

Ressource R2.04

Communication et fonctionnement bas niveau

Cours n°1

Organisation d'un ordinateur

Exécution d'un programme

version du 22 janvier 2023

Présentation de la ressource

Descriptif

L'objectif de cette ressource est de comprendre le fonctionnement des couches systèmes et réseaux bas niveau. Cette ressource permet de découvrir les multiples technologies et fonctions mises en œuvre dans un réseau informatique et de comprendre les rôles et structures des mécanismes bas niveau mis en oeuvre pour leur fonctionnement.

Savoirs de référence étudiés

- Étude d'un système à microprocesseur ou microcontrôleur avec ses composants (mémoires, interfaces, périphériques...)
- Langages de programmation de bas niveau et mécanismes de bas niveau d'un système informatique
- Étude d'architectures de réseaux et notion de pile protocolaire
- Technologie des réseaux locaux : Ethernet, wifi, TCP/IP, routage, commutation, adressage, transport

Mots clés

Protocoles – Pointeurs – Interruptions – Langage bas niveau

Présentation de la ressource

Descriptif

L'objectif de cette ressource est de comprendre le fonctionnement des couches systèmes et réseaux bas niveau. Cette ressource permet de découvrir les multiples technologies et fonctions mises en œuvre dans un réseau informatique et de comprendre les rôles et structures des mécanismes bas niveau mis en œuvre pour leur fonctionnement.

Savoirs de référence étudiés

- Étude d'un système à microprocesseur ou microcontrôleur avec ses composants (mémoires, interfaces, périphériques...)
- Langages de programmation de bas niveau et mécanismes de bas niveau d'un système informatique
- Étude d'architectures de réseaux et notion de pile protocolaire
- Technologie des réseaux locaux : Ethernet, wifi, TCP/IP, routage, commutation, adressage, transport

Mots clés

Protocoles – Pointeurs – Interruptions – Langage bas niveau

Contexte

On a enregistré le fichier `main.go` avec le contenu suivant :

```
package main

import "fmt"

func main() {
    var hello = "Hello, BUT"
    var helloAsRuneArray = []rune(hello)
    for i := 0; i < len(helloAsRuneArray); i++ {
        fmt.Print(string(helloAsRuneArray[i]), ".")
    }
    fmt.Println()
}
```

On souhaite exécuter ce programme.

