créer des exécutables : gcc / ld
- Pour débugger : gdb

GENESIS

Alan Mathison Turing (La Bombe Vs Enigma) est considéré comme le père du processeur.





Avec le concept de la machine de Turing, est engagée l'association de la notion de calcul avec celle de la machine indépendante de l'intervention humaine.

AUJOURD'HUI

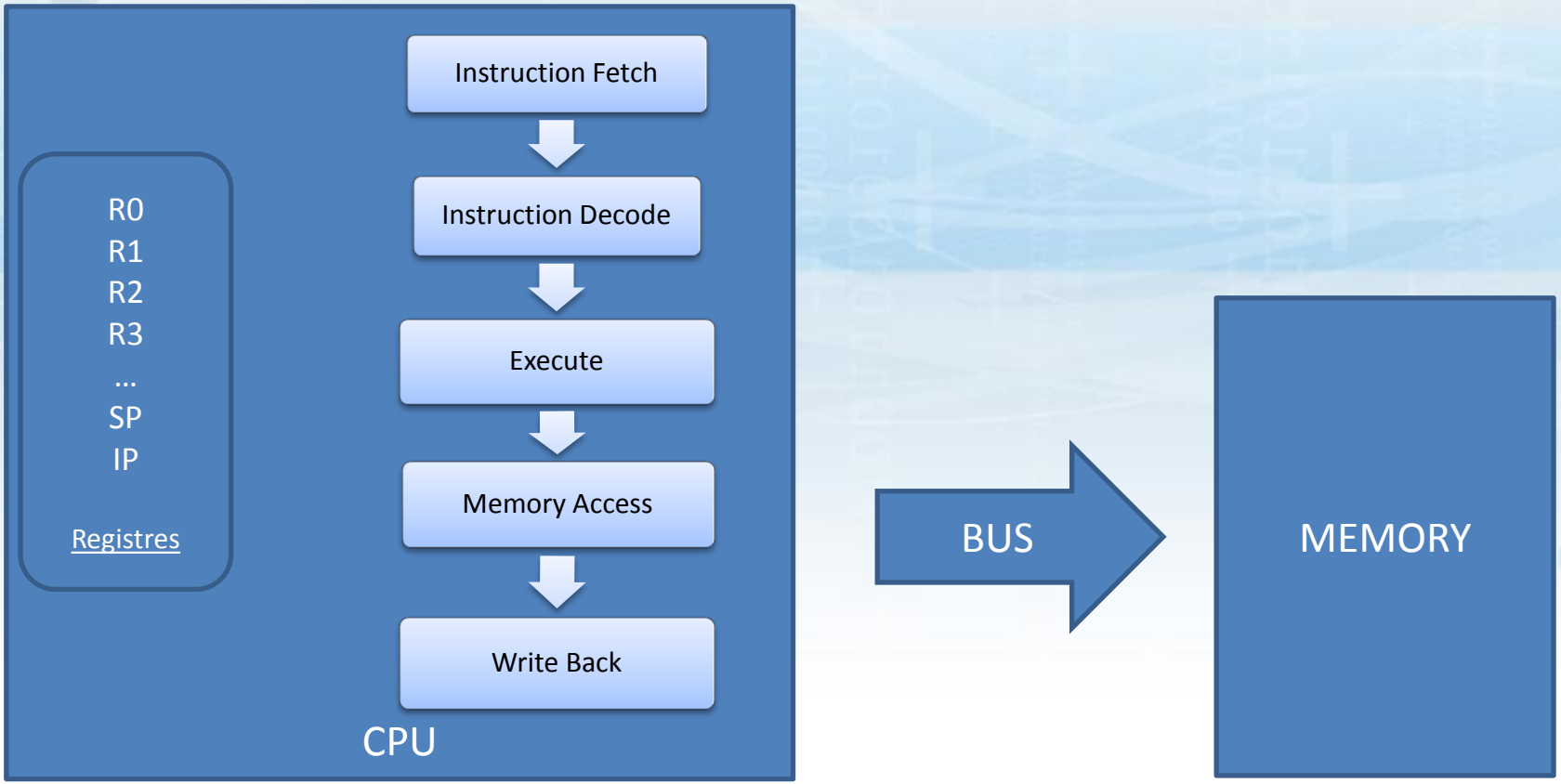
Deux grandes familles de processeurs :

- **RISC** : *Reduced Instruction Set Computer*
- **CISC** : *Complex Instruction Set Computer*

