Architecture des Ordinateurs Avancée (L3)

Cours 4 - Un peu d'assembleur et le pipeline (intro)

Carine Pivoteau [1]

1. En collaboration avec X. Goaoc

Retour sur le TP 3 : taille de ligne et taille du cache

- Exo 2 : un tableau *tab* de taille n (variable),
- une valeur de *pas* choisie dans $\{1, 2, 4, 8, 16, 32, 64, 128\}$,
- $N = 10\ 000\ 000$ accès aux cases `tab[i*pas]` pour $i = 0, 1, 2, ...$
  Autrement dit, un nombre fixe d'accès aux cases d'un tableau espacées d'une petite puissance de 2.
- Que s'attend-on à voir ?
- Qu'observe-t-on pour les différentes valeurs de *pas* lorsque n augmente ?
- Que peut-on en déduire sur les paramètres du cache ?

Retour sur le TP 3 : taille de ligne et taille du cache

- Exo 2 : un tableau *tab* de taille n (variable),
- une valeur de *pas* choisie dans $\{1, 2, 4, 8, 16, 32, 64, 128\}$,
- $N = 10\ 000\ 000$ accès aux cases `tab[i*pas]` pour $i = 0, 1, 2, ...$
  Autrement dit, un nombre fixe d'accès aux cases d'un tableau espacées d'une petite puissance de 2.
- Que s'attend-on à voir ?
- Qu'observe-t-on pour les différentes valeurs de *pas* lorsque n augmente ?
- Que peut-on en déduire sur les paramètres du cache ?

Retour sur le TP 3 : associativité

- Exo 3 : un très grand tableau d'entiers de taille 10^9,
- comme *pas*, une grande puissance de 2, par ex. $2^{20} = 1048576$,
- répéter un grand nombre n de fois un petit nombre k d'accès à des cases séparées de *pas* cases, de telle sorte que $N = n \times k$ soit fixe.
- On choisit $N = 2^{28} = 268435456$,
- et on fait varier k dans $\{1, 2, 3, 4, 6, 8, 10, 12, 16, 18, 20, 30\}$.
- Que s'attend-on à voir ?
- Qu'observe-t-on pour les différentes valeurs de *pas* ?
- Que peut-on en déduire sur les paramètres du cache ? Vérifier les paramètres dans : `/sys/devices/system/cpu/cpu0/cache/.../ways_of_associativity`.

Retour sur le TP 3 : associativité

- Exo 3 : un très grand tableau d'entiers de taille 10^9,
- comme *pas*, une grande puissance de 2, par ex. $2^{20} = 1048576$,
- répéter un grand nombre n de fois un petit nombre k d'accès à des cases séparées de *pas* cases, de telle sorte que $N = n \times k$ soit fixe.
- On choisit $N = 2^{28} = 268435456$,
- et on fait varier k dans $\{1, 2, 3, 4, 6, 8, 10, 12, 16, 18, 20, 30\}$.
- Que s'attend-on à voir ?
- Qu'observe-t-on pour les différentes valeurs de *pas* ?
- Que peut-on en déduire sur les paramètres du cache ? Vérifier les paramètres dans : `/sys/devices/system/cpu/cpu0/cache/.../ways_of_associativity`.

Retour sur le TP 3 : simulateur de cache

Simuler les accès à un cache non associatif de 2^{15} octets avec des lignes de 64 octets.

Principe : conserver, pour chaque ligne de cache, un entier représentant l'indicateur du bloc stocké en cache lors d'un accès fictif à une adresse donnée.

▷ Écrire une fonction `cache_simulator` qui prend en paramètre l'adresse d'une case mémoire et simule un accès en cache pour cette case. Elle renvoie 1 si c'est un cache miss et 0 sinon. Pour tester, nous vos conseillons d'afficher les informations suivantes :

- le numéro de ligne de cache correspondant,
- l'identifiant en RAM correspondant,
- si cela provoque un cache-miss ou non.

