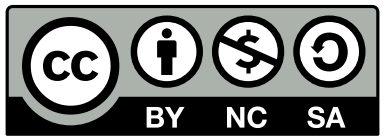
0001 - Introduction

à l'organisation des ordinateurs et à l'assembleur

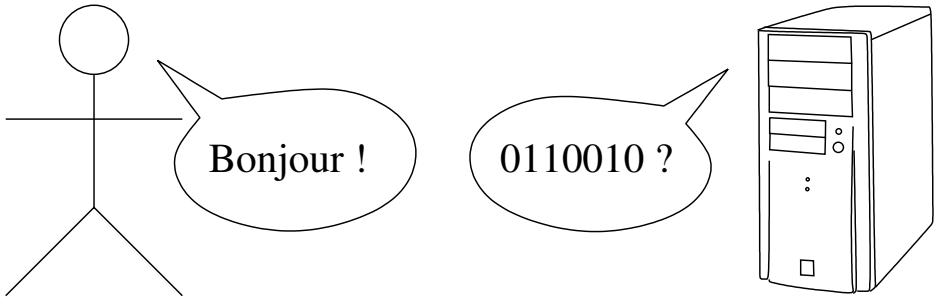
Jean Privat

Université du Québec à Montréal

INF2171 - Organisation des ordinateurs et assembleur
v241



Organisation des ordinateurs et assembleur



Objectifs du cours

- Comprendre comment un ordinateur fonctionne (en vrai)
  - Composantes bas niveau d'un ordinateur
  - Fonctionnement d'un processeur
  - Vocabulaire
- Apprendre à programmer directement un processeur principal
  - En vrai
  - Sans petites roues

Difficultés du cours

- C'est un cours de programmation

Plan

Objectifs du cours

Programmer en assembleur

Qui s'intéresse à l'assembleur ?

- Développeurs de systèmes d'exploitation
- Développeurs de pilotes
- Développeurs de compilateurs et machines virtuelles
- Développeurs de logiciels haute performance
- Experts en cybersécurité

Et les autres programmeurs ?

- Ne veulent pas entendre parler d'assembleur

Pourquoi ce cours alors ?

Organisation des ordinateurs

- Comprendre l'organisation (bas niveau) des ordinateurs
- L'« assembleur » n'est que 20% du titre

Et assembleur

- Assembleur: langage très très proche du langage de la machine
- Comprendre la machine du point de vue du programmeur
- L'assembleur sera donc omniprésent
- Car c'est notre instrument de compréhension

Oui mais pourquoi?

- *Tout* ce que fait l'ordinateur passe forcément par le langage de la machine
- Bonus: apprendre à programmer autrement

Réussir INF2171

Cours difficile?

- Technique
- Programmation
- Non naturel
- Un peu de maths (mais pas beaucoup)

Diapositives

- L'étoile ★ indique un concept **clé** du cours
- La loupe 🔍 indique un concept avancé

Comment échouer INF2171?

Prendre du retard

- Laisser les lacunes s'accumuler
- Ne pas poser de questions (en classe ou sur Mattermost)

Ignorer les laboratoires

- Ne pas se pratiquer
- Tout miser sur le talent

Faire les TP au dernier moment

- Ne pas respecter les consignes
- Ne pas valider les tests automatiques

Nouvelle mouture (cuisine interne)

Passage à RISC-V (plutôt que Pep/8)

- Pep/8: pédagogique, représentative, mais n'est plus maintenu
- RISC-V: vraie architecture moderne, et +/- pédagogique

Réécriture (et amélioration) du contenu

- Gros travail
- N'hésitez pas à signaler les problèmes

INF2171 version RISC-V

Différences majeures

- Vraie architecture
  - On peut exécuter nos programmes sur du silicium
- Plus complet et moderne d'un point de vu architectural
  - Les notions abordées du cours ont été bonifiées
- Plus de détails techniques accessoires
  - Corollaire de la *vrai vie*
  - On essaye de les rendre indolores
- Légère réorganisation du plan
  - Pour présenter les concepts de façon adaptée à RISC-V

Cours préalables et connexe

INF1120 Programmation I (préalable)

- Variables, données, calculs, structures de contrôles, sous-programmes, entrées-sorties, etc.

INF1132 Mathématiques pour l'informatique (extra)

- Algèbre de Boole, arithmétique

Corollaire

- Aucune connaissance en électronique ou en mathématique avancée n'est requise ou nécessaire

Cours suivants

Cours qui ont INF2171 en préalable (directement ou indirectement)

INF3270 Téléinformatique

- On a besoin de la représentation binaire de l'information

INF3173 Systèmes d'exploitation

- On va plus haut: gestion matérielle, supervision des processus, organisation des ressources

INF4170 Architecture

- On va plus bas: micro-architecture, pipelines, caches, etc.

