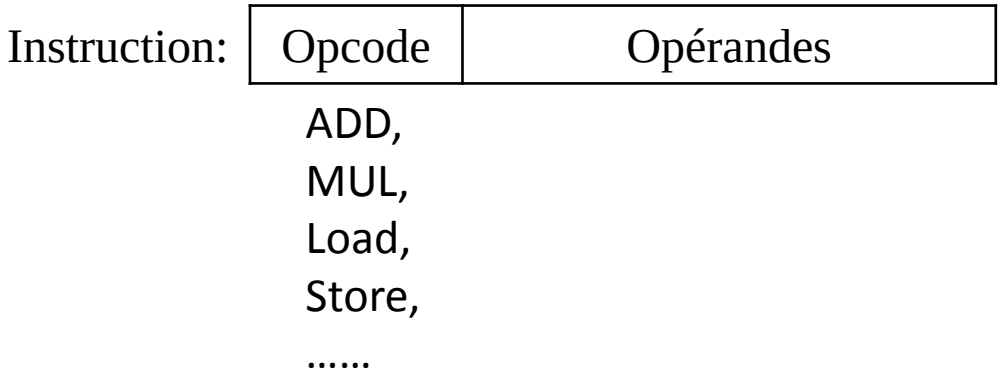
Jeu d’instructions, Pipeline et parallélisme

Le jeu d'instructions

- C'est l'ensemble d'instructions élémentaires (instructions machine) qu'un processeur peut faire (add, load, store, etc.)

- Au cours de la phase de décodage de l'instruction (Decode), le processeur analyse l'instruction et extrait son code d'opération (opcode) pour déterminer de quelle instruction il s'agit et envoyer les commandes correspondantes à l'UAL ou à la mémoire



Le jeu d'instructions

- Il y a deux principaux types de jeux d'instructions permettant de classer les ordinateurs en deux catégories

  - RISC: Reduced Instruction Set Computer
    - Ordinateur/processeur à jeu d'instructions simple

  - CISC: Complex Instruction Set Computer
    - Ordinateur/processeur à jeu d'instructions complexe

RISC vs CISC

- RISC
  - Un nombre réduit d'instructions élémentaires
  - Des instructions simples
  - Des instructions souvent de même taille
  - Temps de traitement réduit : 1 seul cycle machine par instruction
  - Chaque instruction en langage haut niveau (C, java, ..) sera traduite en un ensemble d'instructions élémentaires simples, lors de la compilation → des programmes occupant un espace mémoire assez grand

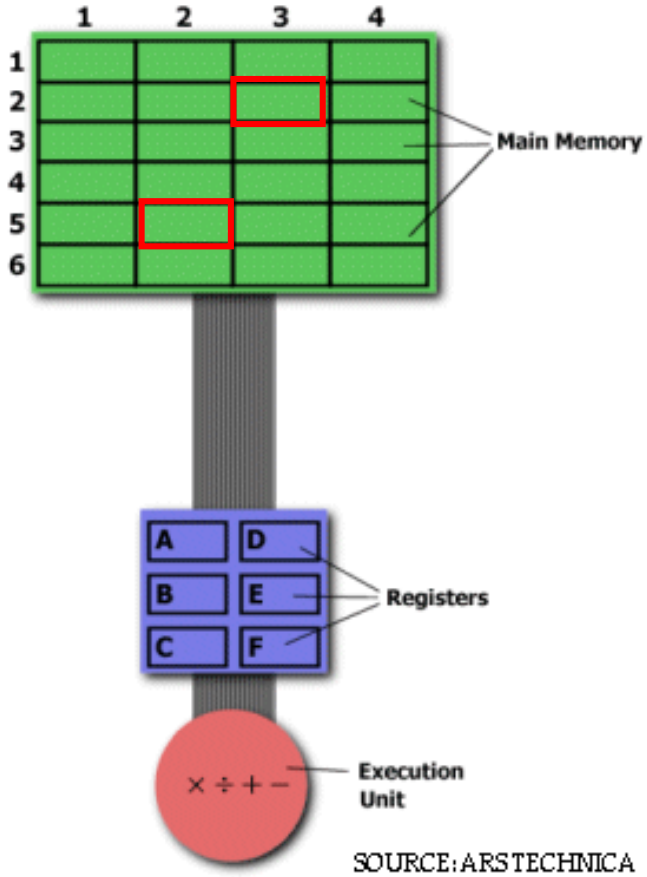
RISC vs CISC

- CISC
  - Un grand nombre d'instructions
  - Des instructions complexes (combinaison d'instructions simples)
  - Des instructions souvent de tailles variables
  - Temps de traitement plus long : plus d'un cycle machine par instruction
    - Le temps d'exécution des instructions est variable selon leur complexité
  - Des programmes plus courts que ceux des ordinateurs RISC (**l'objectif principal des CISC était d'avoir des programmes de petite taille car la mémoire coûtait cher à l'époque**)

RISC vs CISC

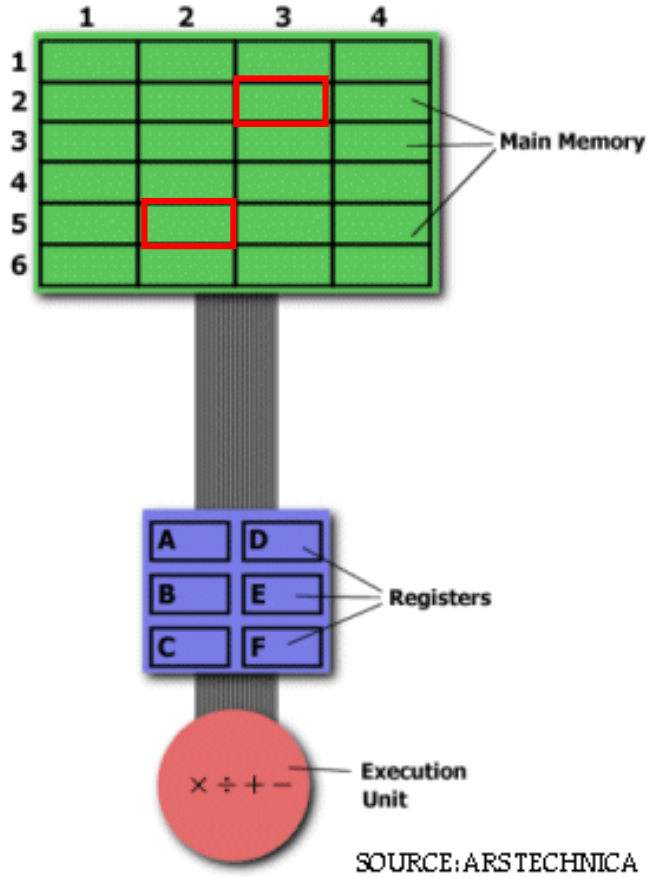
- Exemple
  - Faire le produit des deux nombres qui sont dans les cases 2:3 et 5:2 et mettre le résultat dans la case 2:3
  - Approche CISC
    - Utilisation d'instructions complexes
      - Exemple: une instruction MULT permettant de faire le produit des contenus de deux cases et mettre le résultat dans l'une des deux.

        MULT 2:3, 5:2



RISC vs CISC

- Exemple
  - Faire le produit des deux nombres qui sont dans les cases 2:3 et 5:2 et mettre le résultat dans la case 2:3
  - Approche RISC
    - Les RISCs utilisent des instructions simples
      - **LOAD A, 2:3** (mettre dans le registre A le contenu de la case 2:3)
        LOAD B, 5:2 (mettre dans le registre B le contenu de la case 5:2)
        PROD A, B (faire la multiplication de A et B et mettre le résultat dans A)
        STORE 2:3, A (mettre le contenu de A dans la case 2:3)



RISC vs CISC

- Avantages du RISC
  - Rapidité de l'exécution grâce au pipeline (tâches exécutées en parallèle)
  - Moins de consommation d'énergie grâce à la simplicité du design du processeur

- Avantages du CISC
  - Des programmes plus réduits en espace mémoire avec moins d'instructions machine
  - Plus d'applications et d'environnements de développement disponibles grâce à l'avance prise par ces processeurs sur le marché

RISC vs CISC

- Utilisation
  - Les processeurs CISC sont typiquement utilisés pour les ordinateurs.
  - Les processeurs RISC sont typiquement utilisés pour les smartphones, tablettes et autres appareils électroniques puisqu'ils consomment peu d'énergie et offrent plus d'autonomie
- Exemples de processeurs CISC et RISC
  - CISC: x86 (Intel et AMD)
  - RISC: ARM (plusieurs fabricants), MIPS (MIPS Technologies), Itanium (Intel), PowerPC (AIM)

