Systèmes numériques : de l'algorithme aux circuits

Les processeurs du monde réel

Hadrien Barral (prenom.nom@ens.psl.eu)

Version : 1.1

Qu'est ce qui existe comme architectures ?

Crédit : Épaurif

Qu'est ce qui existe comme architectures ?

https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#Instruction_sets

(loin d'être exhaustif)

Familles d'architectures

*ISC

RISC vs CISC vs ...

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

Qui est mieux ? Pourquoi ?

Pour aller plus loin : `VLIW`, `ZISC`, `MISC`, ...

Familles d'architectures

Usage

Microcontrolleur vs ...

- Calcul pur ou plusieurs entités sur le même matériel ?
- Puissance électrique ou puissance de calcul ?

Cours d'histoire

L'histoire étant écrite par les gagnants, on ne parlera que des architectures suivantes :

- `x86/AMD64`
- `ARM`
- `RISC-V`

Un peu d'histoire : Intel

- Premier microprocesseur : Intel 4004 (1971), 4-bit
- x86 : architecture 16-bit (1978), passage au 32-bit (1985), passage au 64-bit (2003)
  - Poussé par AMD64, Intel voulait faire IA64…qui a été un échec.
- Archétype de l'architecture CISC
  - Nécessité d'avoir un *microcode*
- Intel fait l'architecture, la micro-architecture et fabrique les puces [1]
- Nombre de ventes en 2023 : 300 millions (à la grosse louche)

1. Voir conditions en magasin.

Un peu d'histoire : ARM

- Débuts avec l'ARM1 (1985)
  - 25k transistors (10 fois moins que Intel à l'époque), mais très bonne efficacité énergétique
- ARM fait l'architecture et la micro-architecture, vend l'IP et délègue la fabrication des puces [2]
- Passage au 64-bit en 2012
- Architecture initialement RISC.
  - De nos jours, c'est compliqué car il y a Cortex A/R/M [3]
- 1985-2025 : 250 milliards de puces vendues
  - À comparer au nombre d'humains sur $]-\infty; 2025]$ (<120 milliards)

2. Voir conditions en magasin.
3. Voir conditions en magasin.

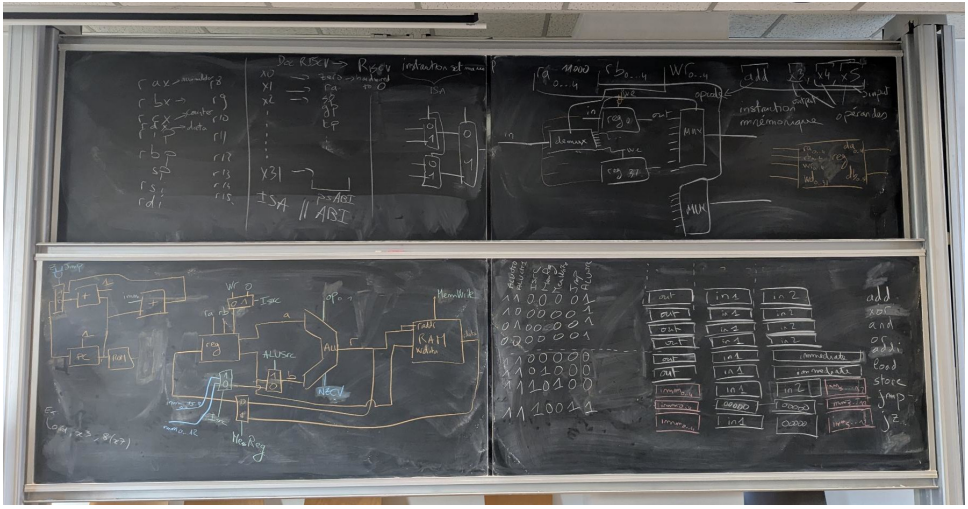
Un peu d'histoire : RISC-V

- Début en 2010 (architecture uniquement). Focus sur une architecture *libre*
- RISC-V fait spécification, d'autres font la micro-architecture, d'autres fabriquent les puces
  - La specification est libre, mais pas nécéssairement le reste !
  - Still, plusieurs cores sont entièrement open-source et open-hardware
- Architecture à la fois 32-bit et 64-bit depuis le début
- Initialement RISC
  - $\rightarrow$ de nos jours, plus nuancé, car multiples extensions existent
- Pour l'instant relativement nouveau, peu de ventes [4]

4. Et difficile à compter.

Les base de l'architecture

Rappel

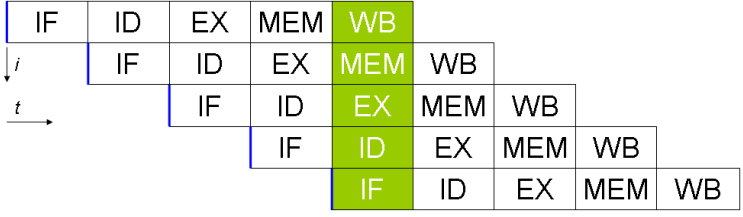


Pipelining

Hypothèse

On se place dans le modèle du *classic RISC pipeline* :

1. *Fetch* : on charge l'instruction
2. *Decode* : on décode l'instruction et on lit les entrées (registre, *flag*, mémoire, ...)
3. *Execute* : on traîte les entrées
4. *Memory* : *(optionel)* on accède à la mémoire
5. *Write-back* : On stocke les sorties (registre, *flag*, mémoire, ...)