Cours connexes

Apparaissent plus tard dans le cheminement type

INF3135 Construction et maintenance

- Comprendre l'assembleur aide à mieux comprendre le C

INF3105 Structures de données et algorithmes

- Comprendre l'assembleur aide à mieux comprendre le C++

Corollaire

- La connaissance du C, C++ ou d'un autre langage compilé n'est ni requise, ni nécessaire

Cours au choix connexes

INF5171 Programmation concurrente et parallèle

- Programmation haute performance et mesures de performance

INF600C cybersécurité applicative

- Rétroingénierie et exploitation binaire

INF600E création de langages informatiques

- Génération (éventuelle) de code machine

Cours de cycles supérieurs

INF7641 Compilation

- Génération et optimisation de code machine

INF7741 Machines virtuelles

- Gestion de la mémoire, parallélisme, optimisations dynamiques

3 cours au choix de bac, 2 de maîtrise

- INF2171 est un cours de première année universitaire
- Mais ouvre sur des domaines passionnants de l'informatique

INF2171

Les objectifs du cours

- Sont les concepts d'**organisation des ordinateurs**: architecture, processeur, registre, bit, octet, codage, mémoire, pile, tas, instruction machine, entrées-sorties, etc.
- L'**assembleur** est un moyen
- **RISC-V** est un exemple de technologie
- Mais on va les apprendre quand même

Important dans la formation en informatique

- Préalable à plusieurs cours
- Va vous aider pour les cours INF3135 (C) et INF3105 (C++)

Essentiel dans plusieurs sous-domaines

- Cybersécurité, programmation haute-performance, compilation, machine virtuelle, système d'exploitation, etc.

Organisation des ordinateurs

Organisation des ordinateurs

Niveaux d'abstraction typiques d'un ordinateur

Du plus **abstrait** au plus **concret**

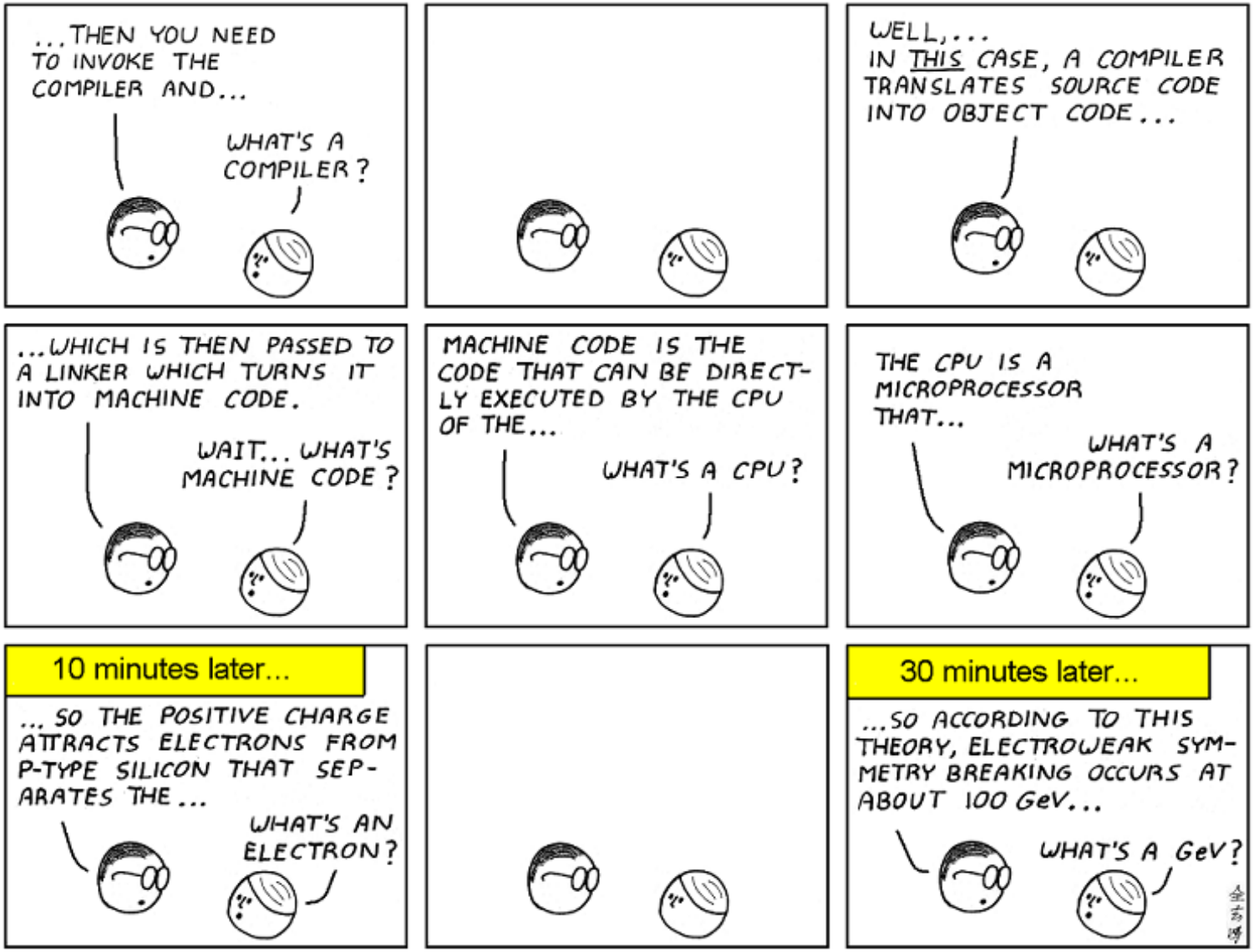
- Applications
- Langages de haut niveau
- Système d'exploitation
- Assembleur
- Langage machine
- Micro-architecture
- Circuits logiques

Organisation des ordinateurs

Niveaux d'abstraction typiques d'un ordinateur

Du plus **abstrait** au plus **concret**

- Applications (INF1120...)
- Langages de haut niveau (INF1120...)
- Système d'exploitation (INF3173)
- Assembleur (INF2171)
- Langage machine (INF2171)
- Micro-architecture (INF4170)
- Circuits logiques (INF4170)



Source: http://abstrusegoose.com/98

Organisation des ordinateurs

Principes de base

- Un ordinateur est une machine
- Tout n'est que bits

Sera répété plusieurs fois au cours de la session

Historique (très simplifiée)

Cailloux et abaques (-5000)

- « Calcul » et « caillou » ont la même étymologie
- Abaques = outils (bouliers, tablettes, etc.)