RISC vs CISC

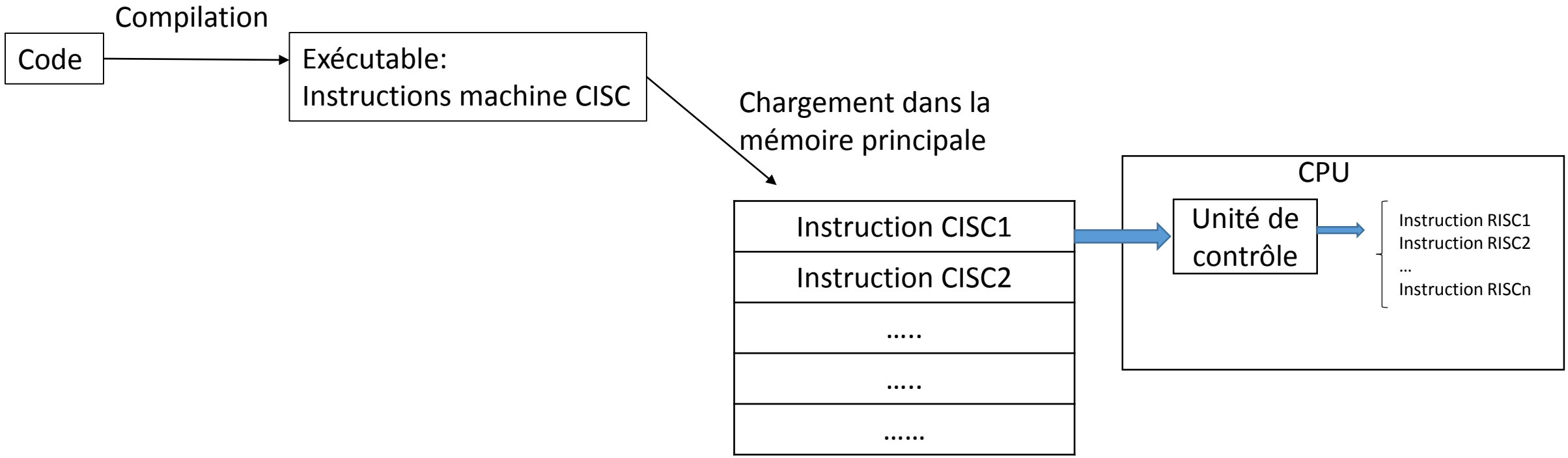
- Pourquoi la plupart des ordinateurs utilisent des processeurs CISC malgré l'avantage des RISC?

  - La plupart des applications existantes ont été développées pour le jeu d'instruction x86, elles sont non utilisables sur un processeur RISC

  - Intel a tenté d'adopter l'architecture RISC (Itanium) mais ces processeurs n'ont pas eu un grand succès à cause des problèmes de compatibilité avec les applications logicielles déjà existantes qui étaient conçues pour des processeurs CISC

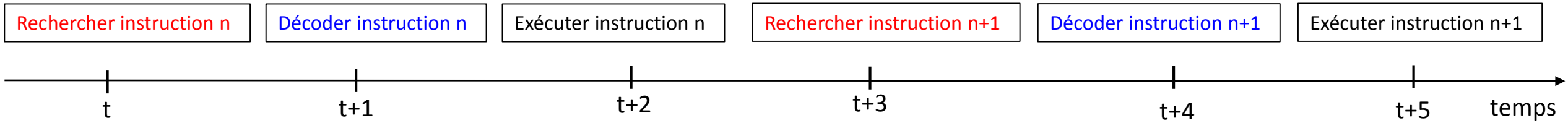
RISC vs CISC

- Convergence entre RISC et CISC
  - De nos jours, les processeurs CISC (Intel) « cachent » un cœur RISC
    - Une instruction CISC d'un programme est traduites en des micro-instructions RISC par l'unité de contrôle du processeur
    - Cela permet aux processeurs CISC de profiter du **pipelining** des RISC et d'augmenter ainsi leur performance.



Le pipeline

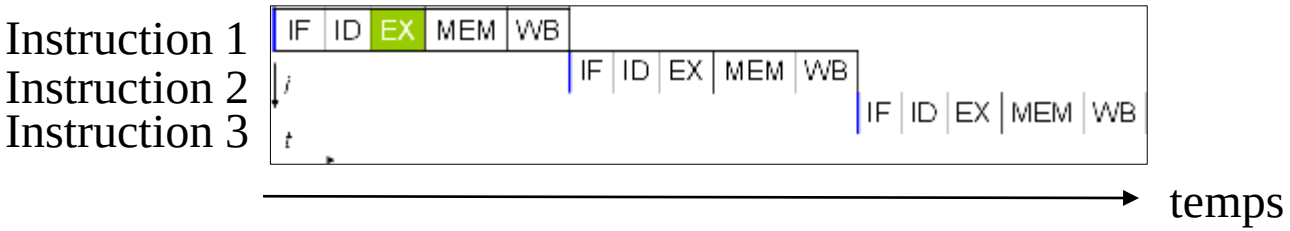
- Rappel: Le cycle d'instruction FDX (Fetch-Decode-Execute)
  - Fetch: recherche d'instruction
  - Decode: décodage d'instruction
  - Execute: exécution de l'instruction



- Le pipeline permet d'**exécuter des parties** de plusieurs instructions **simultanément** → accélérer l'exécution d'un programme
  - Comment?: les processeurs supportant le pipeline sont conçus sous forme d'étages, appelés étages du pipeline
  - Le pipeline est l'un des avantages des processeurs RISC qui le supportent mieux que les CISC

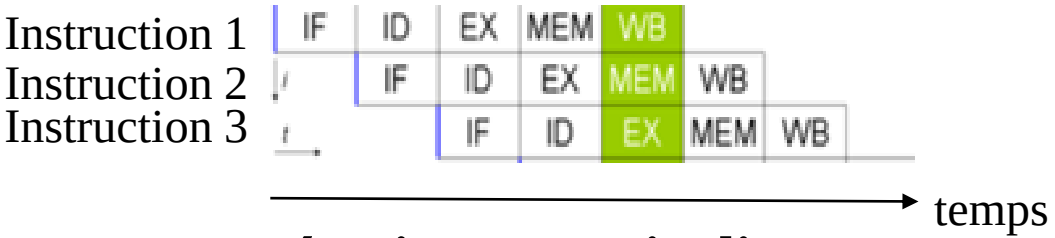
Le pipeline

- Exemple: Pipeline d'un processeur MIPS (5 étages)
  - IF (Instruction Fetch) charge l'instruction à exécuter dans le pipeline.
  - ID (Instruction Decode) décode l'instruction et adresse les registres.
  - EX (Execute) exécute l'instruction (par la ou les unités arithmétiques et logiques).
  - MEM (Memory), dénote un transfert depuis un registre vers la mémoire dans le cas d'une instruction du type STORE (accès en écriture) et de la mémoire vers un registre dans le cas d'un LOAD (accès en lecture).
  - WB (Write Back) stocke le résultat dans un registre



Exécution sans pipeline (séquentielle)

Une instruction ne peut commencer qu'après la fin des 5 phases de la précédente



Exécution avec pipeline

Quand la 1ère instruction est en phase ID, la deuxième est en phase IF et ainsi de suite

→ plusieurs phases sont exécutées en parallèle

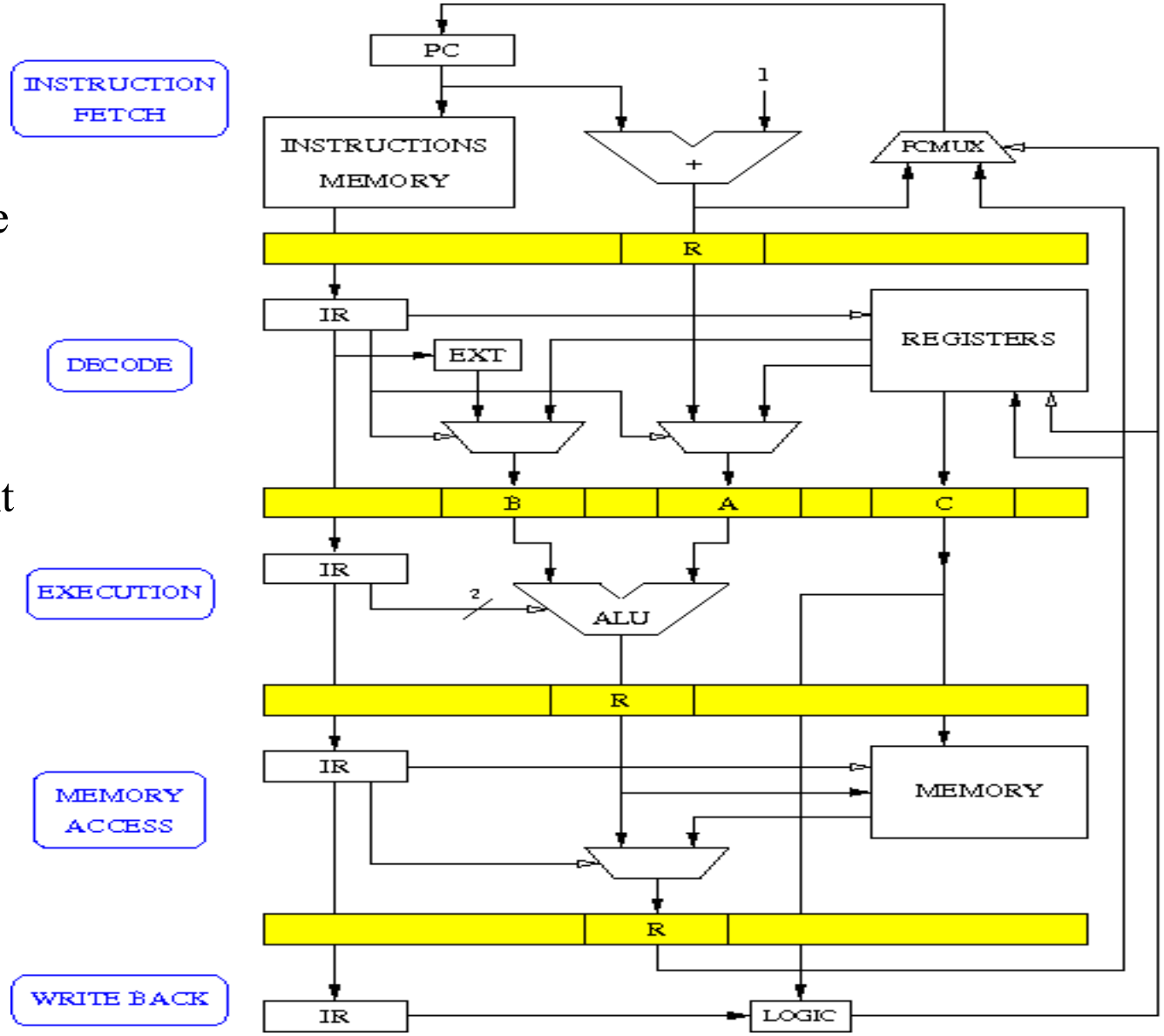
→ Accélération de l'exécution des instructions