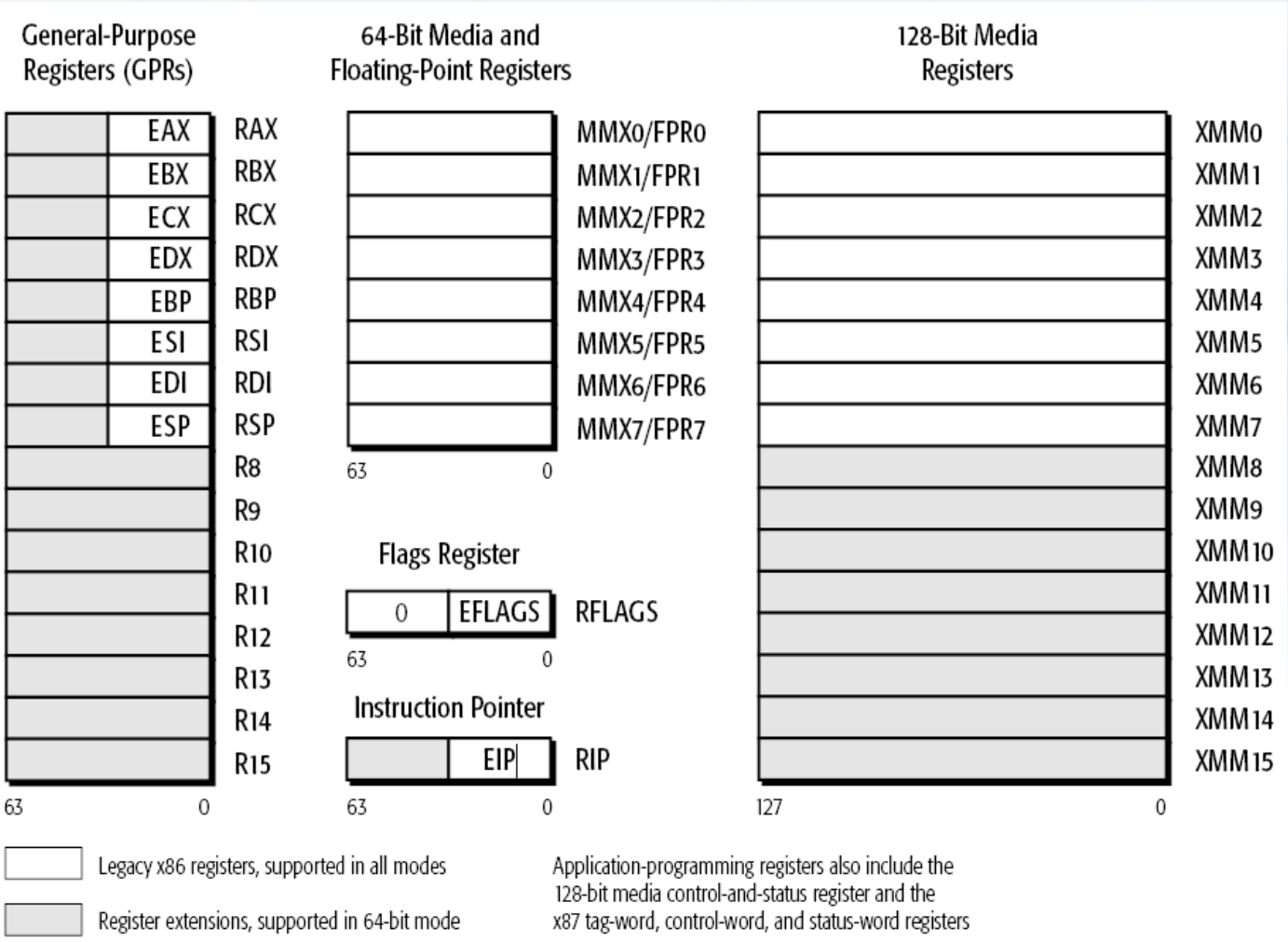
Notre cas d'étude :

- Les processeurs x86_64 (extension de la norme x86)
- Le jeu d'instructions i386 (CISC)