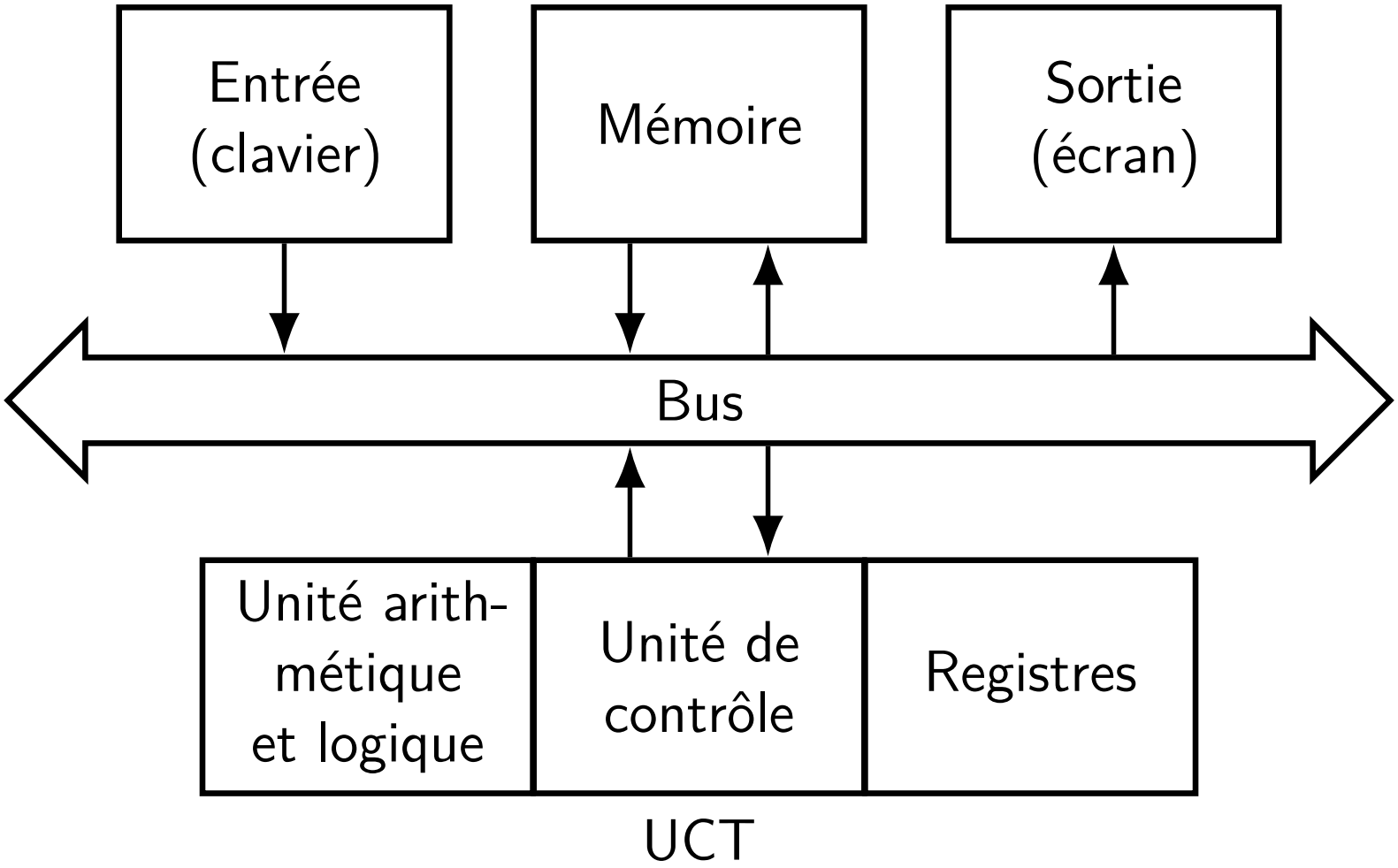
Machines mécaniques

- Pascaline (1642) : une calculatrice
- Machine analytique (1830) : programmable

Premiers vrais ordinateurs

- MARK I (1937-1944) : électromécanique
- ENIAC (1943-1946) : électronique

Architecture de von Neumann



Architecture de von Neumann

- Média d'entrée
  $\rightarrow$ Quantité virtuellement illimité de données
- Une unité d'emmagasinage commune
  $\rightarrow$ Représentation interne uniforme (bits)
  $\rightarrow$ Autant pour les instructions que les données
- Une unité de calcul
  $\rightarrow$ Arithmétique et logique + - * / & |...
- Média de sortie
  $\rightarrow$ Quantité virtuellement illimité de résultats
- Une unité de contrôle
  $\rightarrow$ Interprète les instructions

Mémoire

Élément central

- Emmagasine programmes et données
- Ensemble fini de **cellules** (des **octets**)

Chaque cellule

- A une **adresse**
- A un nombre fixe (8) de **bits** (0 ou 1)
- A **une** et **une seule** valeur à la fois
- Est accessible très rapidement (nano seconde)

Tout n'est que bits

- Une cellule mémoire ne contient que des bits (des 0 et et 1)
- Toute information traitée par un ordinateur doit être codée sous forme binaire
  ... Y compris les instructions de la machine

RAM/ROM

Mémoire vive (*random access memory* RAM)

- Stocke les programmes et les données
- Accessible en lecture et écriture
- Accès direct à l'information (*random*)
- Volatile (alimentation en continue nécessaire)
- Différents types: SRAM, DRAM, SDRAM, DDR SDRAM...