Le code assembleur... produit par gcc

Le langage du processeur, ou presque

- On ne va pas refaire le cours de compilation.
- Mais on doit pouvoir lire/comprendre du code assembleur,
- et imaginer le cycle de vie des instructions dans le processeur.
- Objectif :
  - avoir une vision suffisamment fine pour comprendre certaines caractéristiques des processeurs modernes,
  - comprendre comment on peut en tirer parti au niveau du code.

Obtenir du code assembleur / machine à partir d'un programme C

- Assembleur x86, syntaxe intel.
- Désassembler :

```
gcc demo-C4-0.c
objdump -d -M intel ./a.out
```

ou (Mac) : `objdump -d -x86-asm-syntax=intel ./a.out`

- Compiler en assembleur :

```
gcc demo-C4-0.c -S -masm=intel
```

Exo : être capable de lire de l'assembleur

- `https://en.wikibooks.org/wiki/X86_Assembly`
- Écrire un bout de code C simple, par exemple :

```
int j = 5;
int k = 3*j + 7;
```

- ... et lire le code produit par les deux commandes précédentes.

  ▷ Note : l'assembleur x86 est de type CISC (Complex Instruction Set Computer).

- Et si on compile avec les optimisations de gcc ?

```
gcc demo-C4-0.c -O1
```

- Modifier un peu le code (affichage, boucle, ...) et observer l'assembleur (optimisé ou non).

Les optimisations de gcc

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

- `-O0` (valeur par défaut) : **quasiment aucune optimisation**. Faible temps de compilation et code facile à lire.
- `-O1` : niveau d'optimisation **le plus basique**. Le compilateur essaie de produire un code plus rapide et plus compact sans prendre trop de temps de compilation.
- `-O2` : niveau **recommandé** d'optimisation. Le compilateur essaie d'augmenter la performance sans compromettre la taille et sans prendre trop de temps en compilation.
- `-O3` : le plus haut niveau d'optimisation (pas recommandé).
  - Coûteux en temps de compilation et d'usage de la mémoire.
  - Ne garantit pas une amélioration de la performance.
  - Binaires plus volumineux qui réclament plus de mémoire.
- `-Os` : optimise la **taille du code** (active toutes les options de `-O2` qui n'augmentent pas la taille du code).

Exemple (`demo-C4-1.c`)

Lire le code assembleur produit par les différentes options d'optimisation de `gcc` et commenter.

Par exemple :

- utilisation des registres au lieu de la pile (`-O1`)
- utilisation des instructions de déplacement conditionnel (`-O1`)
- *inlining* des petites fonctions (`-O2`)
- déroulement de boucle (`-O2`)
- vectorisation (`-O2`)
- ...

Exemple (`demo-C4-1.c`)

Lire le code assembleur produit par les différentes options d'optimisation de `gcc` et commenter.

Par exemple :

- utilisation des registres au lieu de la pile (`-O1`)
- utilisation des instructions de déplacement conditionnel (`-O1`)
- *inlining* des petites fonctions (`-O2`)
- déroulement de boucle (`-O2`)
- vectorisation (`-O2`)
- ...

Exemple (`demo-C4-1.c`)

Lire le code assembleur produit par les différentes options d'optimisation de `gcc` et commenter.

Par exemple :

- utilisation des registres au lieu de la pile (`-O1`)
- utilisation des instructions de déplacement conditionnel (`-O1`)
- *inlining* des petites fonctions (`-O2`)
- déroulement de boucle (`-O2`)
- vectorisation (`-O2`)
- ...

Exemple (`demo-C4-1.c`)

Lire le code assembleur produit par les différentes options d'optimisation de `gcc` et commenter.

Par exemple :

- utilisation des registres au lieu de la pile (`-O1`)
- utilisation des instructions de déplacement conditionnel (`-O1`)
- *inlining* des petites fonctions (`-O2`)
- déroulement de boucle (`-O2`)
- vectorisation (`-O2`)
- ...

