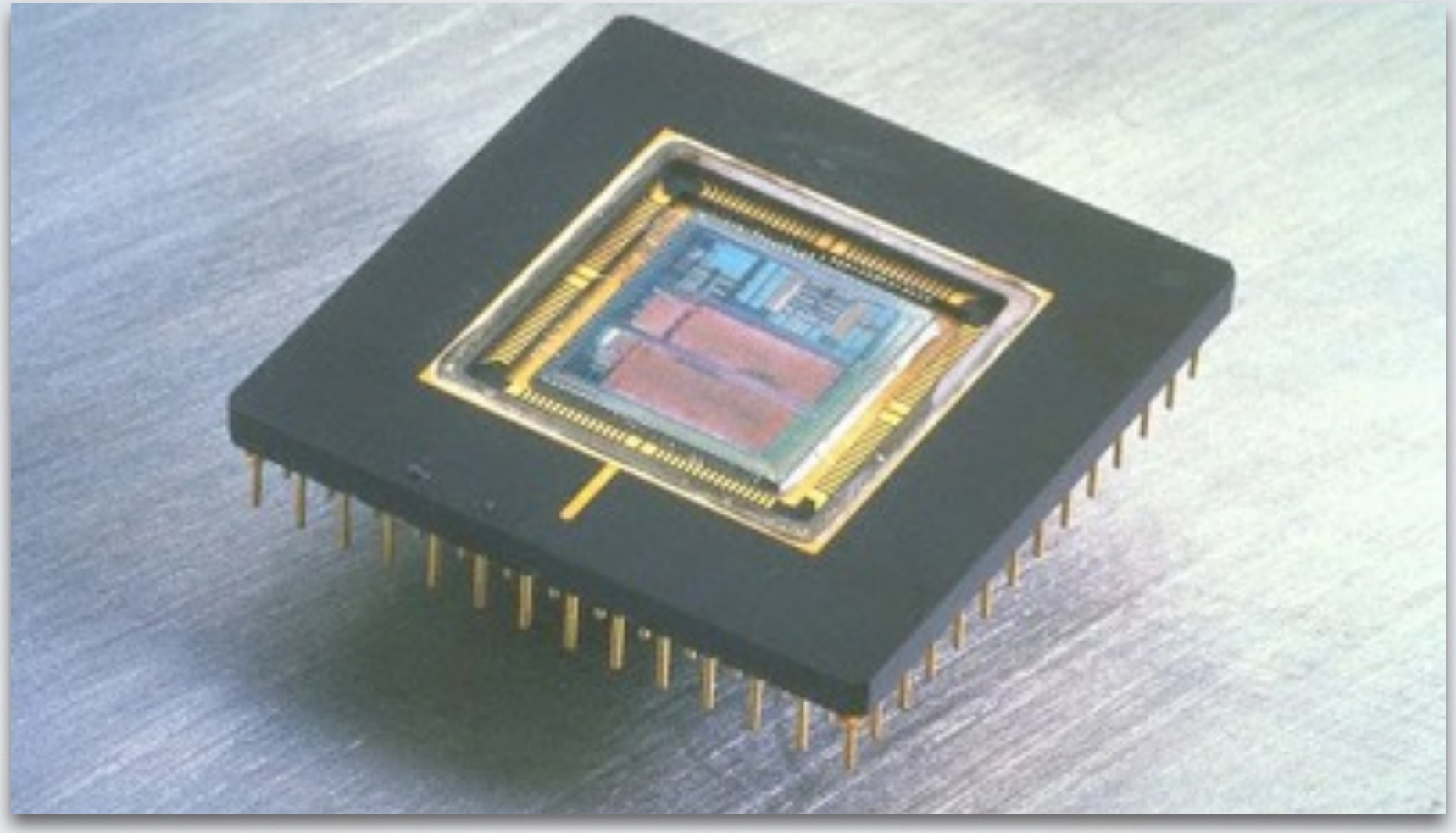
Polytech Informatique 3
ALM :
Architectures Logicielles et Matérielles