Organisation d'un ordinateur

Quelques mots sur la hiérarchie de la mémoire

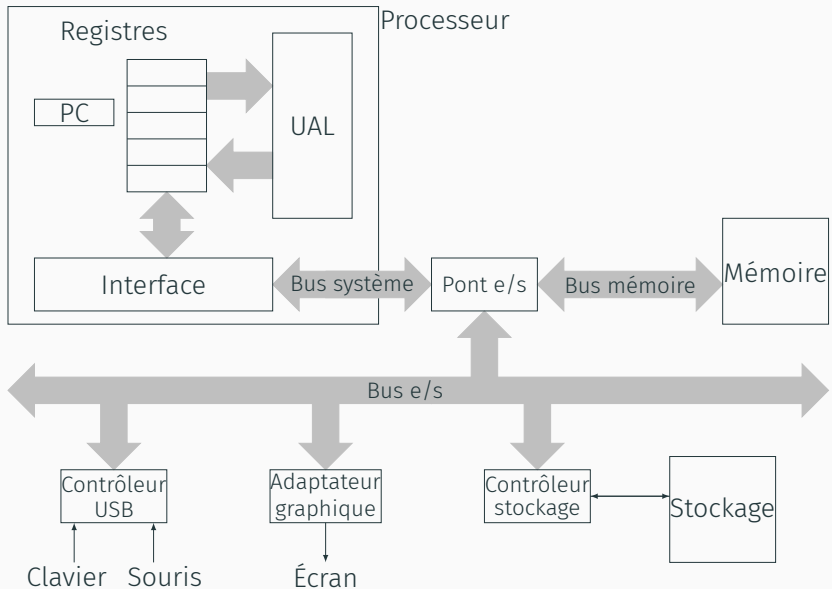
Le processeur

Le langage machine

Exécution des instructions

En pratique

Organisation générale d'un ordinateur



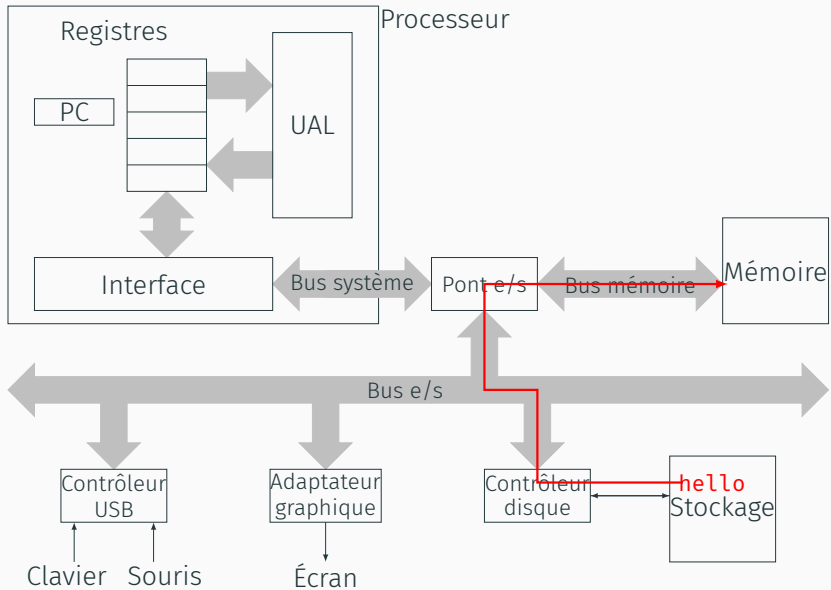
Première étape : chargement du programme en mémoire

Le programme est stocké sur une mémoire de masse, dans un fichier exécutable (p.ex. au format ELF¹).

Pour qu'il puisse être exécuté, il doit être copié (on dit chargé) en mémoire.

Cette étape est pilotée par le système d'exploitation (ou OS pour *Operating System*).

1. *Executable and Linkable Format* : format pour les fichiers objets et binaires.

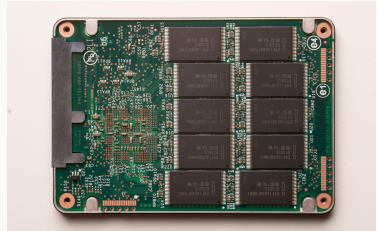


Le stockage de masse

Disque dur

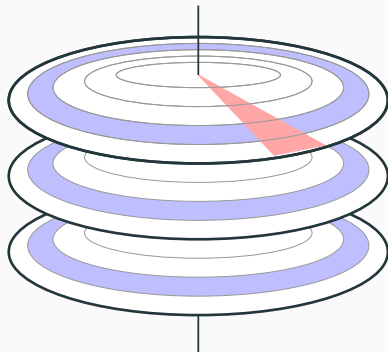
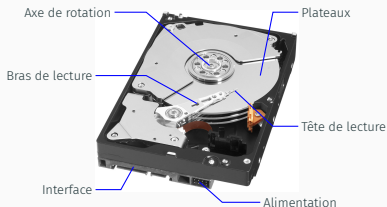


SSD



Principe de fonctionnement d'un disque dur

Stockage de masse persistant par modification du champ magnétique d'un matériau ferromagnétique



Avantages et inconvénients d'un disque dur

Avantages

- Grande capacité (plusieurs TB)
- Prix par octet
- Durabilité du stockage magnétique

Inconvénients

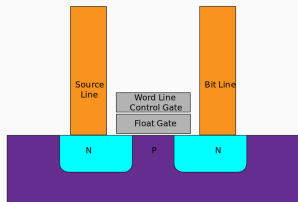
- Lent (débit, et surtout **latence**)
- Temps d'accès non homogène (la position des données sur le disque est importante)
- Fragilité du système électro-mécanique

SSD : principe de fonctionnement

SSD est l'acronyme de *Solid State Drive*).

C'est un stockage de masse persistant en mémoire Flash

Une cellule de mémoire Flash est construite autour d'un transistor à grille flottante qui permet *d'emprisonner* des électrons et ainsi stocker 1 bit.



Avantages et inconvénients d'un disque SSD

Avantages

- Rapidité (latence et débit)
- Robustesse (absence de pièce mécanique)

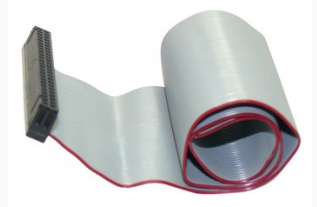
Inconvénients

- Prix par octet
- Importance du *firmware* et du pilote sur les performances et la durée de vie
- Peu adapté aux systèmes réalisant des écritures répétées (capture vidéo)
- Durabilité réduite de la technologie de stockage (environ 10 ans avec les techno. actuelles)

Transport de l'information : les bus

Bus = ensemble de fils électriques.