La plupart des ces optimisations sont liées à caractéristiques plus ou moins avancées de l'architecture du processeur

- `http://www.agner.org/optimize/optimizing_assembly.pdf`
- `http://icps.u-strasbg.fr/~bastoul/local_copies/lee.html`

Exo (demo-C4-2.c)

- En C, déclarer et initialiser 3 entiers a, b et c.
- Compiler avec gcc sans optimisation.
- Mesurer le nombre de cycles que prend l'exécution de la paire d'instructions :

```
b = a + 13;
a = c + 17;
```

- Puis mesurer le nombre de cycles pour :

```
b = a + 13;
a = b + 17;
```

- Vérifier le code assembleur produit.
- Commenter.

Le pipeline

L'efficacité d'un processeur : une histoire de cycles

- L'horloge d'un circuit électronique (par ex. le processeur) correspond au signal électrique qui rythme ses actions.
- À chaque cycle d'horloge, des calculs sont effectués. La durée du cycle correspond à peu près au temps d'un calcul simple.
- La fréquence d'horloge est le nombre de cycles pour un temps donné. Par exemple, 1 GHz = 1 milliard de cycles en une seconde.
- On pourrait imaginer qu'il suffit d'augmenter la fréquence pour avoir un processeur plus rapide...
- ... mais certaines opérations, comme les accès mémoire, peuvent prendre plus d'un cycle :(

L'efficacité d'un processeur : une histoire de cycles

- L'horloge d'un circuit électronique (par ex. le processeur) correspond au signal électrique qui rythme ses actions.
- À chaque cycle d'horloge, des calculs sont effectués. La durée du cycle correspond à peu près au temps d'un calcul simple.
- La fréquence d'horloge est le nombre de cycles pour un temps donné. Par exemple, 1 GHz = 1 milliard de cycles en une seconde.
- On pourrait imaginer qu'il suffit d'augmenter la fréquence pour avoir un processeur plus rapide...
- ... mais certaines opérations, comme les accès mémoire, peuvent prendre plus d'un cycle :(