Polytech Informatique 3

ALM : Architectures logicielles et Matérielles

- Equipe enseignante: (prenom.nom@univ-grenoble-alpes.fr)
  - Ghada Moualla
  - Olivier Richard
  - Pascal Sicard

ALM: BUT DU JEU

- Comprendre le fonctionnement d'un ordinateur d'un point de vue matériel
- Du coup comprendre comment s'exécute un programme sur cette machine
- Primordial pour un informaticien dont le principal outil est l'ordinateur
- Permet de développer des programmes (le soft) en toute connaissance de l'aspect matériel de la machine (le hard)

ALM: BUT DU JEU

- Notions indispensables à tout informaticien polyvalent, en particulier quand on «touche»
  - au système d'exploitation
  - aux périphériques (écran, clavier, …) et entrée/sortie d'un ordinateur
  - programmation «haute performance» (contrainte mémoire, temps réel ...)
- Bases de la conception de circuits ….
- 2ème semestre : ALM2:
  - Processeur et son entourage, les entrées/sorties,
  - Les bases du système d'exploitation
  - Programmation sur système embarqué à base de processeur ARM

ORGANISATION

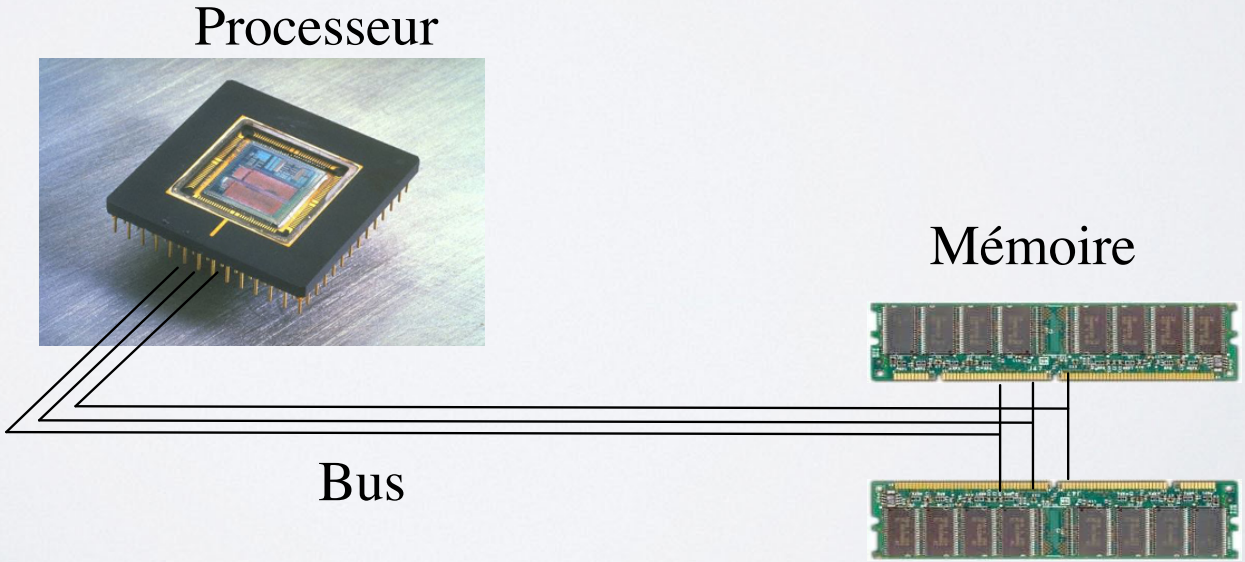
- On commence par le soft
  - Programmation particulière dans un langage « proche » du matériel: le **langage d'assemblage**
  - On apprends au passage les bases du **langage C** et on comprends comment il peut être traduit en langage d'assemblage
- On étudie ensuite le matériel nécessaire à l'exécution des programmes:
  - du transistor au processeur, à la mémoire, …

ARCHITECTURE MATÉRIELLE DE L'ORDINATEUR

- Un ordinateur:
  - Des circuits permettant d'effectuer des séries de calculs
  - On peut changer à volonté ces séries de calculs que l'on appelle des programmes
  - Entre chaque calcul, on peut mémoriser les résultats intermédiaires et s'en resservir
  - On a besoin
    - d'un centre de calcul (le **processeur**)
    - de mémoire pour stocker les valeurs intermédiaires (La **mémoire vive:** la RAM…)