Le pipeline

- Pipeline RISC classique

  - Le passage d'une instruction dans chaque étage dure un cycle d'horloge

  - Le résultat de chaque étage est stocké dans des buffers qui sont accessibles par l'étage d'après au cycle d'horloge suivant



Le pipeline

- Le nombre d'étages du pipeline dépend du processeur
  - ARM Cortex-M0+, Atmel AVR et PIC: deux étages
    - Fetch-Decode: charge et décode une instruction
    - Exec: exécution de l'opération avec les accès à la mémoire et/ou aux registres
  - IBM Stretch project
    - Fetch : chargement de l'instruction
    - Decode : décodage de l'instruction ;
    - Exec : exécution de l'instruction, avec éventuellement des accès mémoires.

  - On pourrait aussi avoir un pipeline à 7 étages :
    - PC : mise à jour du Program Counter ;
    - Fetch : chargement de l'instruction depuis la mémoire ;
    - Decode : décodage de l'instruction ;
    - Register Read : lecture des opérandes dans les registres ;
    - Exec : calcul impliquant l'ALU ;
    - Mem : accès mémoire en lecture ou écriture ;
    - Writeback : écriture du résultat d'une lecture ou d'un calcul dans les registres.

Le pipeline

- Le temps d'exécution d'un programme avec un pipeline

- Chaque étage du pipeline s'exécute en un cycle d'horloge → le temps d'exécution d'un programme contenant N instructions = (nombre d'étages + (N-1)) cycles d'horloge

N	Temps d'exécution du programme
3	(5 + 2) cycles =7 cycles
10	= (5 + 9) cycles =14 cycles
100	(5 + 99) cycles =104 cycles

Le pipeline

- Le CPI (nombre de cycles par instruction)
  - C'est une mesure de la performance du processeur

N	Temps d'exécution du programme	CPI
3	(5 + 2) cycles =7 cycles	7/3= 2,33 cycles
10	= (5 + 9) cycles =14 cycles	14/10= 1,4 cycles
100	(5 + 99) cycles =104 cycles	104/100= 1, 04 cycles

➔Ce processeur a un CPI de 1.04 cycle par instruction pour un programme de 100 instructions

➔Plus on a d'instructions, plus on tend vers le temps d'exécution idéal (1 cycle par instruction)

Le pipeline

- Accélération grâce au pipeline

  - Accélération= [(nb cycles sans pipeline - nb cycles avec pipeline)/ nb cycles sans pipeline]*100

  - Exemple: si on suppose que pour la version non pipelinée du processeur, une instruction prend 2 cycles et que la version pipelinée contient 5 étages

N	Temps d'exécution version non pipelinée	Temps d'exécution version pipelinée	Accélération
3	2x3= 6 cycles	(5 + 2) cycles =7 cycles	[(6-7)/6]*100=-16.66%
10	2x10=20 cycles	= (5 + 9) cycles =14 cycles	[(20-14)/20]*100=30%
100	2x100=200 cycles	(5 + 99) cycles =104 cycles	[(200-104)/200]*100=48%

  - L'accélération due au pipeline augmente avec le nombre d'instructions d'un programme

  - Quelle est l'accélération maximale qu'on peut avoir pour cet exemple pour un très grand nombre d'instructions?

Le pipeline

- Problème / aléa:
  - Il y a des cas où le processeur est obligé d'arrêter le pipeline pour un certain nombre d'étages
  - Parmi ces cas, si une instruction utilise une donnée qui est modifiée par l'instruction précédente qui n'a pas fini son exécution. Le blocage du pipeline est appelé souvent pipeline « stall ».

    - Exemple d'une suite d'instructions en langage C

      d=a+b;

      e=d-c;

    - Le processeur ne peut pas décoder la deuxième instruction et déterminer les valeurs de ces opérandes avant l'exécution de la première instruction

Le pipeline

- Problème / aléa:
  - Exemple d'une suite d'instructions en langage C

    d=a+b;

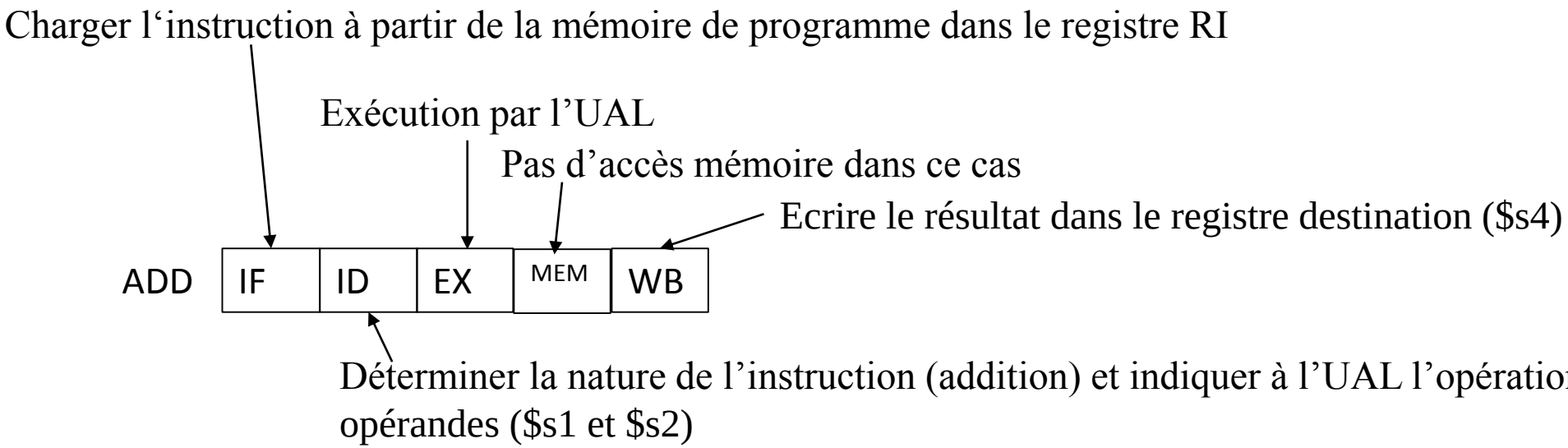
    e=d-c;

Traduction en assembleur (processeur MIPS)

On utilise les registres $s1 à $s5 pour stocker a,b,c,d et e.