Mémoire morte (*read-only memory* ROM)

- Contient de quoi amorcer l'ordinateur
- Contient le code des primitives basiques d'entrées-sorties (BIOS/UEFI)
- Accessible en lecture seulement
  Éventuellement en écriture par des moyens détournés (flashage)
- Accès direct à l'information (*random* pas exclusif à la RAM)
- Persistante (non volatile)
- Différents types: ROM, PROM, EEPROM...

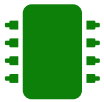
Unités d'entrée-sortie

Communiquer avec l'extérieur

- Humains et environnement
- On parle communément de périphériques

Exemples

- Clavier
- Écran
- Disque dur
- Haut-parleur



Unité centrale de traitement (UCT)

- *Central processing unit* (CPU)
- Processeur (principal)

Unité de calcul

- Exécute les opérations logiques et arithmétiques
- Addition, soustraction, ET binaire, etc.

Unité de contrôle

- Lit les instructions en mémoire
- Lit les données en mémoire (ou d'un périphérique)
- Fournit les opérandes à l'unité de calcul
- Récupère les résultats de l'unité de calcul
- Écrit les résultats en mémoire (ou vers un périphérique)

Bus

Définition

- Du latin *omnibus*: « pour tous »
- Medium de communication entre les composantes
- Ensemble de lignes de communication

Les bus sont spécialisés

- Données $\rightarrow$ bus de donnée
- Adresses $\rightarrow$ bus d'adresse
- Signaux $\rightarrow$ bus de contrôle

Registres

Mémoire très locale

- Cellules mémoire de l'UCT
- Accès plus rapide que la mémoire principale
  (à pied, pas besoin de prendre les transports en commun)

Contenu

- Opérandes, résultats, configuration et états particuliers, résultats temporaires, etc.
- Sert aussi au contrôle de l'UCT

Horloge

Synchronise les activités

- Engendre un signal régulier (GHz)
- Les activités de l'ordinateur sont synchronisées
- Vitesse limitée par la technologie courante

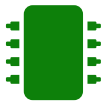
Cycle d'horloge

- La plus petite unité de temps
- Une instruction peut nécessiter plusieurs cycles

Exemple

`cat /proc/cpuinfo`

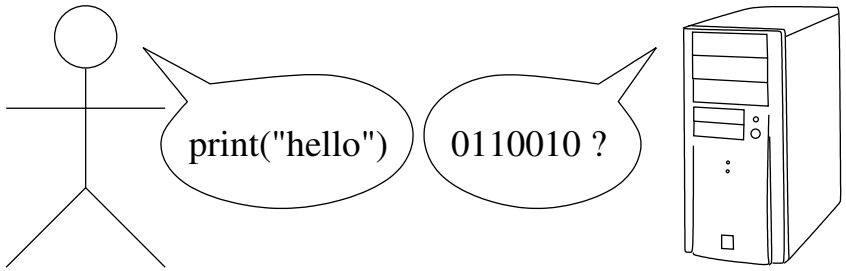
Montre des informations sur l'UCT, dont la fréquence



Architecture

Comment programmer un microprocesseur?
Java ? Python ? C ?

- Quel langage comprend l'UCT



Langage machine !

- Le seul langage compris par l'UCT
- Propre à chaque **architecture**

Programme natif

- **natif** = écrit dans langage de la machine qui l'exécute
- Ou **traduit** ou **compilé** vers ce langage machine (C, C++, assembleur, etc.)
- Comment exécuter un programme « non natif » alors ?
  $\rightarrow$ on va y venir

Architecture

Abstraction de l'UCT **POUR** les programmeurs

- *Instruction Set Architecture* (ISA)
- Le langage binaire de l'UCT

Indique les règles de programmation

- Jeu d'instructions
- Registres (spécificité, rôles)
- Types des données
- Fonctionnalités: adressages, interruptions...

Attention: « Architecture » vs. « Micro-architecture »

- Architecture: **spécification** pour **utiliser** l'UCT
- Micro-architecture : **implantation** particulière d'une architecture pour un processeur donné (détails internes)

Taille d'architecture

8 bits, 16 bits, 32 bits, 64 bits, 128 bits, etc.

- Taille des registres généraux (entiers)
- et/ou largeur du bus d'adresse
- et/ou largeur du bus de donnée
- et/ou considérations marketing

Impact

- 👍 Traitement efficace de données et programmes plus volumineux
- 👎 Taille d'architecture $\sim$ taille et complexité des circuits $\sim$ prix

Compatibilité

- Pas de compatibilité entre les différentes tailles (en général)
- Mais se ressemble suffisamment pour déjouer notre vigilance

Exemples d'architectures

x86 Intel 16b (1978), 32b (1985); AMD 64b (2003)

- Intel® 64 and IA-32 Architectures Software Developer's Manual (2023) 5066 pages
- AMD64 Architecture Programmer's Manual (2023) 3336 pages
- Domine PC, portables et serveurs. 350M ventes par an

ARM 32b (1985), 64b (2011)

- ARM A32/T32 Instruction Set Architecture® Armv8, for Armv8-A architecture profile (2021) 1933 pages
- Domine systèmes embarqués et téléphones. 1.5G ventes par an.
- Apple M1 et M2 sont des ARM