Les bus sont conçus pour transmettre des **mots** de taille fixe (de 8 à 64 bits), qui peuvent être une/des commande(s), une adresse, ou une/des donnée(s).



Bus parallèle : n fils (n = largeur du mot), mots transmis un bit par fil.

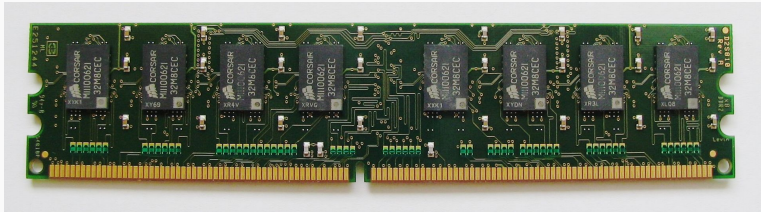
Exemple : processeur ↔ mémoire (données, adresse, contrôle, horloge)

Bus série : 1 fil, mot transmis bit après bit

Exemple : disque dur SATA (Serial ATA).

La mémoire centrale (non persistante)

Principalement : DRAM
Dynamic Random Access Memory



Caractéristiques

- Stockage d'un bit : charge d'un condensateur → nécessité de rafraîchir la mémoire (chaque 10 à 100 ms)
- Données stockées dans des supercellules (généralement 8 bit) organisées en grille
- Logique d'accès réalisée à l'aide de transistors CMOS