ARCHITECTURE MATÉRIELLE DE L'ORDINATEUR

- Connexions entre le processeur et la mémoire
  - Des nappes de fils (appelés bus)
  - Chaque fil peut être soumis à deux tensions électriques



PRINCIPE DE FONCTIONNEMENT

- Le processeur exécute une suite d'instructions permettant de faire des calculs sur des **valeurs** pouvant se trouver en mémoire
- Il faut que le processeur sache où sont stockées ces valeurs: les emplacements en mémoire sont référencés par des **adresses**
- Il faut que le processeur puisse charger (lire) ou stocker (écrire) des valeurs en mémoire

OÙ SONT LES PROGRAMMES ?

- Les programmes (suite d'instructions) sont stockés aussi dans la mémoire !
- Le processeur doit donc pour exécuter un programme:
  - Lire les unes après les autres les instructions du programme en mémoire
  - Pour chaque instruction:
    - Comprendre puis exécuter l'instruction
      - Lire une valeur en mémoire
      - Faire un calcul
      - Ecrire une valeur en mémoire

MÉTAPHORE DU CUISINIER

- Les « **valeurs** » dans la mémoire
  - Ce sont les produits de base, stockés dans des boîtes (**cases mémoires**) sur des étagères (**mémoire**)
- Le processeur
  - C'est le cuisinier
- Le programme
  - C'est la recette
- Le cuisinier exécute la recette

  - Chaque instruction manipule certains produits, les transforme, et les réserve
  - Les boîtes contenant les produits portent des numéros permettant de les retrouver (les **adresses**)
  - Le livre de recette est aussi sur les étagères, le cuisinier récupère sur les étagères chaque instruction, puis la comprends, puis l'exécute

VALEURS ÉLÉMENTAIRES

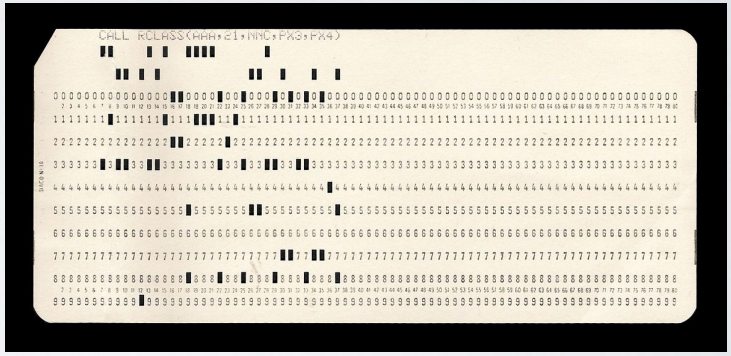
- Dans un ordinateur les seules valeurs possibles sont 0 et 1 (2 tensions électriques dans les circuits)
- Toute information (entier, caractère, couleur, son, image, …) manipulée par un ordinateur est donc codée à l'aide d'une suite de 0 et de 1 (des **bits**)
- C'est aussi vrai pour les programmes…
- La question majeure est donc:
    - Quelle interprétation donner à une suite de 1 et de 0 ? Est ce une couleur, un caractère, un entier… et avec quelle codage cette information est elle représentée ?
    - 10001001 peut représenter le caractère A, le nombre 137, la couleur noire ?
- La plupart du temps les **bits** sont regroupés en paquet de 8 : des **octets** (ou byte)

UN PROGRAMME EXÉCUTÉ PAR UN PROCESSEUR

- Un programme est donc une suite d'instructions que comprend le processeur
- Chaque instruction est composée de 0 et de 1
- Exemple d'une instruction: 10000100010000111111001000001
- C'est le **seul langage** de programmation que comprend le processeur: on parle de **langage machine**
- Chaque processeur possède son propre langage machine !
- Un programme en langage machine s'appelle un **exécutable**