RISC-V 32b, 64b, 128b (2015)

- RISC-V Instruction Set Manual Vol. I: Unprivileged ISA (20191213) & Vol. II: Privileged Architecture (20211203) 393 pages

Compteur ordinal et mémoire

- L'UCT exécute un programme (en langage machine)
- Or ce programme (machine) est en mémoire
- Donc l'UCT sait où est l'instruction courante

Compteur ordinal

- *Program counter* (PC)
- Registre dédié de l'UCT
- Contient l'adresse en mémoire de l'instruction courante

Boucle d'exécution de l'UCT

L'UCT ne s'arrête jamais

- Exécute continuellement les instructions

1 pas d'exécution = 4 étapes (simplifiés)

- Charger le contenu de la mémoire dont l'adresse se trouve dans le compteur ordinal
  Accès mémoire implicite. *instruction fetch*
- Décoder l'instruction obtenue
  Extraire le code de l'opération, les opérandes et les détails
- Exécuter les opérations indiquées par l'instruction
- Augmenter le compteur ordinal (à l'instruction suivante)

Note: certain processeurs (x86, Pep/8) incrémentent le compteur ordinal avant l'exécution.

- Démo https://eseo-tech.github.io/emulsiV/

Instructions et données

Attention : pas de distinction pour l'UCT

- L'UCT travaille en aveugle (tout n'est que des bits)
- L'UCT ne distingue pas:
  - un tas de bits qui correspond à une instruction du programme
  - un tas de bits qui correspond à une donnée
  - un tas de bits non utilisé/initialisé
- Le programmeur est **responsable**

Erreur habituelle

- Essayer de faire exécuter des trucs en mémoire qui ne sont pas des instructions du programme

Architecture RISC-V

- Architecture RISC (*Reduced Instruction Set Computer*)
  Par opposition aux architectures CISC (on y reviendra)
- Développé initialement à l'université Berkeley (Californie)
- Spécification libre (*Creative Commons*) sans redevance
- Marché en croissance. Systèmes embarqués. Recherche universitaire (micro-architecture). Hobbyistes
- Nombreuses ressources externes disponibles
  (qualité et pertinence variable)
- Plusieurs fabricants, micro-architectures, simulateurs et émulateurs

Familles d'ISA RISC-V

- Fonctionnalités à la carte

Exemples

- `RV64I` ISA de base 64 bits avec arithmétique simple
- `RV32E` ISA de base 32 bits avec arithmétique simple, mais moins de registres
- `RV64IM`: `RV64I` + extension "M" multiplications et divisions entières
- `RV32IF`: `RV32I` + extension "F" arithmétique flottante simple précision

RV32I

Architecture 32 bits (4 octets) de base

- 32 registres généraux de 32 bits (`x0` à `x31`).
  - `x0` est particulier: contient toujours zéro, ignore les écritures
  - Les registres ont aussi des noms d'ABI (une autre fois...)
- Un registre spécial `pc` de 32 bits (compteur ordinal)
- 6 registres d'états (compteurs matériels)
- Instructions sur 32 bits, alignés (parfois 16 si extension C)
- Une quarantaine d'instructions

RV64I

Architecture 64 bits (8 octets) de base

- 32 registres généraux de **64** bits (`x0` à `x31`).
- Un registre spécial `pc` de **64** bits (compteur ordinal)
- 6 registres d'états (compteurs matériels)
- Instructions sur 32 bits, alignés (parfois 16 si extension C)
- Une quarantaine d'instructions

C'est principalement la taille des registres qui changent

Notre cible: RV64

- Les ordinateurs intéressants RISC-V sont en RV64
- Les grandes distributions Linux ne supportent pas toujours RV32
- Toutefois, certains simulateurs pédagogiques gèrent pas (ou mal) le 64 bits

Environnements d'exécution

Environnements d'exécution

- Comment les programmes machines *arrivent* en mémoire? À quelles adresses?
- Comment le compteur ordinal et autres registres sont initialisés? À quelles valeurs?
- Comment réaliser des entrées-sorties? Sur quels périphériques?
- Comment sont gérées les exceptions et interruptions? Qui les traite?

Les architectures spécifient des règles du jeu, mais les détails varient (grandement) d'un environnement à l'autre.

Exemples d'environnements d'exécution

- Système embarqué simple
- Système d'exploitation multitâche (Linux, Windows)
- Simulateur et émulateur

Système embarqué

- Exemples: *system on a chip* (SoC), petits appareils, etc.
- Programme +/- en dur en ROM
- Valeurs des registres préinitialisés
- Entrées-sorties spécifiques et connues d'avance (adresses statiques)
- Exceptions et interruptions traitées directement par le programme

Systèmes d'exploitation multitâche (INF3173)

- Exemples: PC, téléphone mobile, super-calculateur, etc.