LE PROCESSEUR, LA MÉMOIRE & LES REGISTRES



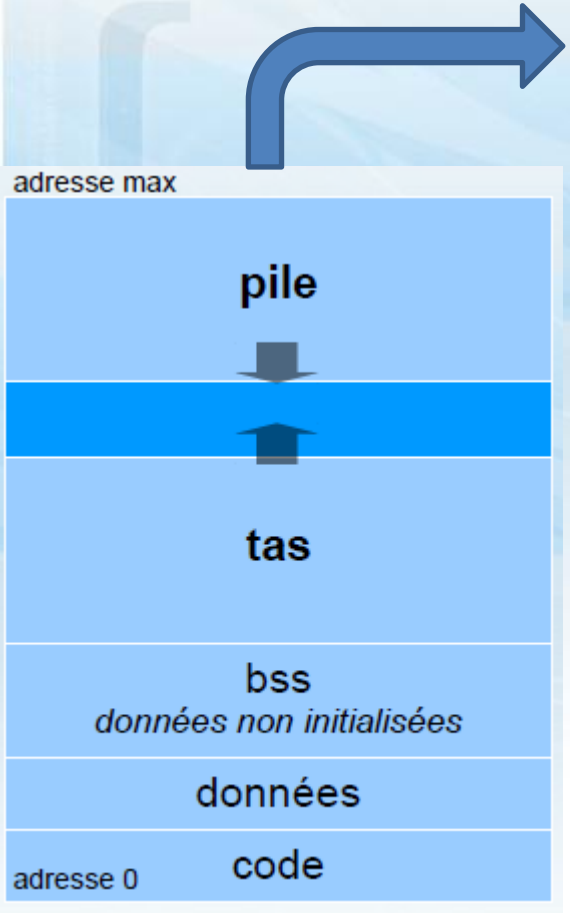
LES REGISTRES



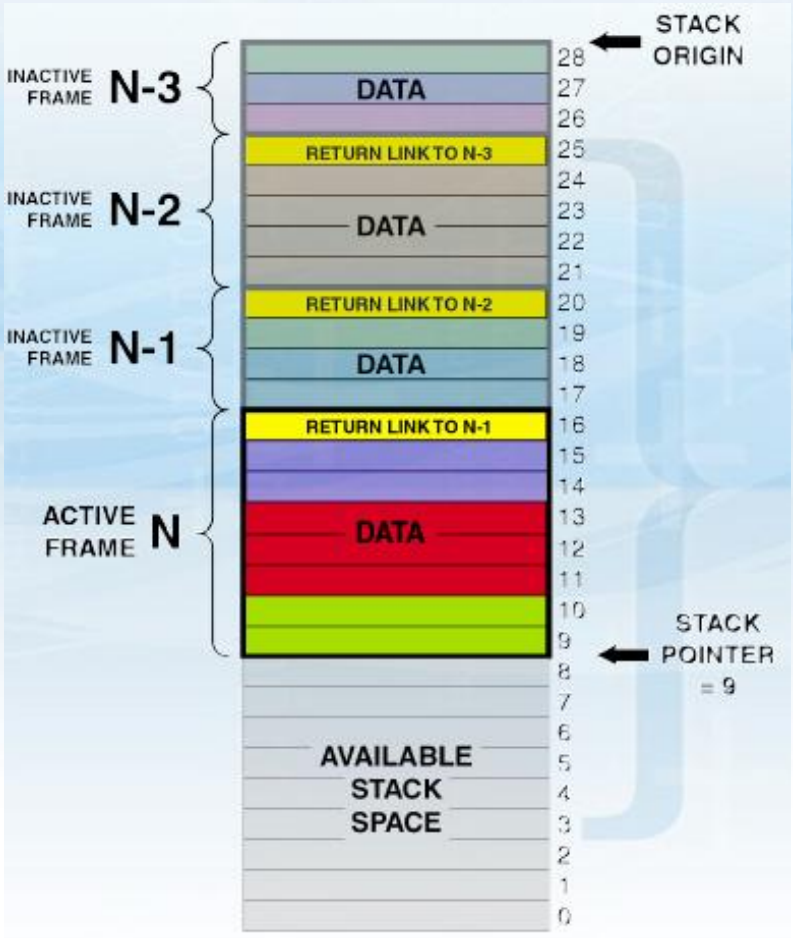
Roles conventionnels historiques (16-Bit) :

- AX : Accumulator Register
- BX : Base Register
- CX : Counter Register
- DX : Data Register
- SI : Source Index
- DI : Destination Index
- BP : Base Pointer
- SP : Stack Pointer

LA MEMOIRE



Mémoire d'un processus



La pile

La pile :

- Last In First Out (push/pop)
- Se remplit des adresses hautes vers les adresses basses
- RSP (Stack Pointer) pointe sur la prochaine case qui sera exploitée par un push ou un pop (la fin de l'espace exploité de la pile)
- RBP (Base Pointer) pointe sur le début du cadre de pile de la fonction courante
- Permet le passage d'arguments & l'usage des variables locales

L'ASM : A QUOI BON ?

Sécurité :
- Hacking (Buffer, Heap, Integer overflow, ROP & autres exploitations)
- Recherche (Reverse engineering)
- Cracking / Patching

Développement :
- Opérations noyau (exploitation directe du processeur par l'OS) de bas niveau
- Pilotes de périphériques
- Optimisations de routines (WARNING !!)