LANGAGE D'ASSEMBLAGE

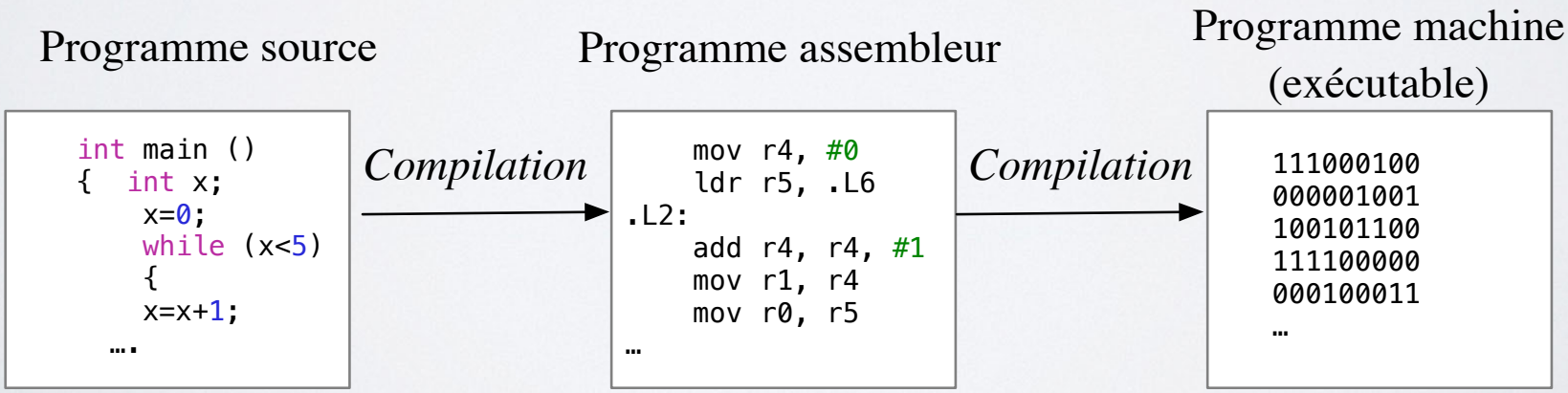
- Au début les informaticiens écrivaient les programmes en langage machine. Dur dur !



- On invente un langage plus compréhensible par l'humain: le **langage d'assemblage (ou assembleur)**
- Une instruction assembleur = une instruction machine
- Exemple: **add r1, r2, #3** = 11101001110001001001100

LANGAGE DE HAUT NIVEAU

- On invente des langages plus évolués permettant de programmer plus facilement (ADA, JAVA, C, PYTHON…)
- On élabore des compilateurs permettant de traduire ces langages de plus haut niveau en langage d'assemblage puis en langage machine
-



REPRESENTATION/CODAGE DE L'INFORMATION

- Les entiers naturels sont codés en **base 2**
  - 13 en base 10 sur un octet : 0000 1101 en base 2
  - Poids faible à droite comme habituellement pour la base 10
  - Sur n bits on représente les entiers naturels de 0 à 2^n-1
    - Exemple sur un octet: 0 à 255

REPRESENTATION

- Les entiers relatifs (***signés***) sont représentés dans un codage appelé le **complément à 2**
- Ce codage à l'avantage de pouvoir utiliser les opérations arithmétiques classiques de la base 2 (addition, soustraction…)
- Voir sur wikipedia l'article sur le complément à 2

REPRESENTATION

- Les caractères sont codés sur un octet à l'aide du codage **ASCII** (American Standard Code for Information Interchange) sur 7 bits
- 1 bit pour les accents, non normalisé: problème classique du passage des caractères accentués entre différents types de d'ordinateur
- Des codes normalisés sont aussi utilisés pour les caractères non latin (par exemple les katakana japonais sont codés sur 3 octets 身)
- On représente aussi des réels, des couleurs, du son, des images….