Le système d'exploitation

- Charge dynamiquement (et au besoin) les programmes depuis le disque
- Configure les registres pour chaque programme
- Contrôle les entrées-sorties (via appels systèmes)
- Gère les exceptions et interruptions
- Les détails en INF3173

Simulateur et émulateur

Note linguistique

- Simulation = pour l'analyse et l'étude (on modélise)
- Émulation = pour l'utilisation et la substitution (on réplique)

Granulatité

- Simulation d'un environnement minimal (*bare bone*)
  pour un programme unique invité
- Simulation d'un système d'exploitation (*user mode*)
  pour un programme unique invité
- Simulation totale (*full-system*)
  pour un système d'exploitation invité
  et pour tous ses programmes

Simulateurs et émulateurs

RARS (pédagogique)

- *RISC-V Assembler and Runtime Simulator*
- Écrit en Java/Swing, licence MIT, dépot github
- Le fork du cours corrige des bogues
  (utilisez la version originale à vos risques et périls)
- Simple et fonctionnel
- Cochez *Settings>64 bits*

emulsiV (pédagogique)

- *Visual simulator for a simple RISC processor called Virgule*
- Écrit en JavaScript, licence MPL, dépot github
- Démo en ligne
- Vue micro-architecture simple

Simulateurs et émulateurs

QEMU (industriel)

- Quick Emulator
- Écrit en C, licence GPL2, dépot gitlab
- `apt install qemu-user` (chez Debian et autres)
- Émulation mode utilisateur et système entier

Ripes (pédagogique)

- *Visual Computer Architecture Simulator*
- Écrit en C++/Qt, licence MIT, dépot github
- D'avantage pertinent pour un cours d'architecture

Assembleur et programmation

Comment programmer du code machine?

Écrire le code machine à la main

- Langage binaire
- Codage complexe des instructions
- Éditeur binaire/hexadécimal, carte perforée, clavier binaire, etc.

Lire et écrire du code machine?

« ...there's way too much information to decode the Matrix. You get used to it, though. Your brain does the translating. I don't even see the code. All I see is blonde, brunette, redhead. »

— Cypher, Matrix (1999)

Langage d'assemblage

- Forme symbolique d'un langage machine
- Destiné aux humains
- Traduction (quasi-)littérale vers du code machine

```
# Fonction Fibonacci
fib:
  addi  sp, sp, -32
  sd    ra, 24(sp)
  sd    s0, 16(sp)
  # ...
```

Syntaxes spécifiques

- Tous se ressemblent, mais pas de syntaxe universelle
- Chaque outil et/ou architecture a sa propre syntaxe ou ses propres variations +/- compatibles

Exemples de syntaxes

RISC-V

```
ld a0, 8(a1)
```

ARM64

```
ldr x0, [x1, 8]
```

x86-64 syntaxe Intel

```
mov rax, QWORD PTR [rsi+8]
```

x86-64 syntaxe AT&T

```
movq 8(%rsi), %rax
```

- **Note**: Syntaxe RISC-V officielle: https://github.com/riscv-non-isa/riscv-asm-manual

Caractéristiques syntaxiques classiques

- Format rigide des lignes de code
  - ligne ~ instruction assembleur ~ instruction machine
  - ligne ~ directive ~ traitement spécifique par l'outil
- Noms des instructions courts (**mnémoniques**)
- Registres nommés: `a0`, `x0`, `rax` (utilisation standard)
- Valeurs littérales: `42`, `"hello"` (décimaux, chaînes, etc.)
- Symboles et étiquettes: `foo`
- Commentaires # (ou parfois ;)

On y reviendra...

Assembleur vs. code machine

- Pseudo-instructions (beaucoup en RISC-V)
- Directives (commencent par un point)
  - Données initiales, globales, constantes
  - Organisation de la mémoire
  - Indications pour empaquetage et le chargement (exécutable, bibliothèque)
  - Information de débogage
  - Etc.

Assembleur

Langage de programmation impératif

- Séquences d'instructions
- Boucles
- Structures conditionnelles
- Appels de sous-programmes
- ... Bref, rien de nouveau

Langage sans petites roues

- pas/peu de contrôle automatique
  pas de type statiques (ni dynamiques), pas de signature de méthode, etc.
- pas/peu d'aide (avertissements, complètements, etc.)

Assemblage et outillage

Assemblage et outillage

Assembleur intégré

- Dans un simulateur avec un éditeur de texte
- Clé en main mais spécifique et limité
- Format de sortie optionnel
  Souvent directement chargé en mémoire simulé par l'outil
- Exemple: RARS

Outil externe dédié

- Autonome ou dans une suite d'outils
- Fait pour la *vraie vie*
- Format de fichier sortie spécifique (ELF ou autre)
- Nécessite une édition de liens (on y revient)
- Exemple: GNU assembler

La variété des environnements d'exécution 🌍 complique l'affaire

GNU binutils

- Outils binaires bas niveau de GNU
- Écrit en C, licence GPL2
- Pour RISC-V: `apt install binutils-riscv64-linux-gnu` (chez Debian et autres)

GNU assembler

- Famille d'assembleurs, génère des fichiers objets Unix ELF (*Executable and Linkable Format*)
- Commande `as` (native) ou `riscv64-linux-gnu-as` (croisé)
- Documentation, 448 pages
- `as hello.s -o hello.o`

GNU ld

- Combine des fichier objets (`.o`) en un exécutable (ou autre)
- Commande `ld` (native) ou `riscv64-linux-gnu-ld` (croisée)
- `ld hello.o -o hello`

Désassemblage

Traduction instruction machine → assembleur

- Raisonnable
- Perte possible de pseudoinstructions et de formes littérales
- Le plupart des simulateurs interactifs en sont capable