ADD $s4, $s1, $s2 # $s4← $s1+$s2

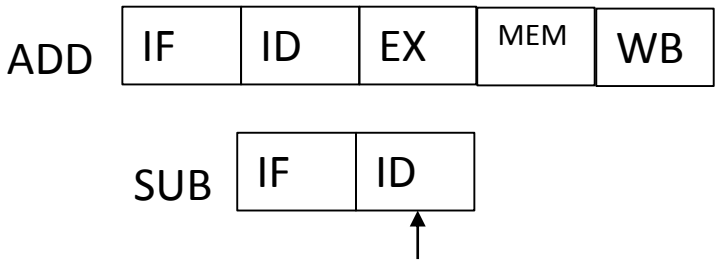
SUB $s5, $s4, $s3 # $s5← $s4-$s3



Le pipeline

- Problème / aléa:
  - Si on essaie d'exécuter les deux instructions dans le pipeline:

ADD $s4, $s1, $s2 # $s4← $s1+$s2
SUB $s5, $s4, $s3 # $s5← $s4-$s3



Déterminer la nature de l'instruction (soustraction) et indiquer à l'UAL l'opération (-) et les opérandes ($s4 et $s3)

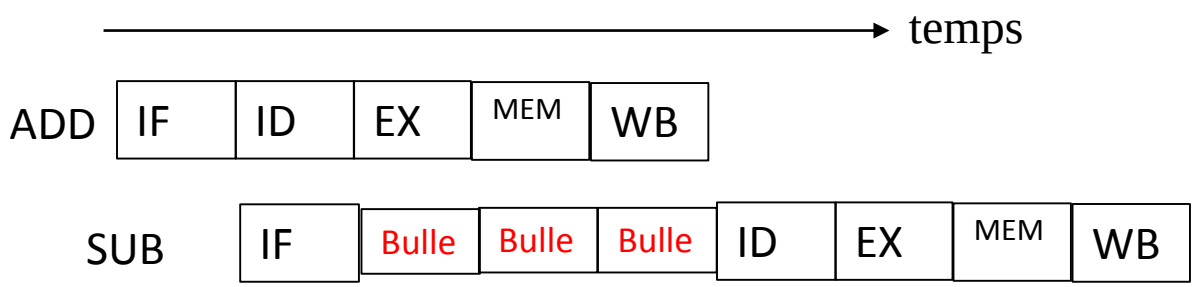
Problème: au moment du décodage de SUB, $s4 n'a pas encore changé de valeur → on ne peut pas laisser la soustraction s'appliquer sur l'ancienne valeur de $s4

Solution: retarder le pipeline pour SUB de façon à avoir son Decode après l'écriture de la nouvelle valeur de $s4
Le processeur détecte automatiquement ce type de problème et insère des « bulles » dans le pipeline

Le pipeline

- Problème / aléa:

ADD $s4, $s1, $s2 # $s4← $s1+$s2
SUB $s5, $s4, $s3 # $s5← $s4-$s3

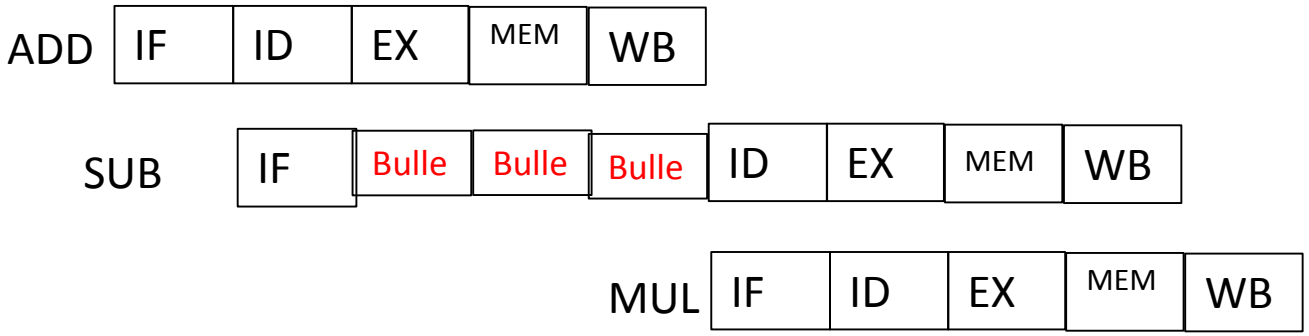


- Les bulles sont des blocages générés par le processeur

Le pipeline

- Problème / aléa:

ADD $s4, $s1, $s2 # $s4← $s1+$s2
SUB $s5, $s4, $s3 # $s5← $s4-$s3
MUL $s6,$s7,$s1 # $s6← $s7*$s1

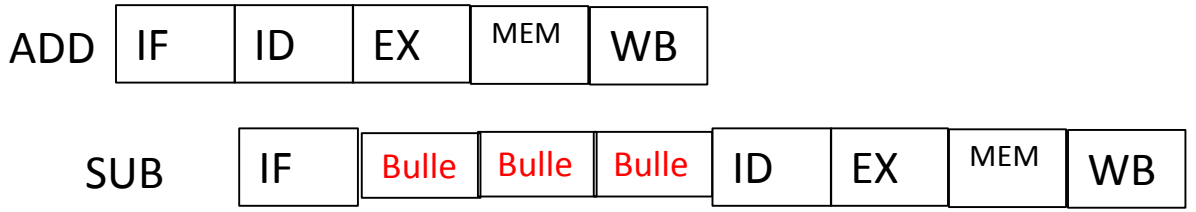


- Si une instruction est retardée, toutes les instructions suivantes sont retardées

Le pipeline

- **Différents types d'aléas**
  - I. Aléas de données
    - Deux instructions successives i et j peuvent présenter trois types d'aléa de données
      1) LAE (Lecture après Ecriture), RAW en anglais

ADD $s4, $s1, $s2
SUB $s5, $s4, $s3



Problème: SUB essaie de lire une source avant que ADD ne la modifie

Le pipeline

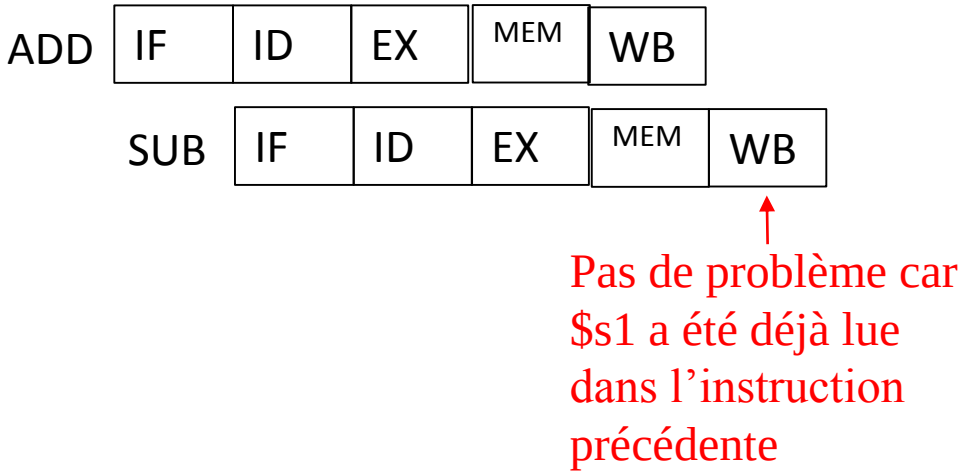
- **Différents types d'aléas**
  - I. Aléas de données
    - Deux instructions successives i et j peuvent présenter trois types d'aléa de données
      2) EAL (Ecriture après Lecture), WAR en anglais

      ADD $s4, $s1, $s2

      SUB $s1, $s5, $s3

      Problème: SUB essaie d'écrire dans une destination avant que ADD ne la lise

      → Ce problème ne peut pas arriver si on exécute les instructions dans l'ordre (ADD puis SUB) puisque l'étage ID se trouve avant l'étage WB



Le pipeline

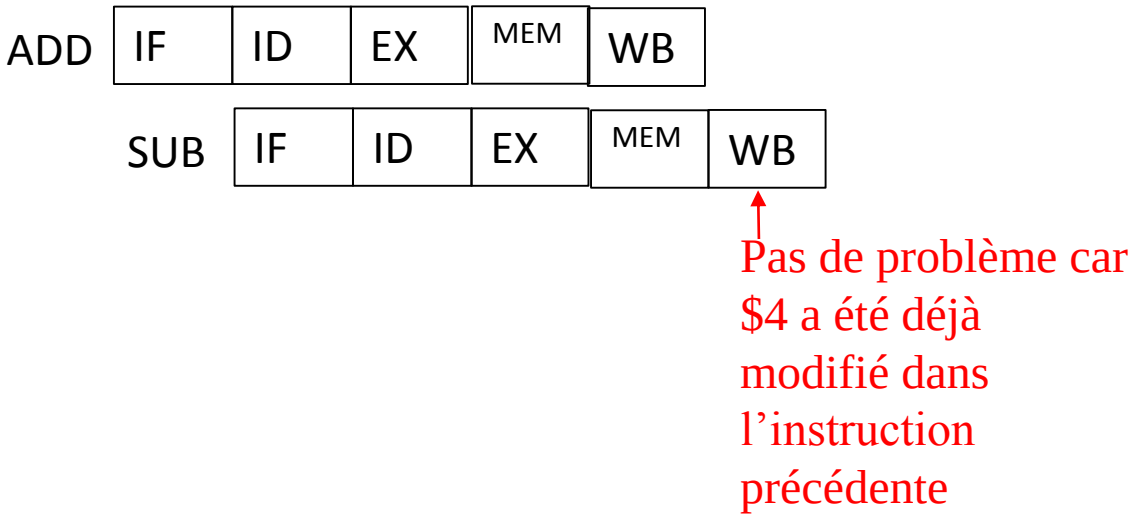
- **Différents types d'aléas**
  - I. Aléas de données
    - Deux instructions successives i et j peuvent présenter trois types d'aléa de données
      3) EAE (Ecriture après Ecriture), WAW en anglais