Source : Wikimedia

Pipeline hazards

Dans le *pipelining*, il existe des situations où l'instruction suivante ne peut pas être exécutée au cycle d'horloge suivant. Ces événements sont appelés *hazards*.
Dit autrement, il y a *hazard* dans les situations où le pipelining produirait un mauvais résultat.
3 types :

- *Structural hazard*
- *Data hazard*
- *Control hazard*

Structural hazards

Structural hazard : plusieurs instructions veulent utiliser la même ressource (e.g. la mémoire ou l'ALU) en même temps.

Note : la *classic pipeline RISC* n'a par construction pas ce problème.
Sinon, une des instructions doit attendre.

Exercice

Que se passe-t-il si on a une mémoire unifié programme+données ?

Data hazards

Data hazard : lorsqu'une instruction ne peut être exécutée au cycle d'horloge suivant car les données nécessaires à son exécution ne sont pas encore disponibles.

Exemple

```
add x8, x1, x2
sub x9, x3, x8
```

Par défaut, blocage du pipeline (*stalling*).

- Utopique de penser qu'un compilateur peut éviter tous les *hazards*
- Solution classique N° 1 : *forwarding* (aussi nommé *bypassing*)
- Solution classique N° 2 : *interlock*

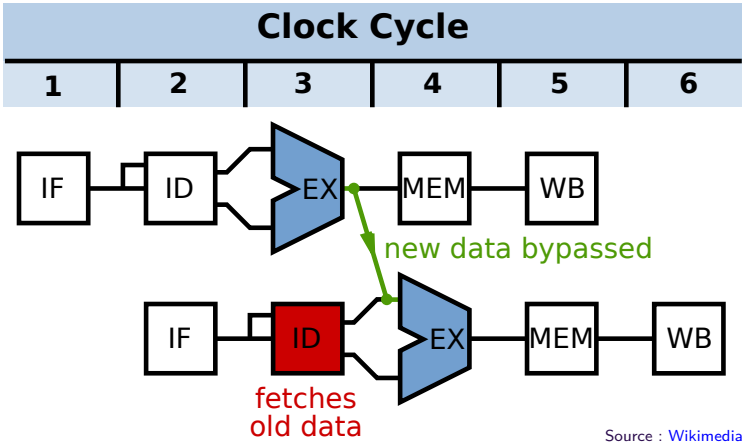
Data hazards

Forwarding

Forwarding : on récupères les entrées directement depuis les buffers internes plutôt que depuis les entrées *programmer-visible*.

Exemple

```
add x8, x1, x2
sub x9, x3, x8
```



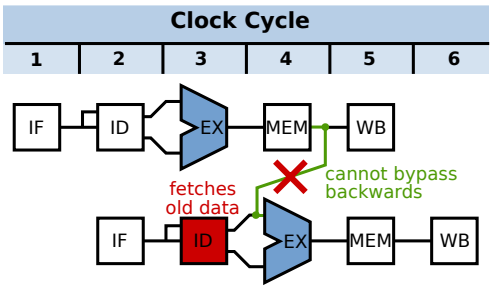
Source : Wikimedia

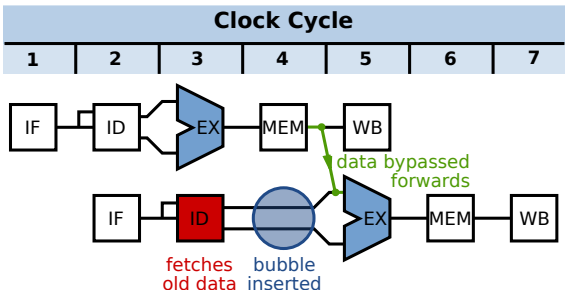
Data hazards

Interlock

Exemple

```
lw  x2, 0(r1)
add x9, x2, x3
```





Data hazards

Forwarding

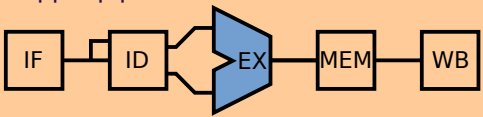
Exercice

```
x = a + b;
y = a + c;
```

```
ld  x1, 0(x31)  // Load a
ld  x2, 8(x31)  // Load b
add x3, x1, x2  // a + b
sd  x3, 24(x31) // Store x
ld  x4, 16(x31) // Load c
add x5, x1, x4  // a + c
sd  x5, 32(x31) // Store y
```

1. Trouvez les *hazards*.
2. Réordonnez les instructions pour faire mieux.

Rappel pipeline :



Control hazards

Control hazard : lorsque l'instruction appropriée ne peut pas être exécutée au cycle suivant parce que l'instruction qui a été chargée n'est pas celle qui est nécessaire. Dit autrement, que le flux des adresses d'instructions du pipeline n'est pas celui attendu.

Exemple

```
start: add x1, x1, x3
       beq x1, x0, start
       sltiu x5, x6, x7
```

Est-ce un vrai problème en pratique pour les performances ? → OUI !

Control hazards

Résolution

Comment gérer ce problème ? →

- *Predict Not Taken* 🤪
- *Branch Likely* 🥳
- *Branch Delay Slot* 🤮
- *Branch Prediction* 🔮

Exemples sur de vrais CPUs

- *Computer Organization and Design : The Hardware / Software Interface : Risc-V Edition*, section 4.11
- https://chipsandcheese.com/p/arms-cortex-a53-tiny-but-important
- https://en.wikichip.org/

Hiérarchie mémoire

(*au tableau*)