Organisation d'un ordinateur

Quelques mots sur la hiérarchie de la mémoire

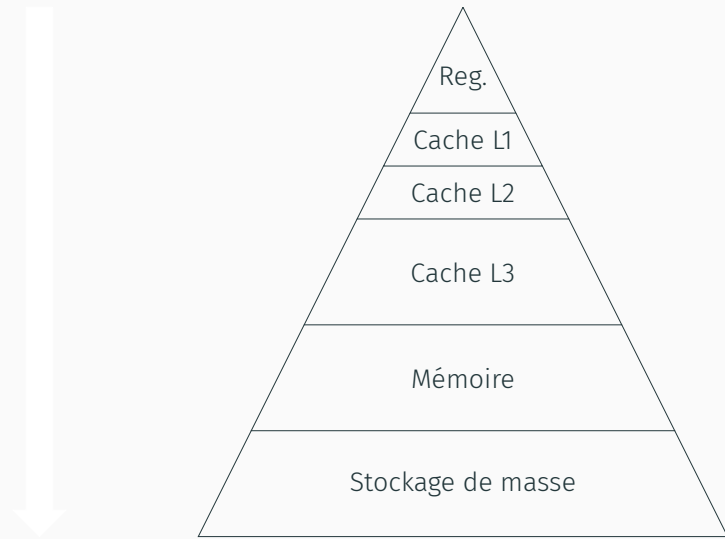
Le processeur

Le langage machine

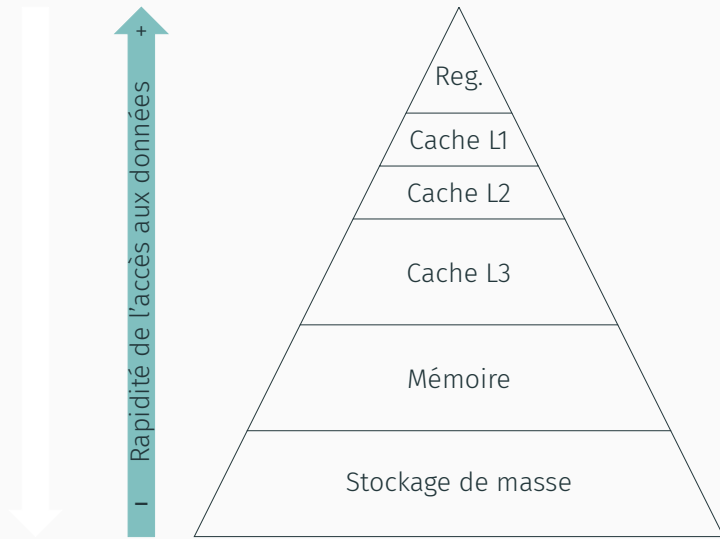
Exécution des instructions

En pratique

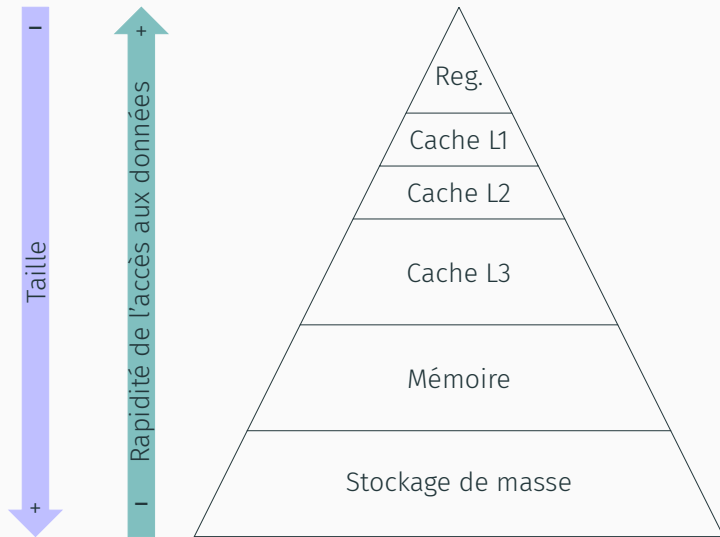
Hiérarchie de la mémoire, principe



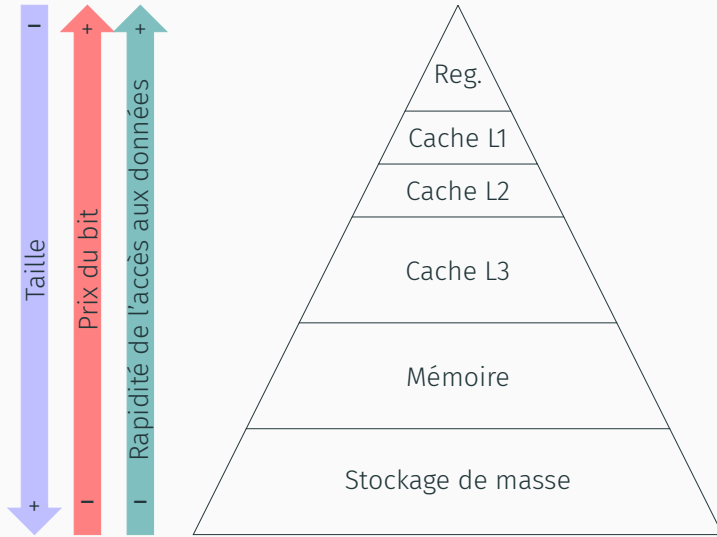
Hiérarchie de la mémoire, principe



Hiérarchie de la mémoire, principe



Hiérarchie de la mémoire, principe



Principe de fonctionnement de la hiérarchie mémoire

Chaque niveau sert de **cache** au niveau suivant :

- contient un petit sous-ensemble des données du niveau suivant
- offre un accès plus rapide à ces données

L'accès se fait toujours au niveau le plus haut. Si la donnée recherchée n'est pas présente, elle est chargée depuis les niveaux inférieurs vers les niveaux supérieurs puis l'accès est recommencé.

Les transferts se font uniquement entre niveaux voisins.

Attention :

- Quand une donnée est modifiée, il faut propager vers les niveaux inférieurs
- Dans les systèmes multiprocesseurs, plusieurs niveaux sont présents en plusieurs exemplaires (L1, L2) : il faut assurer leur cohérence.

Localités spatiale et temporelle

Pour que la hiérarchie mémoire fonctionne bien, il faut que les données cachées (= les données des niveaux supérieurs) soient bien choisies.

Localité spatiale

Si une donnée est utilisée, les données proches (c'est-à-dire situées à des adresses mémoire proches) seront sans doute bientôt utiles.

Localité temporelle

Une donnée qui vient d'être utilisée sera sans doute bientôt utile à nouveau dans un avenir proche.

Pourquoi un système aussi compliqué ?