      ADD $s4, $s1, $s2

      SUB $s4, $s5, $s3

      Problème: SUB essaie d'écrire dans une destination avant que ADD ne l'ait fait

      → Ce problème ne peut pas arriver si les écritures se font dans l'ordre



Le pipeline

- **Différents types d'aléas**

  II. Aléas de contrôle

  Si l'instruction présente est un **branchement conditionnel**, avant que l'adresse de saut ne soit calculée, les instructions suivantes entrent déjà dans le pipeline.
  Si le saut est pris, ces instructions seraient chargées dans le pipeline pour rien → **des cycles d'horloge perdus**

Programme	Cycles d'horloge											
	1	2	3	4	5	6	7	8	9	10	11	12
Branchement	IF	ID	IE	MA	WB							
Inst. suivante n° 1		IF	ID	IE								
Inst. suivante n° 2			IF	ID								
Inst. suivante n° 3				IF								
Inst. cible n° 1						IF	ID	IE	MA	WB		
Inst. cible n° 2							IF	ID	IE	MA	WB	
Inst. cible n° 3								IF	ID	IE	MA	WB

Programme
……..
Se brancher sur **Saut** si (condition)
Inst. suivante n°1
Inst. suivante n°2
Inst. suivante n°3
Inst. suivante n°4
Inst. suivante n°5
Saut:
Inst. Cible n°1
Inst. Cible n°2
Inst. Cible n°3

Le pipeline

- **Différents types d'aléas**
  - II. Aléas de contrôle

Programme	Cycles d'horloge											
	1	2	3	4	5	6	7	8	9	10	11	12
Branchement	IF	ID	IE	MA	WB							
Inst. suivante n° 1		IF	ID	IE								
Inst. suivante n° 2			IF	ID								
Inst. suivante n° 3				IF								
Inst. cible n° 1						IF	ID	IE	MA	WB		
Inst. cible n° 2							IF	ID	IE	MA	WB	
Inst. cible n° 3								IF	ID	IE	MA	WB

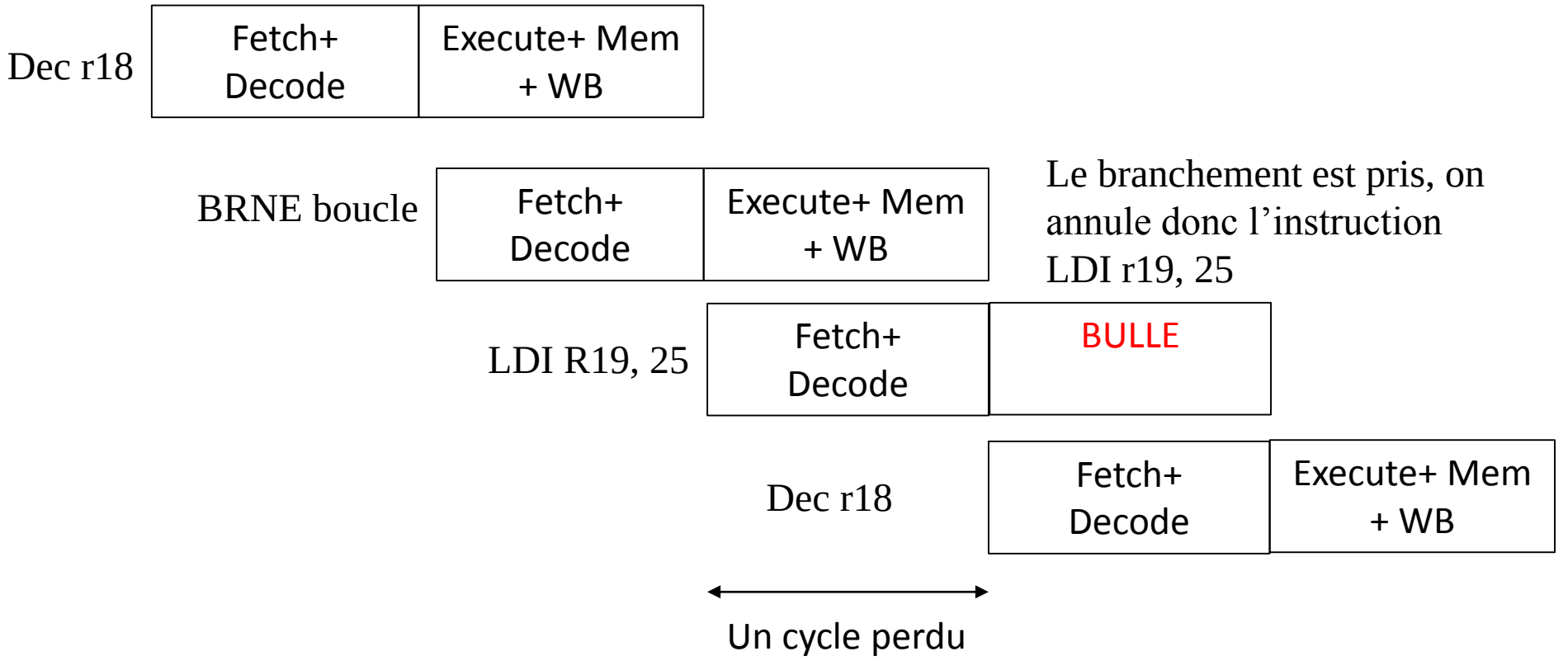
Programme
……..
Se brancher sur **Saut** si (condition)
Inst. suivante n°1
Inst. suivante n°2
Inst. suivante n°3
Inst. suivante n°4
Inst. suivante n°5
Saut:
Inst. Cible n°1
Inst. Cible n°2
Inst. Cible n°3

- Dans l'exemple, le processeur a chargé les instructions Inst. Suivante n°1, Inst. Suivante n°2 et Inst. Suivante n°3 car ce sont les instructions qui se trouvent après l'instruction de branchement dans le programme et il ne pouvait pas savoir si le branchement allait se produire ou pas
- Au cycle 5, le processeur écrit dans le registre le résultat de l'instruction de saut, à partir de ce moment le processeur a déduit qu'il fallait faire le saut et il a commencé à charger les instructions correspondantes
- **Solution:** si on arrive à **prédire les branchements**, on arrive à charger les bonnes instructions dans le pipeline (à partir du cycle 2 dans l'exemple)
- Il y a des processeurs qui sont équipés d'un système de prédiction de branchement, ce qui permet d'améliorer leurs performances

Le pipeline

- **Différents types d'aléas**
  - II. Aléas de contrôle
  - Exemple AVR 8-bits (Arduino): pipeline de 2 étages
    - Si r18= 20 → le branchement est pris

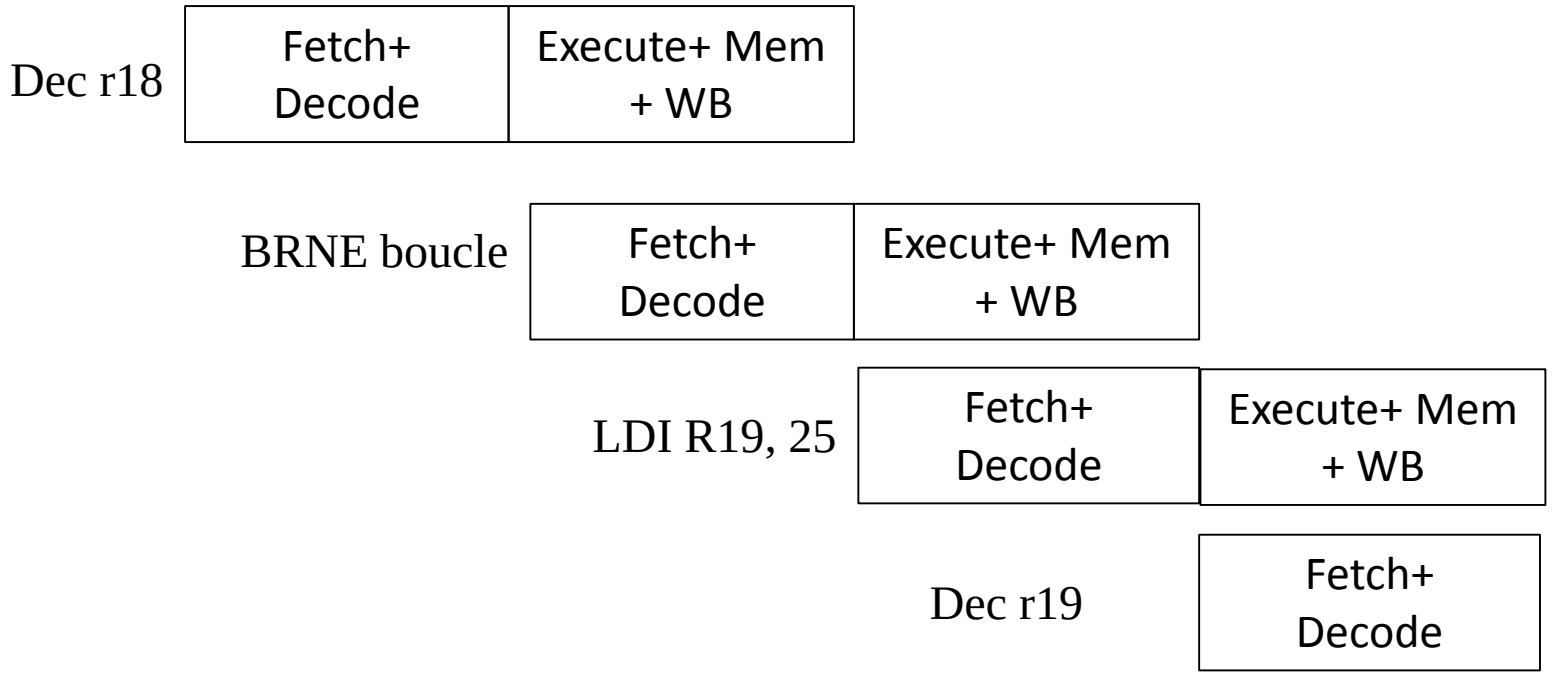
```
Ldi r18,20 //r18 ←20
Boucle:
Dec r18 // r18← r18-1
Brne boucle /*brancher sur
Boucle si ce n'est pas égal à
zéro*/
Ldi r19,25  //r19 ← 25
Dec r19 //r19 ← r19 -1
…..
```