REPRESENTATION HEXADÉCIMAL

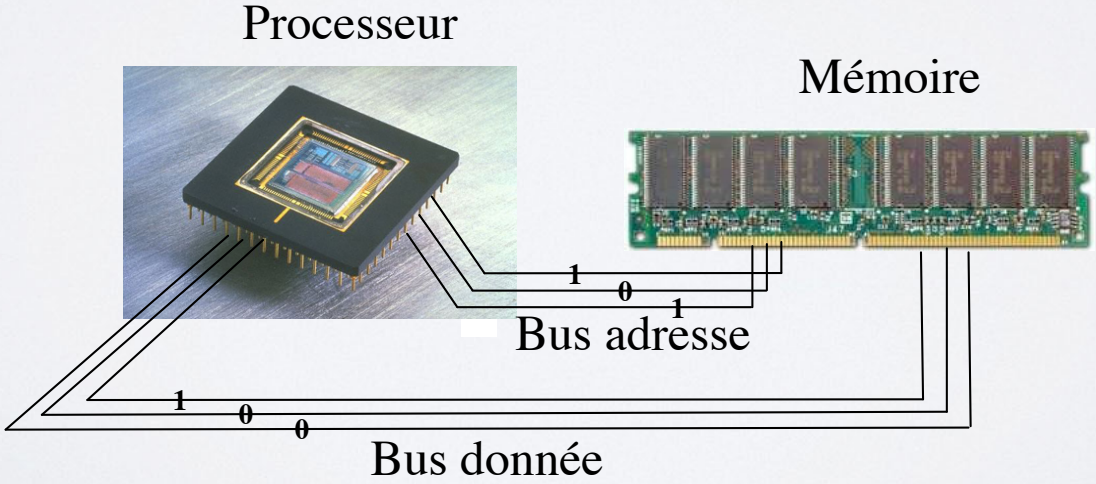
- On utilise plutôt la base 16 (**hexadécimal**) pour écrire les suites de 1 et de 0 (difficile à lire)
- En effet 4 bits peuvent être représentés par un des 16 chiffres hexadécimaux (0,1, 2,….9, A, B, C, D, E, F)
- Exemple
  - 1000 1111 s'écrit en hexadécimal 8F
  - 0001 1010 s'écrit en hexadécimal 1A
- On met souvent **0x** devant la valeur pour se rappeler que c'est de l'hexadécimal

LE STOCKAGE EN MEMOIRE

- Les principes de fonctionnement qui suivent sont propres au processeur que l'on va étudier (**ARM**), ce ne sont pas des généralités
- En mémoire **chaque octet** est référencé par un numéro unique: son **adresse**
- Les adresses sont des entiers naturels qui sont codées en **base 2**

LE STOCKAGE EN MEMOIRE

- Deux bus permettent l'accès à la mémoire:
  - le bus adresse sur lequel le processeur met l'adresse à laquelle il veut lire ou écrire
  - le bus donnée sur lequel le processeur récupère ou envoie la valeur lue ou écrite en mémoire
  -



LE STOCKAGE EN MEMOIRE

- La taille de la mémoire dépend du nombre de fil du bus adresse
  - Exemple 32 fils= 2^{32} adresses donc 4 giga octets adressables possible
- Le bus donnée peut avoir différentes tailles, souvent 32 (machines 32 bits)
- L'accès à la mémoire peut se faire en une seule instruction pour un seul octet, 2 octets ou 4 octets consécutifs