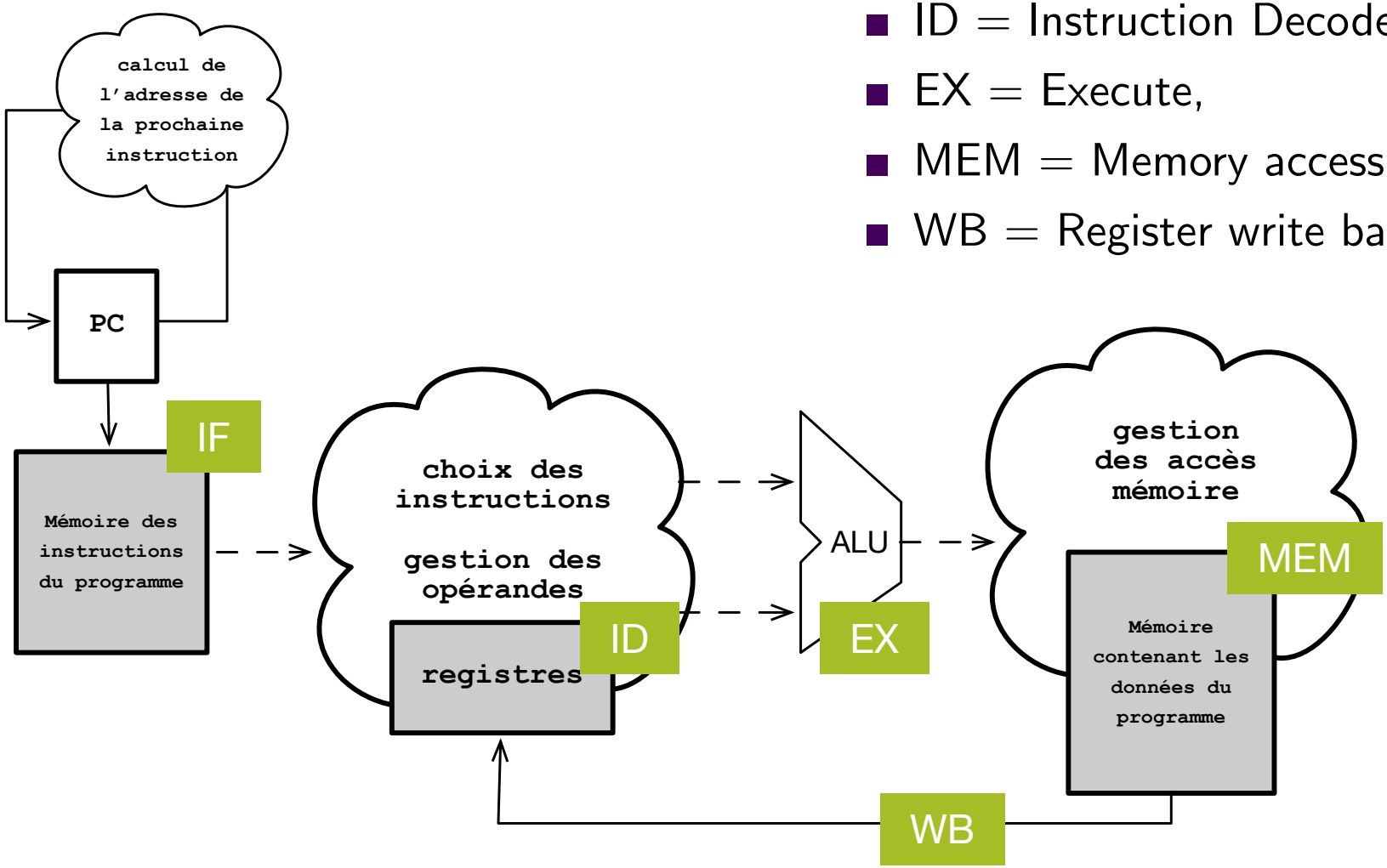
L'efficacité d'un processeur : latence vs. débit

- Le nombre de cycles que prend une opération pour s'effectuer, c'est sa **latence** (*latency*).
  `https://www.agner.org/optimize/instruction_tables.pdf`
- Comment améliorer la latence ?
  - Travailler sur le temps d'accès à la mémoire permet d'améliorer la latence des opérations correspondantes.
  - C'est la seule façon d'améliorer l'efficacité d'un processeur ?
- Une autre idée est d'effectuer **plusieurs opérations par cycle**. On introduit du parallélisme au niveau des instructions (**ILP**).
- De cette façon, on cherche à améliorer le **débit** (*troughput*). On parle en général du nombre moyen d'instructions exécutées à chaque cycle (**IPC**).

Une instruction à la loupe

Découpage classique d'une instruction (modèle RISC = *Reduced Instruction Set Computer*) en 5 étapes :

- IF = Instruction Fetch,
- ID = Instruction Decode,
- EX = Execute,
- MEM = Memory access,
- WB = Register write back.



Un exemple simple

Programme compilé $\simeq$ séquence d'octets en mémoire (lang. machine). Lors de l'exécution, le registre `ip` contient l'adresse du premier octet de l'instruction courante. Par exemple : `01 c2 83 c0 01 3d 41 42 0f 00`...

Traiter cette instruction nécéssite plusieurs opérations distinctes :

- **décoder l'instruction**. La prochaine instruction à exécuter est codée par `01 c2` ce qui correspond à `add edx, eax`.
- **lire les opérandes** : c'est à dire transmettre en entrée du circuit additionneur les valeurs des registres `edx` et `eax`.
- **exécuter l'instruction** : effectuer le calcul via un circuit additionneur et positionner les retenues.
- **écrire les résultats** : mettre à jour le registre `edx` avec le résultat et positionner les flags modifiés par l'instruction `add`.