Le pipeline

- Différents types d'aléas
  - II. Aléas de contrôle
  - Exemple AVR 8-bits (Arduino): pipeline de 2 étages
    - R18=1 → le branchement n'est pas pris

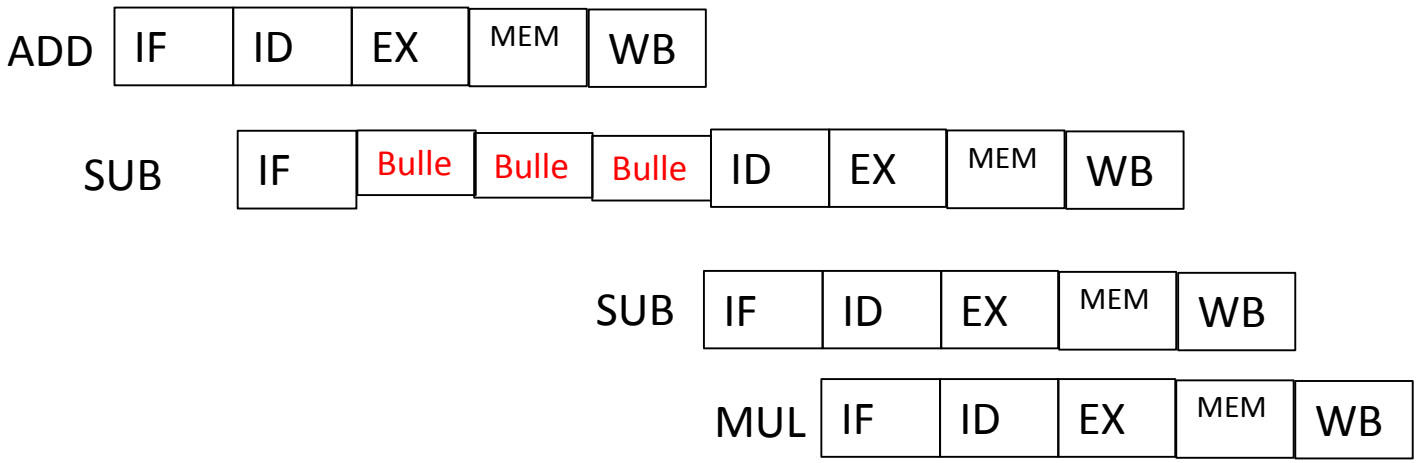
```
Ldi r18,20 //r18 ←20
Boucle:
Dec r18 // r18← r18-1
Brne boucle /*brancher sur
Boucle si ce n'est pas égal à
zéro*/
Ldi r19,25  //r19 ← 25
Dec r19 //r19 ← r19 -1
…..
```



Le pipeline

- Solution pour les aléas de données
  1) Bloquer le pipeline: Le processeur insère des bulles dans le pipeline pour retarder les instructions

ADD $s4, $s1, $s2 # d=a+b
SUB $s5, $s4, $s3 # e=d-c
SUB $s6, $s1, $s2 # f=a-b
MUL $s7,$s1,$s2 # h=a*b

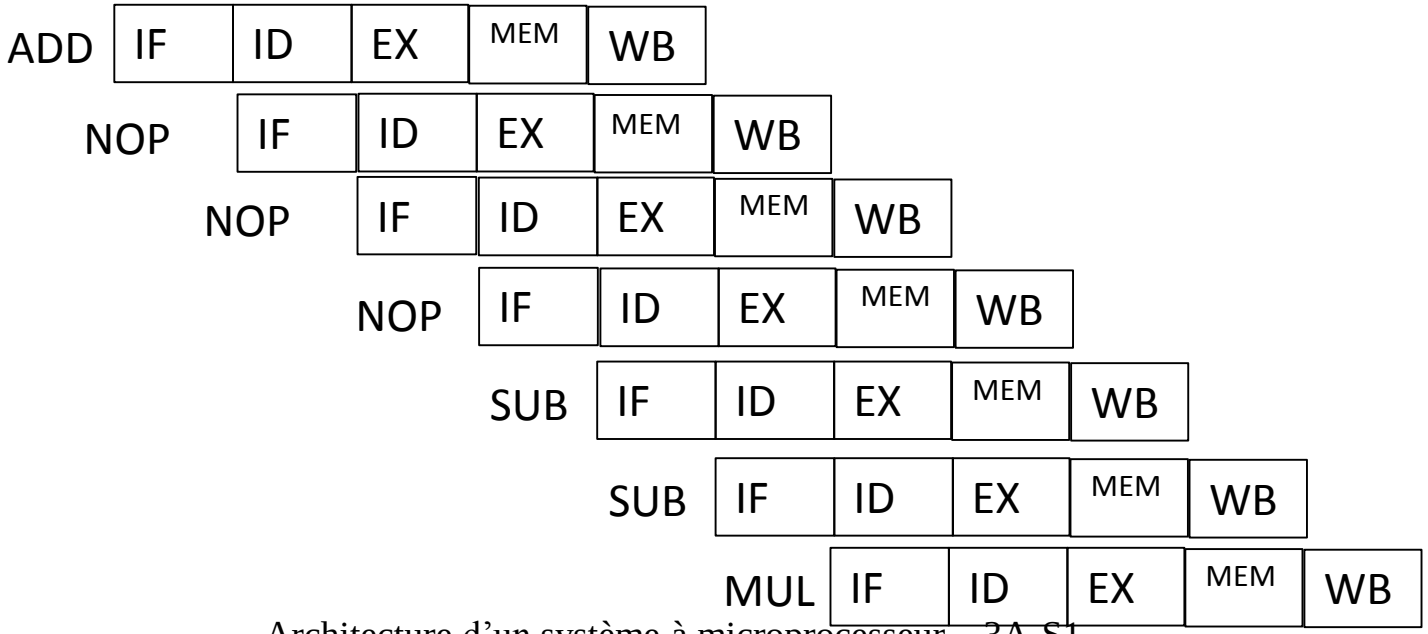


Le pipeline

- Solution pour les aléas de données
  2) Insérer des opérations **NOP** (No Operation) entre deux instructions dépendantes
     - Les opérations NOP peuvent être insérées:
       - Par le **programmeur**
       - Automatiquement par le **compilateur** quand il détecte une dépendance de données entre deux instructions

```
ADD $s4, $s1, $s2  # d=a+b
NOP
NOP
NOP
SUB $s5, $s4, $s3  # e=d-c
SUB $s6, $s1, $s2  # f=a-b
MUL $s7,$s1,$s2 # h=a*b
```

- Cette solution évite un blocage du pipeline mais ne permet pas de diminuer le temps d'exécution (intéressante si le processeur n'est pas capable de bloquer le pipeline)