Traduction programme machine → assembleur

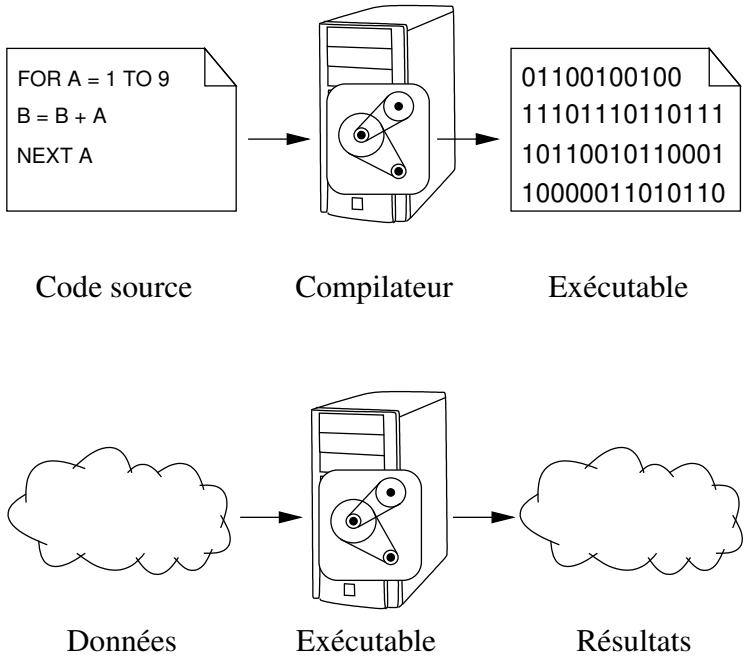
- Plus difficile
- Où sont les instructions et les données ?
- Perte des commentaires et étiquettes (début de fonction)
- Détails en INF600C