Latency numbers every programmer should know

Organisation d'un ordinateur

Quelques mots sur la hiérarchie de la mémoire

Le processeur

Le langage machine

Exécution des instructions

En pratique

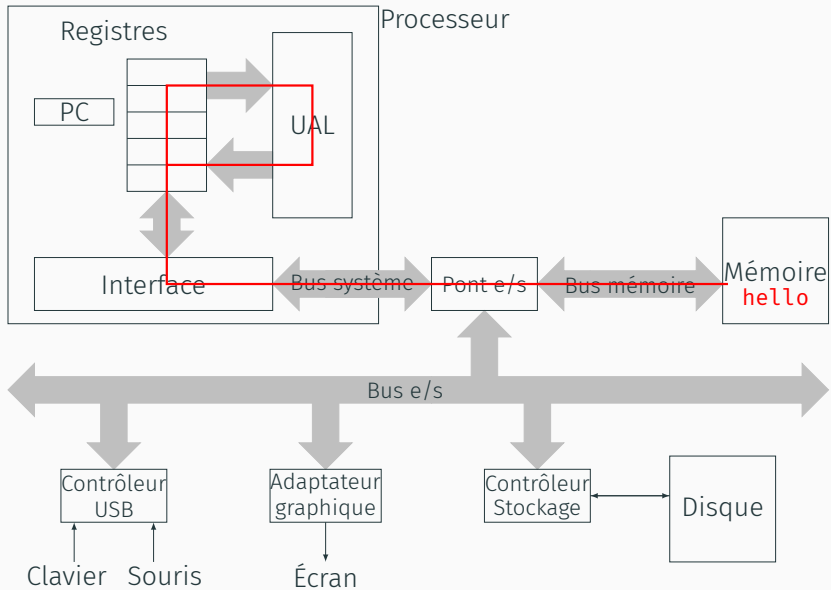
Deuxième étape : exécution depuis la mémoire

Une fois le programme chargé, le processeur peut commencer à exécuter la liste des instructions qu'il contient.

Les instructions sont lues depuis la mémoire puis décodées et exécutées dans le processeur.

En plus des calculs, le processeur lit et écrit en mémoire les données manipulées par le programme.

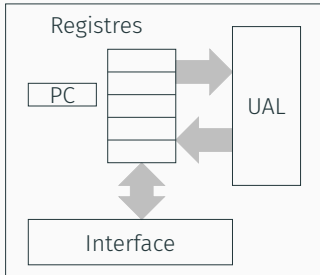
Les instructions du programme peuvent également amener le processeur à accéder aux autres périphériques d'entrée-sortie de la machine.



Le processeur

Il est cadencé par une horloge interne construite autour d'un crystal de quartz : à chaque top il exécute une (partie d'une) **instruction**.

C'est le **compteur de programme** qui indique l'adresse en mémoire de l'instruction suivante.



Les registres

Le processeur contient de la mémoire : des **registres** de tailles fixes (1 mot mémoire). Pour assurer des accès ultra-rapides, ils sont conçus en SRAM (Static RAM, stockage via des transistors CMOS)

Certains sont visibles du programme, qui y accède par leur **nom**. La plupart servent à stocker les opérandes et/ou les résultats des instructions exécutées par le processeur.

D'autres ont un rôle particulier, par exemple :

- le compteur de programme,
- le registre de status,
- ou encore le pointeur de pile,

Le processeur contient également des registres privés, utiles pour son fonctionnement interne.

Au sein d'un processeur, une **unité arithmétique et logique** (UAL, ou *ALU*) est un organe chargé de réaliser les calculs. Les processeurs modernes disposent de plusieurs UALs.

Les UALs élémentaires fournissent des opérations d'arithmétique entière et de logique (cf. TP).

D'autres UALs sont spécialisées pour l'arithmétique des nombres à virgule flottante (*FPU*).

Enfin, des UALs spécialisées ont été développées pour accélérer certains calculs (p. ex. les unités vectorielles)

Organisation d'un ordinateur

Quelques mots sur la hiérarchie de la mémoire

Le processeur

Le langage machine

Exécution des instructions

En pratique

Le langage machine

Pour exécuter **hello**, le processeur exécute une séquence d'instructions.

Une instruction = une opération + des opérandes.

La séquence est codée en binaire dans la section **.text** du fichier ELF contenant le programme, qui a été copiée en mémoire lors de l'étape précédente.

Ce qu'on appelle **langage machine** c'est à la fois :

- la liste des opérations et les opérandes qu'elles supportent,
- et la façon de les coder en binaire.

Il existe une syntaxe textuelle qu'on appelle assembleur (ou langage d'assemblage).

Quelques opérations que peut réaliser un processeur

Load

Écriture d'un mot de la mémoire vers un registre.

Store

Écrire d'un mot d'un registre vers la mémoire.