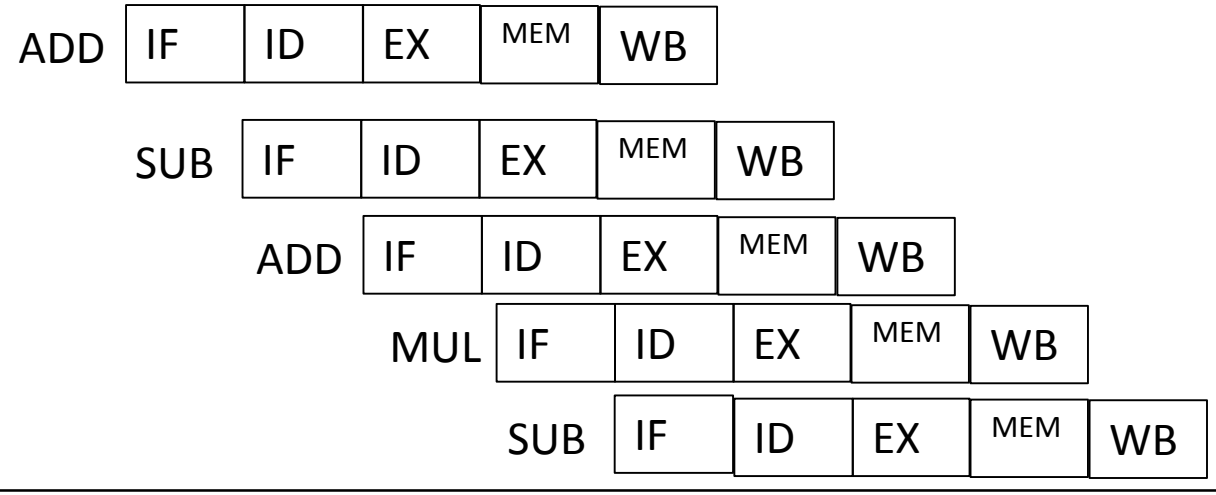
Le pipeline

- Solution pour les aléas de données
  3) Réarranger les instructions de façon à insérer du code entre deux instructions dépendantes
     - Ce travail peut être fait:
       - Par le **programmeur**
       - Automatiquement par le **compilateur** quand il détecte une dépendance de données entre deux instructions

```
ADD $s4, $s1, $s2  # d=a+b
SUB $s5, $s4, $s3  # e=d-c
SUB $s6, $s1, $s2  # f=a-b
ADD $s8, $s1, $s2 # i=a+b
MUL $s7,$s1,$s2 # h=a*b
```

Programme avec un aléa de données

```
ADD $s4, $s1, $s2  # d=a+b
SUB $s6, $s1, $s2  # f=a-b
ADD $s8, $s1, $s2 # i=a+b
MUL $s7,$s1,$s2 # h=a*b
SUB $s5, $s4, $s3  # e=d-c
```



Solution en réarrangeant les instructions : Cette solution évite un blocage du pipeline et permet de diminuer le temps d'exécution

Le pipeline

- Le pipeline peut avoir des aléas de données
  - Solutions selon la complexité des processeurs
    - Il y a des processeurs incapables de bloquer le pipeline, des opérations NOP permettent de le faire
    - Si le processeur est capable de bloquer le pipeline, il va générer des bulles
    - Il y a des processeurs capables de détecter les dépendances et d'exécuter les instructions dans un ordre différent de celui du programme
    - Si le processeur n'est pas capable de détecter les dépendances et d'exécuter les instructions dans un ordre différent de celui du programme → le programmeur/le compilateur change l'ordre des instructions

Le pipeline

- Le pipeline peut avoir des aléas de contrôle
  - Solutions selon la complexité des processeurs
    - Si le processeur n'est pas capable de prédire les branchements, il va se contenter d'annuler les instructions chargées inutilement dans le pipeline si le saut est pris
    - Si le processeur est capable de prédire les branchements il peut éviter les aléas de contrôle

Le pipeline

- Les processeurs Intel

Year	Micro-architecture	Pipeline stages	max. Clock	Tech process
1989	486 (80486)	3	100 MHz	1000 nm
1993	P5 (Pentium)	5	300 MHz	600 nm
1995	P6 (Pentium Pro; later Pentium II)	14 (17 with load & store/retire)	450 MHz	350 nm
1999	P6 (Pentium III) (Copper Mine)	12 (15 with load & store/retire)	1400 MHz	250 nm
2000	NetBurst (Pentium 4) (Willamette)	20 unified with branch prediction	2000 MHz	180 nm
2002	NetBurst (Pentium 4) (Northwood, Gallatin)		3466 MHz	130 nm
2003	Pentium M	10 (12 with fetch/retire)	2133 MHz	130 nm
2004	NetBurst (Pentium 4) (Prescott)	31 unified with branch prediction	3800 MHz	90 nm
2006	Intel Core	12 (14 with fetch/retire)	3000 MHz	65 nm
2007	Penryn		3333 MHz	45 nm
2008	Nehalem **Core i3, Core i5, Core i7**	20 unified (14 without miss prediction)	3600 MHz	
	Bonnell	16 (20 with prediction miss)	2100 MHz	
2010	Westmere	20 unified (14 without miss prediction)	3730 MHz	32 nm
2011	*Saltwell*	16 (20 with prediction miss)	2130 MHz	
	Sandy Bridge **Core i3, Core i5, Core i7**	14 (16 with fetch/retire)	4000 MHz	
2012	Ivy Bridge		4100 MHz	22 nm
2013	*Silvermont*	14-17 (16-19 with fetch/retire)	2670 MHz	
	Haswell	14 (16 with fetch/retire)	4400 MHz	
2014	Broadwell		3700 MHz	14 nm
2015	*Airmont*	14-17 (16-19 with fetch/retire)	2640 MHz	
	Skylake **Core i9**	14 (16 with fetch/retire)	4200 MHz	
2016	*Goldmont*	20 unified with branch prediction	2600 MHz	
	Kaby Lake	14 (16 with fetch/retire)	4500 MHz	
2017	Coffee Lake **Core i9**		5000 MHz	
	Goldmont Plus	? 20 unified with branch prediction ?	2800 MHz	
2018	Cannon Lake	14 (16 with fetch/retire)	3200 MHz	10 nm
	Whiskey Lake		4600 MHz	14 nm
	Amber Lake		4200 MHz	
(2018)	Cascade Lake		? MHz	
(2019)	Cooper Lake			

https://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures

Le pipeline

- Les processeurs ARM

  - ARM jusqu'à la version 7 (jusqu'à 2001): 3 étages
  - ARM 8-9 (1996- 2006): 5 étages
  - ARM 10 (2000-2002): 6 étages
  - ARM 11 (2002-2005): 8-9 étages
  - Cortex A8 (2005): 13 étages
  - Cortex A15 (2010): 15-25 étages
  - Cortex A7 (2011): 8-10 étages
  - Cortex-A35 (2014): 8 étages
  - Cortex-A75 (2017): 11-13 étages

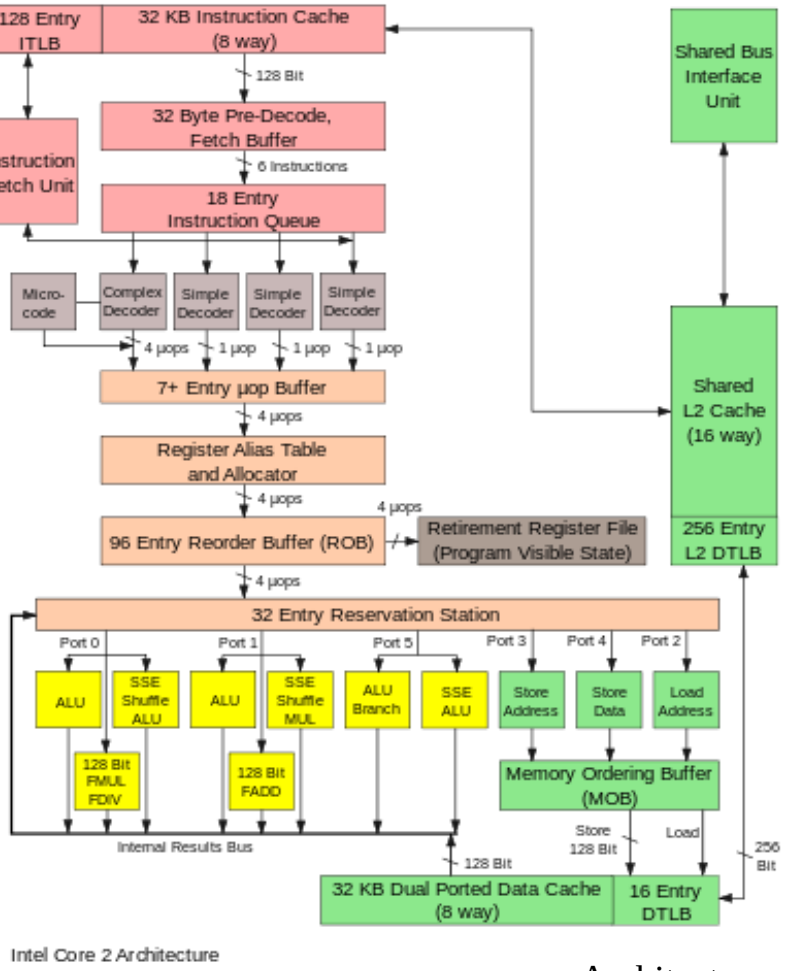
https://en.wikipedia.org/wiki/List_of_ARM_microarchitectures

Le pipeline et les unités de calculs

- Dans les processeurs modernes, chaque processeur/cœur a plusieurs unités de calcul

→Possibilités de traiter plusieurs instructions en même temps selon la nature de l'instruction

→Plusieurs pipelines → accélération du temps d'exécution



Intel Core 2 Architecture

Exemple d'architecture d'un intel dual core

Unités de calcul :

- Plusieurs ALU (calcul sur les entiers et en virgule fixe)
- Plusieurs FPU (Floating Point Units: calcul sur les flottants)
- Plusieurs SSE (Streaming SIMD Extensions)
- etc.