A L'INTÉRIEUR DU PROCESSEUR

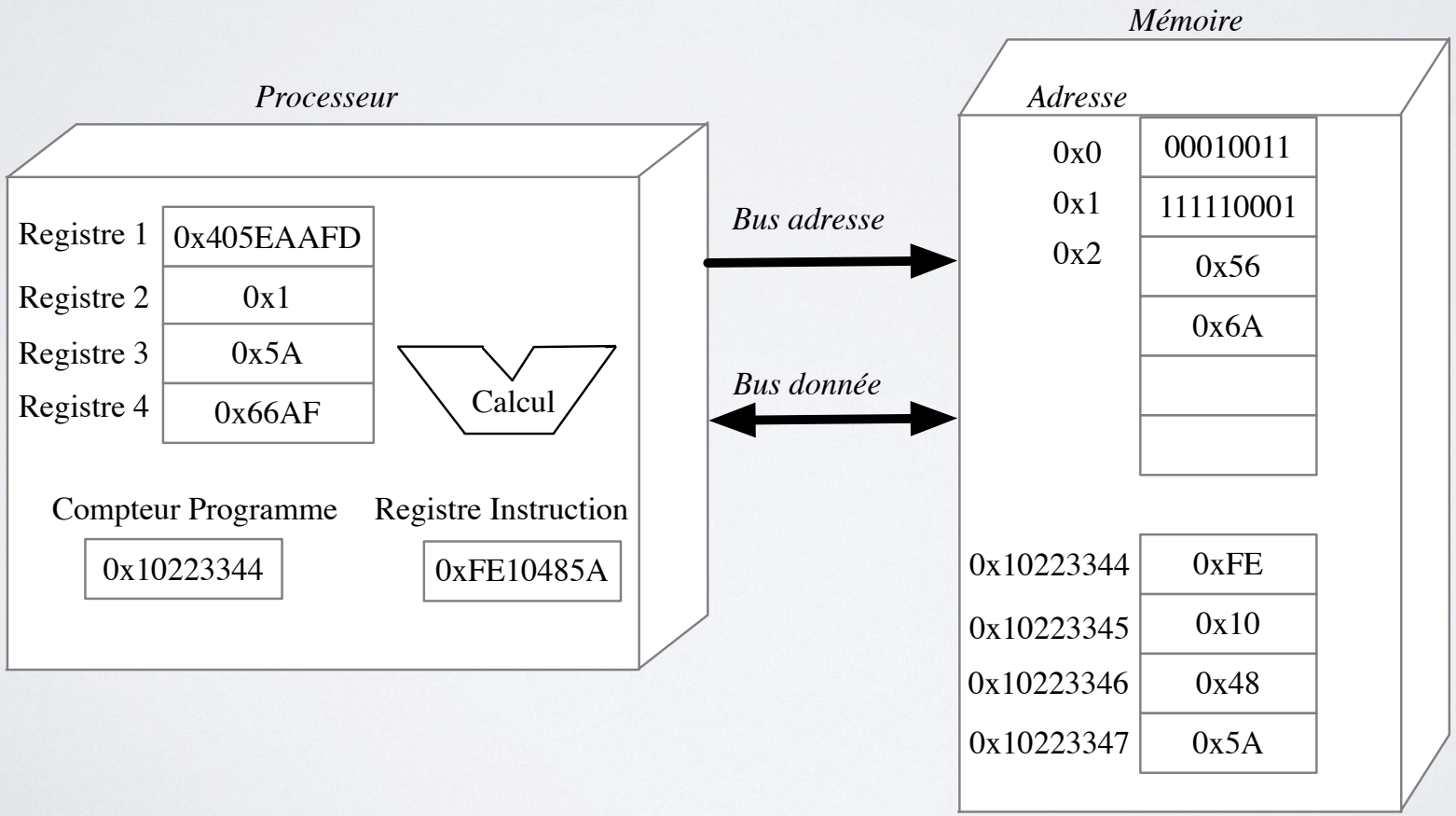
- Le processeur contient aussi des cases mémoires, peu nombreuses mais leur accès est plus rapide: les **registres**
- Des registres particuliers:
  - des registres de calcul : sauvegarde de résultats intermédiaires pendant les calculs
  - un registre contenant l'adresse de l'instruction en cours d'exécution (ou la prochaine…): le **compteur programme (PC).**
  - un **registre « instruction »** contenant l'instruction en cours d'exécution : le programmeur ne le voit pas, c'est dans ce registre que le processeur charge l'instruction qu'il va exécuter
  - un **registre d'état** contenant des informations sur l'état courant du processeur: on en reparlera plus tard

PRINCIPES DE FONCTIONNEMENT DU PROCESSEUR

- **Toutes les instructions sont sur 32 bits**
- **Tous les registres possèdent 32 bits**
- Pour effectuer des calculs sur des valeurs, ces valeurs peuvent être:
  - dans des registres de calculs (le numéro du registre est spécifié alors dans l'instruction)
  - dans l'instruction elle-même mais le champ des valeurs est restreint (par exemple codage seulement 8 bits)
- Pour accéder à la mémoire, l'adresse doit être mise impérativement au préalable dans un registre de calcul (une instructions sur 32 bits ne peut pas contenir une adresse mémoire sur 32 bits !)