objdump -d

`objdump -d` (GNU) désassemble un binaire en code machine

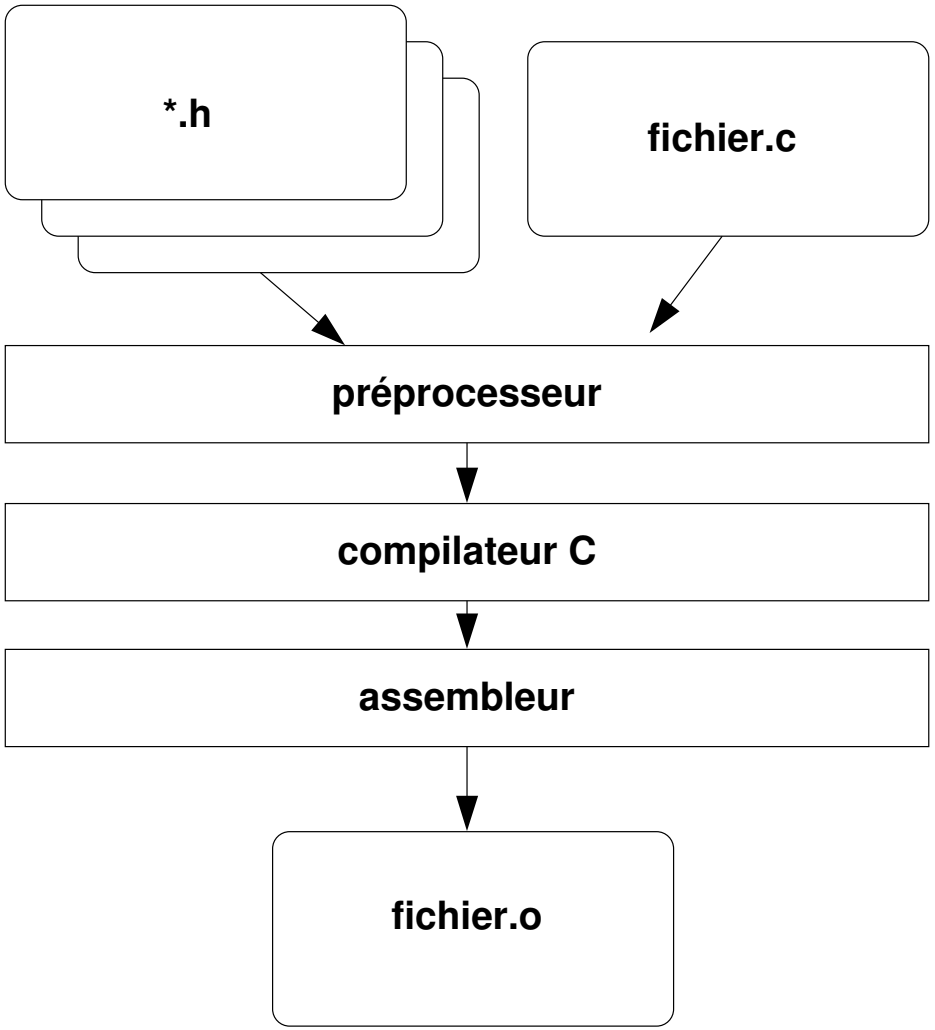
Compilation

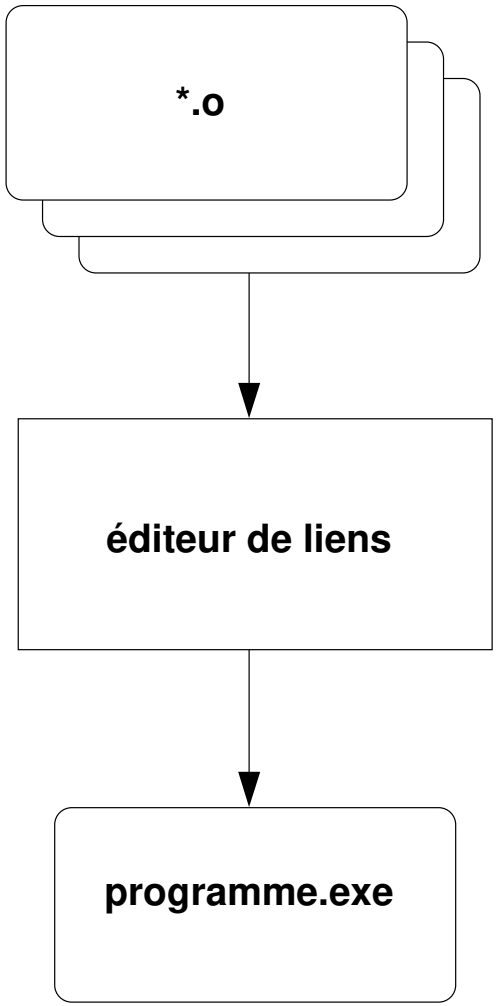
- **Transforme** un programme en un langage de haut niveau vers un programme **équivalent** en langage machine
- Exemples: compilateurs C, C++, Rust, Go, Haskell, etc.
- Tâche complexe, détails en INF600E et INF7641



Note: le compilateur est un programme comme les autres

Schéma de compilation classique en C





GCC (GNU Compiler Collection)

Suite de compilateurs

- C, C++, fortran, go, etc.
- Pour RISC-V: `apt install gcc-riscv64-linux-gnu` (chez Debian et autres)
- Commande `gcc` (native) ou `riscv64-linux-gnu-gcc` (croisée)
- Basé sur `bintools` pour les détails bas niveau
  - Utile comme *frontend* pour `as` et `ld` (surtout `ld`)
  - `gcc hello.s -static -nostdlib -o hello`

Compiler explorer

Permet l'expérimentation interactive

- Plusieurs compilateurs (et versions)
- Plusieurs architectures
- Plusieurs options
- → https://godbolt.org/

Options de compilation GCC

- Plus de 3000 options: langage source, langage cible, diagnostiques, optimisations, assemblage, édition de liens, génération de code, machine cible, etc.
- Liste des options de GCC

Options pertinentes

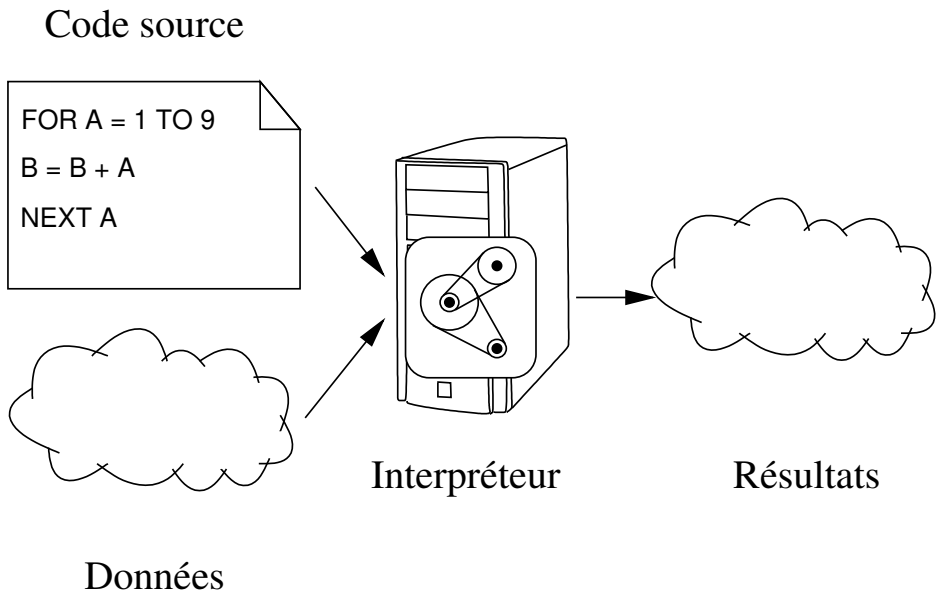
- Niveau d'optimisation : `-O`
  - `-O0`, `-O1`, `-O2`, `-O3`
- Architecture cible `-march` (ISA)
  - `-march=rv32i`, `-march=rv64ima`
- Micro-architecture cible: `-mtune` (modèle d'UCT)
- ABI: `-mabi` (*aplication binary interface*)
  - Pour s'intégrer à des langages de haut niveau, des bibliothèques et des systèmes d'exploitation
  - On y reviendra peut-être...
- Liste des options de GCC pour RISC-V

Décompilation

- Transformer un code machine en un programme équivalent dans un langage de haut niveau
- Très difficile, on en discute en INF600C

Interprétation

- Un programme exécute le programme plutôt que directement le processeur
- Exemples: interpréteur Python, Ruby, Bash, etc.
- Détails en INF600E et INF7741



Note: l'interpréteur est un programme comme les autres

Machine virtuelle + JIT

Compilateur juste-à-temps, (*just-in-time*, JIT)

- Aussi appelé *dynamic translation*
- Interpréteur spécialisé qui **génère du code machine à la volée**
- Exemples: machines virtuelles Java, C#, JavaScript ; QEMU...
- Très sophistiqué, détails en INF7741

Note: la machine virtuelle est un programme comme les autres

Conclusion

Résumé

Les entités en œuvre

- L'architecture cible (et son langage machine)
- Le langage d'assemblage
- L'outil d'assemblage
- L'environnement d'exécution (processeur ou simulateur)
- Le programmeur

Difficultés

- Compatibilité entre tous ces éléments

La prochaine fois

Arithmétique

- Nombres entiers
- Additions et soustractions
- Nos premiers programmes assembleur RISC-V