Les multi-coeurs

- Avantage: Plusieurs cœurs physiques fonctionnant simultanément
  →accélération de l'exécution des programmes

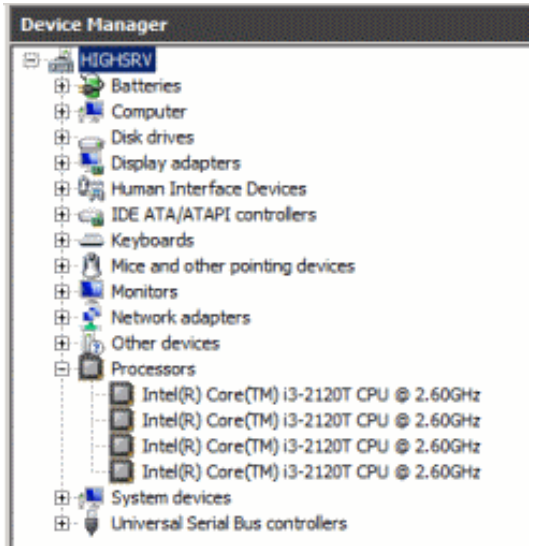
- Complexité

  - Logicielle: les programmes doivent être écrits avec des bibliothèques dédiées à la programmation parallèle pour exploiter pleinement les caractéristiques de ces processeurs.

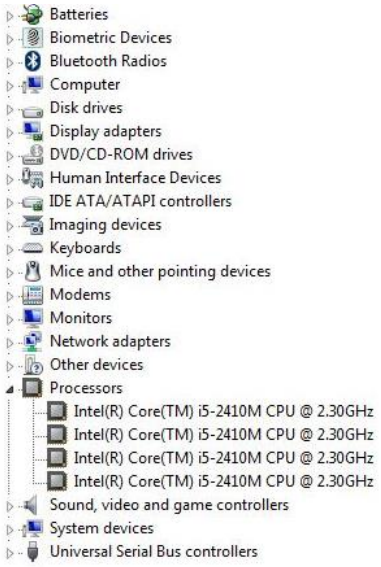
  - Matérielle: besoin de gérer les données partagées par les cœurs pour vérifier que chaque cœur ait les bonnes valeurs de données utilisées par les autres cœurs

Les multi-coeurs

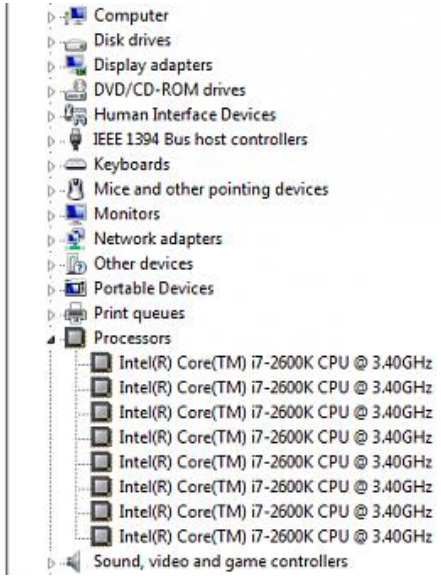
- Exemples: Les core i3, core i5, core i7
  - Différences
    - La taille de cache: i3 a moins de cache que i5, qui a moins de cache que i7
    - Le hyper-threading: i3 offre deux cœurs et supporte le hyper-threading (2x2=4 threads en même temps), i5 pour les ordinateurs de bureau offre 4 cœurs et ne supporte pas le hyper-threading (1x4= 2x2=4 threads en même temps), i7, pour la plupart des modèles pour ordinateurs de bureau, a 4 cœurs et supporte le hyper-threading (2x4=8 threads en même temps)



Core i3



Core i5



Core i7

Single-threading, Multi-threading, SMT et hyper-threading

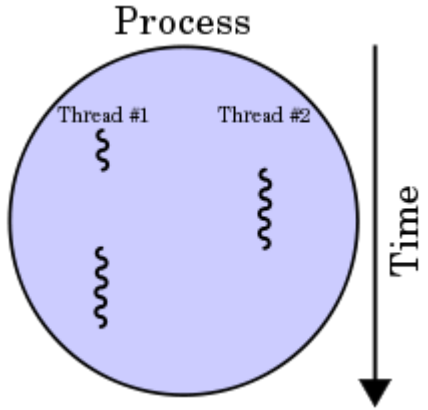
- Single-threading
  - Chaque processus (programme) contient un seul thread
  - Un processus est un programme en cours d'exécution par le processeur
  - Le système d'exploitation ordonnance les processus de façon à donner l'impression que tous les programmes sont en train de s'exécuter en même temps

  - Inconvénient: si une instruction d'un programme donné se bloque en attendant que la donnée à manipuler arrive de la mémoire par exemple, le programme se bloque

  → perte de temps

Single-threading, Multi-threading, SMT et hyper-threading

- Multi-threading
  - La possibilité de transformer un processus/programme en un ensemble de **threads indépendants** pour pouvoir faire des alternances entre les threads dans le but d'accélérer l'exécution

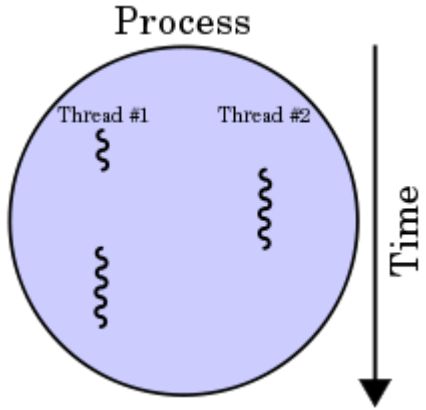


Threads logiciels

  - Pour cela, il faut écrire des programmes multi-threadés. Il y a des bibliothèques (C++, java, etc.) pour cela.

Single-threading, Multi-threading, SMT et hyper-threading

- Multi-threading
  - Deux types de multi-threading
    1) A Gros grain (Coarse-grain): Un thread s'exécute jusqu'à ce qu'il se bloque, au lieu d'attendre sans rien faire, le processeur peut exécuter un autre thread, le thread bloqué peut reprendre son exécution dès qu'il a les ressources nécessaires



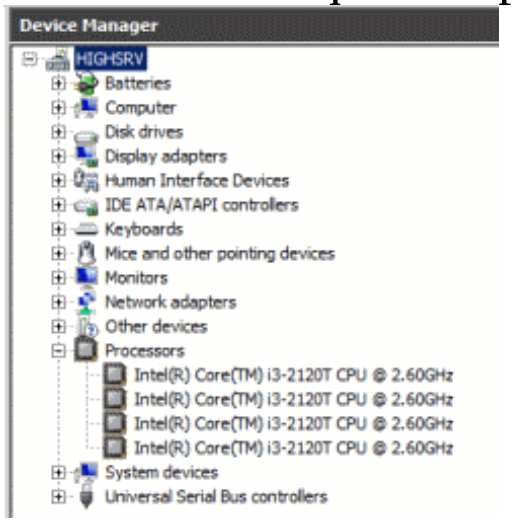
    2) A grain fin (Fine-grain): l'exécution est alternée: exemple: 1 instruction de thread 1 suivie d'une instruction de thread2, suivie d'une instruction de thread 1, etc.

  - Pour un cycle d'horloge donné, sur chaque processeur/cœur il y a qu'un seul thread qui est exécuté

Single-threading, Multi-threading, SMT et hyper-threading

- SMT (Simultaneous Multi-threading) et hyper-threading
  - Sur un même cycle d'horloge des instructions provenant de différents threads peuvent s'exécuter en parallèle
- Hyper-Threading: c'est la technologie Intel pour le SMT

- Support matériel: chaque cœur/processeur contient des threads matériels (deux généralement) qu'on appelle aussi processeurs logiques ayant chacun des registres d'état différents, entre autres, pour gérer plusieurs threads en parallèle
- Le hyper-threading sur un seul cœur nécessite la réplication de moins de ressources que la réplication d'un cœur entier → le hyper-threading est moins cher que la réplication des cœurs.



Le processeur est dual-core
Chaque cœur contient deux threads matériels

Multi-threading et Multi-coeurs

- Sur un système multi-cœurs, chaque cœur peut exécuter un thread séparément en parallèle avec les threads des autres cœurs

- Un cœur peut supporter le multi-threading et le hyper-threading

- Si un cœur contient des threads matériels, des threads logiciels peuvent être exécutés séparément sur chaque thread matériel (hyper-threading)

Bibliographie

- Les architectures RISC/CISC

  https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

- Exemples d'architectures de processeurs

  http://aelmahmoudy.users.sourceforge.net/electronix/arm/chapter4.htm

  http://www.edwardbosworth.com/My5155_Slides/Chapter09/MicroprogrammingHistory.htm