PROCESSEUR/MÉMOIRE

- Les instructions sont exécutées dans l'ordre dans lesquelles elles sont stockées en mémoire sauf en cas d'instructions particulière de branchement



EXEMPLES D'INSTRUCTIONS

- **Calcul**
  - add r1, r2, r3 //commentaire: addition des valeurs de r2 et r3, le resultat est stocke dans r1
  - sub r1, r2, #77 //r1=r2+77 (en décimal)
  - mov r1, r8 // r1=r8
- **Lecture/écriture mémoire**
  - ldr r1, [r2] // r1 prend la valeur des 4 cases mémoires dont la première adresse est dans r2
  - str r2, [r5] // les 4 cases mémoires dont la première adresse est dans r5 prennent la valeur qui est dans r2
  - Il existe aussi la possibilité de lire/écrire un seul octet (ldrb, strb)ou deux octets (ldrh, strh)
- Autres
  - Branchement conditionnel, …
- On peut écrire n'importe quel algorithme avec ce jeu d'instructions

UNIVERS PEDAGOGIQUE

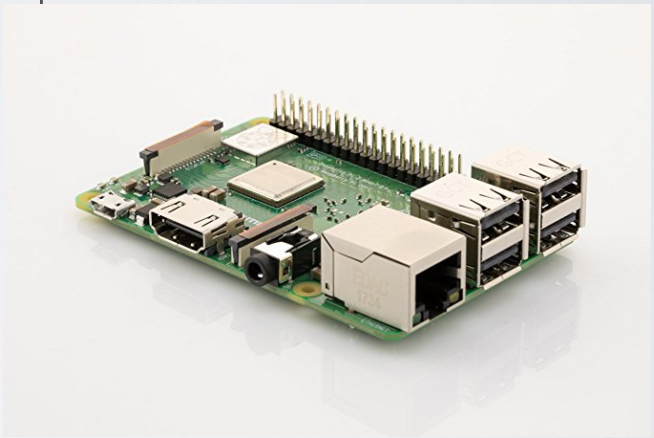
- Plus de cours magistraux (sauf cette présentation)
- Soft: Travaux pratiques en autonomie quasi complète
- Hard: explications courtes, travaux pratiques, exercices
- Evaluation des connaissances:
  - Des comptes rendus de travaux pratiques à rendre
  - Des QCMs non évalués
  - Des QCMs évalués
  - Un examen écrit final

ENVIRONNEMENT DE TRAVAIL

- Tous les documents sont sur le moodle:
  - https://im2ag-moodle.e.ujf-grenoble.fr/course/view.php?id=339
  - Documents en ligne
  - Forums
  - Rendu de TPs….
- Jusqu'à la toussaint on fait du soft :
  - Répertoire **Partie SOFT**: fichiers **cours.zip** et **sprint.zip**
  - à télécharger sur vos machines
- Ensuite du hard jusqu'à noël

ENVIRONNEMENT DE TRAVAIL

- On ne va pas travailler sur le processeur de vos machines (pas forcément le même)
- On pourrait travailler sur une carte composée d'un vrai processeur et de mémoire
  - compliqué pour des débutants
  - Fait au 2ème semestre



- Utilisation d'un émulateur d'un processeur simple et très utilisé dans l'embarqué: le ARM
- Utilisation d'un compilateur (gcc) et d'un debogueur (gdb)

ENVIRONNEMENT DE TRAVAIL

- Il faut installer sur vos portables (sous linux):
  - L'émulateur arm : qemu-system-arm
  - Les outils de compilation:
    - gcc-arm-none-eabi et binutils-arm-none-eabi
  - Le debogueur
    - gdb-arm-none-eabi ou gdb-multiarch

BIBLIOGRAPHIE

- Bouquin
  - P. Amblard, JC. Fernandez, F. Lagnier, F. Maraninchi, P. Sicard, P. Waille Architectures logicielles et matérielles. Editions DUNOD 2000
  - En bibliothèque ou en ligne sur le WEB (dans le Moodle)
- Le web en particulier wikipedia