Add, Sub, And, Shift, ...

Faire une opération arithmétique ou logique dont la ou les opérandes (sources et cibles) sont des registres.

Branch (ou jump)

Sauter vers une instruction donnée, éventuellement sous certaines conditions.

Jeu d'instruction CISC vs. RISC

Remarque

ISA vs. microarchitecture La traduction en anglais de jeu d'instruction est instruction set architecture.

L'ISA est l'interface de programmation de la machine proposée aux utilisatrices et utilisateurs.

L'implémentation de l'ISA sous forme de circuits est appelée la microarchitecture du processeur.

ISA CISC : *Complex Instruction Set Computers*

- Beaucoup d'instructions
- Des instructions complexes/longues à exécuter

ISA RISC : *Reduced Instruction Set Computers*

- Nombre (volontairement) limité d'instructions (<100 à l'origine)
- Des instructions basiques/rapides à exécuter

Organisation d'un ordinateur

Quelques mots sur la hiérarchie de la mémoire

Le processeur

Le langage machine

Exécution des instructions

En pratique

Exécution d'une instruction

On décompose classiquement l'exécution d'une instruction en plusieurs étapes :

1. **Fetch** : l'instruction est lue depuis la mémoire
2. **Decode** : l'instruction est décodée
3. **Execute** : exécute l'instruction (par exemple en utilisant l'UAL)
4. **Mem** : si l'instruction est un accès mémoire, un second cycle est nécessaire
5. **Writeback** : le résultat de l'instruction est écrit dans les registres cibles

Pipeline, superscalaire, exécution dans le désordre

Les processeurs modernes utilisent un **pipeline** à plusieurs étages :

- chaque étage correspond à une étape de l'exécution d'une instruction., plusieurs instructions sont donc simultanément dans le pipeline.
- un processeur peut utiliser plusieurs pipeline en parallèle : on parle de processeur **superscalaire**.
- pour exploiter au mieux le pipeline, les processeurs peuvent exécuter les instructions « **dans le désordre** » (*Out-of-Order*). Pour cela, ils détectent les dépendances entre les instructions.
- pour ne pas être bloqué à attendre un résultat, les processeurs hautes performances intègrent des mécanismes d'**exécution spéculative**.

Organisation d'un ordinateur

Quelques mots sur la hiérarchie de la mémoire

Le processeur

Le langage machine

Exécution des instructions

En pratique

Source → Exécutable

On construit un fichier exécutable à partir du fichier source. Cela se fait avec une commande

```
go build
```

qui enchaîne plusieurs étapes :

- analyse lexicale
- pré-processing
- analyse syntaxique
- analyse sémantique
- génération de code et optimisations (→ code objet)
- édition des liens
- assemblage (→ exécutable)

Source → Code objet

On peut stopper la procédure à l'étape d'émission du code objet avec la commande suivante :

```
go tool compile main.go
```

On peut demander au compilateur d'émettre le code assembleur :

```
go tool compile -S -S main.go
```

ou

```
go build -gcflags '-S -S'
```

Code objet → code binaire

Remarque

La chaîne de développement Go utilise est un assembleur « portable ». Il est traduit vers l'assembleur de la machine cible lors de la phase d'édition des liens.

À partir du fichier objet, on peut obtenir le code exécutable en réalisation l'édition des liens :

```
go tool link main.o
```

Le code du *runtime* Go et des fonctions externes est automatiquement ajouté dans l'exécutable.

Le fichier de sortie est un binaire au format ELF (Executable and Linkable Format)

Lancer l'exécution du programme

Une fois le programme construit, on peut le lancer à l'aide de la commande :

```
./hello
```

qui enchaîne plusieurs étapes :

- création d'un nouveau processus
- initialisation de l'espace d'adressage du processus :
 - chargement du programme dans la mémoire de travail
 - chargement des données initialisées dans la mémoire de travail
- initialisation du pointeur d'instruction
- exécution du programme instruction par instruction

Observer l'exécution avec un debugger

Un debugger est un outil qui permet d'exécuter un programme pas-à-pas et d'observer le contenu de sa mémoire.

On utilisera GDB (cf. tp n°1), un outil libre et gratuit, largement utilisé, qui permet d'observer l'exécution à deux niveaux :

- au niveau du langage source
- au niveau ISA

Remarque

Pour observer l'exécution d'un programme au niveau de la microarchitecture, il faut passer par un émulateur ou un simulateur.