Embarqué :
- Constructions des premières couches de haut niveau sur du matériel vierge
- …

L'ASM : LA SYNTAXE

Une instruction = **mnémonique + opérande(s) [commentaire(s)]**

- **mnémonique** : représente l'instruction à exécuter, liée à un « *opcode* »
- **opérande(s)** : argument(s) de l'instruction, séparés par des virgules
- **commentaire(s)** : commence par « *;* »

  ex : mov rax,4 ; stockage de la valeur 4 dans rax
- ***label*** : permet de fixer un point du programme pour l'appeler ensuite
- ***directive*** : information pour le compilateur

  ex : db (define byte), dword (double word), .text, .data, global, bits

Différents modes d'adressage :

- **registre** : mov rax, rbx
- **direct** : mov rax, *adresse*
- **immédiat** : mov rax, 4
- **indirect** : mov rax, [rbx]

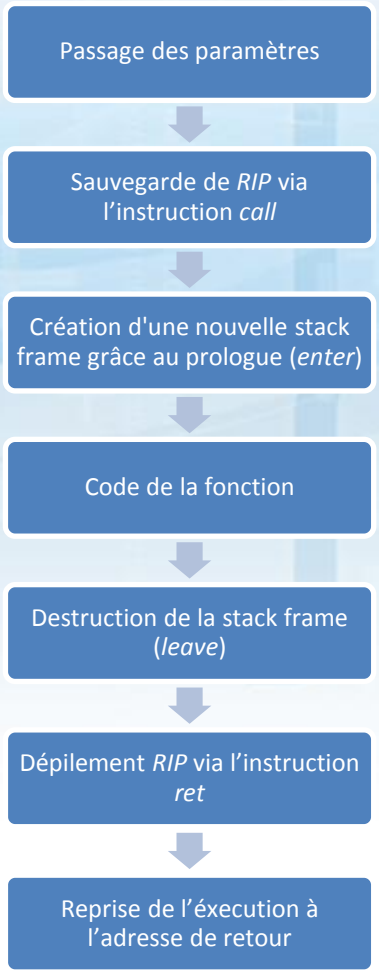
Les conventions d'appels

Appels de fonctions externes :

- Les paramètres de type entier/pointeur sont placés dans l'ordre de passage, dans *rdi*, *rsi*, *rdx*, *rcx*, *r8* et *r9*.
- Les arguments à virgule flottante sont placés dans l'ordre de passage dans *xmm0*, *xmm1*, *xmm2*, *xmm3*, *xmm4*, *xmm5*, *xmm6*, *xmm7*.
- Les fonctions à nombre d'arguments variable (printf, …) nécessitent dans *rax*, le nombre d'arguments à virgule flottante passés en paramètres via les régistres *xmm**.
- On passe le reste des arguments (s'il y en a donc plus de 6 ou 8 suivant le type d'argument), sur la pile.
- Les régistres *rbp*, *rbx* et *r12* à *r15* sont préservés par les routines, les autres sont utilisés sans sauvegarde.
- Les valeurs de retour de type entier/pointeur sont placées dans *rax*; celles de type virgule flottante sont placées dans *xmm0*.

Appels de sycalls :

- Les arguments sont passés dans les registres comme pour les appels de fonctions "userland" (sauf le 4ème paramètre qui est placé dans r10). On met dans rax, le numéro du syscall (/usr/include/asm/unistd_64.h) puis on déclenche l'appel avec l'instruction "*syscall*".



Hello World

```
[medega_j@fedor asm]$ nasm -f elf64 hello.S -o hello.o
[medega_j@fedor asm]$ gcc hello.o -o hello
[medega_j@fedor asm]$
```

```
[BITS 64]

        ;; déclaration des symboles
        global main             ;; export du symbole
        extern printf           ;; symbole importé

        ;; code
        section .text           ;; déclaration d'une section pour contenir le code

main:                           ;; déclaration d'un label
        push rbp                ;; prologue
        mov rbp, rsp

        mov rdi, FormatStr      ;; passage du premier argument de printf
        call printf             ;; appel à printf

        mov rsp, rbp            ;; epilogue
        pop rbp

        mov rax, 60             ;; numéro du syscall "exit"
        xor rdi, rdi            ;; passage du premier (& unique) paramètre dans rdi  (mov rdi, 0h ⇔ rdi ^= rdi)
        syscall                 ;; appel au noyau pour exécution

        ret                     ;; sortie de la fonction

        ;; read only data
        section .rodata         ;; déclaration d'une section pour contenir les données initialisées

        FormatStr db 'Hello World !',0Ah,0
```

- Directives nasm
- Opcodes
- Opérandes
- Labels
- Commentaires
- Variables
- Fonctions

Questions ?