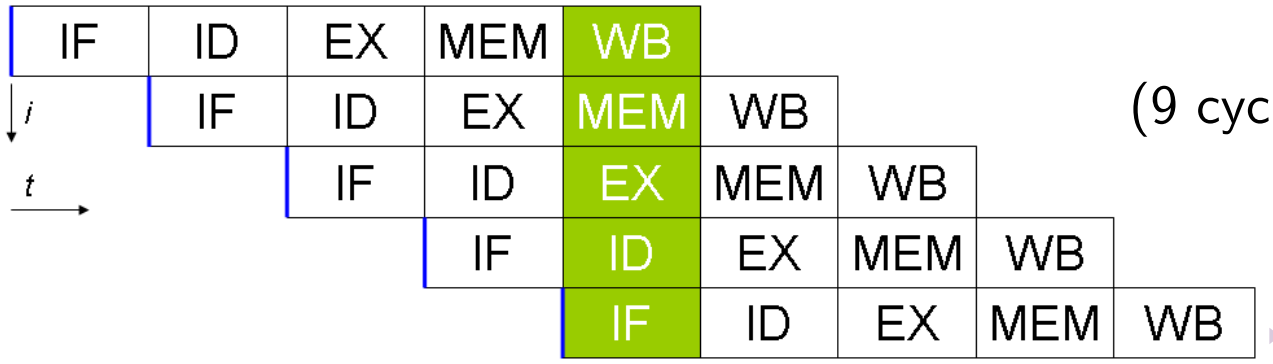
Ensuite, on peut ajouter 2 à `ip` pour le faire pointer sur l'instruction suivante. Et recommencer : décodage, lecture d'opérandes, exécution, écriture ...

Le pipeline (années 60)

Exécution séquentielle
(15 cycles pour 3 instructions)

Avec pipeline :

- Chaque étape utilise une portion de circuit différente (en gros ...).
- On suppose que chaque étape prend environ un cycle à s'exécuter.
- On cherche à introduire de l'**ILP** (*Instruction Level Parallelism*).
- On peut utiliser le principe d'une chaîne de montage.



Exécution parallèle
(9 cycles pour 5 instructions)

Est-ce aussi simple que ça ?

- Pourquoi on ne met pas encore plus d'étages ?
- Est-ce que le pipeline est toujours rempli ?
- Qu'est-ce qui pourrait l'en empêcher ?

Est-ce aussi simple que ça ?

- Pourquoi on ne met pas encore plus d'étages ?
- Est-ce que le pipeline est toujours rempli ?
- Qu'est-ce qui pourrait l'en empêcher ?

Est-ce aussi simple que ça ?

- Pourquoi on ne met pas encore plus d'étages ?
- Est-ce que le pipeline est toujours rempli ?
- Qu'est-ce qui pourrait l'en empêcher ?
  - La dépendance des données...
  - Les instructions conditionnelles...

Est-ce aussi simple que ça ?

- Pourquoi on ne met pas encore plus d'étages ?
- Est-ce que le pipeline est toujours rempli ?
- Qu'est-ce qui pourrait l'en empêcher ?
  - La dépendance des données...
  - Les instructions conditionnelles...

Est-ce aussi simple que ça ?

- Pourquoi on ne met pas encore plus d'étages ?
- Est-ce que le pipeline est toujours rempli ?
- Qu'est-ce qui pourrait l'en empêcher ?
  - La dépendance des données...
  - Les instructions conditionnelles...
- Est-ce qu'on peut trouver des solutions ?
- Quels compromis faut-il faire ?

Est-ce aussi simple que ça ?

- Pourquoi on ne met pas encore plus d'étages ?
- Est-ce que le pipeline est toujours rempli ?
- Qu'est-ce qui pourrait l'en empêcher ?
  - La dépendance des données...
  - Les instructions conditionnelles...
- Est-ce qu'on peut trouver des solutions ?
- Quels compromis faut-il faire ?
- ... Réponse au prochain cours.