Circuits logiques

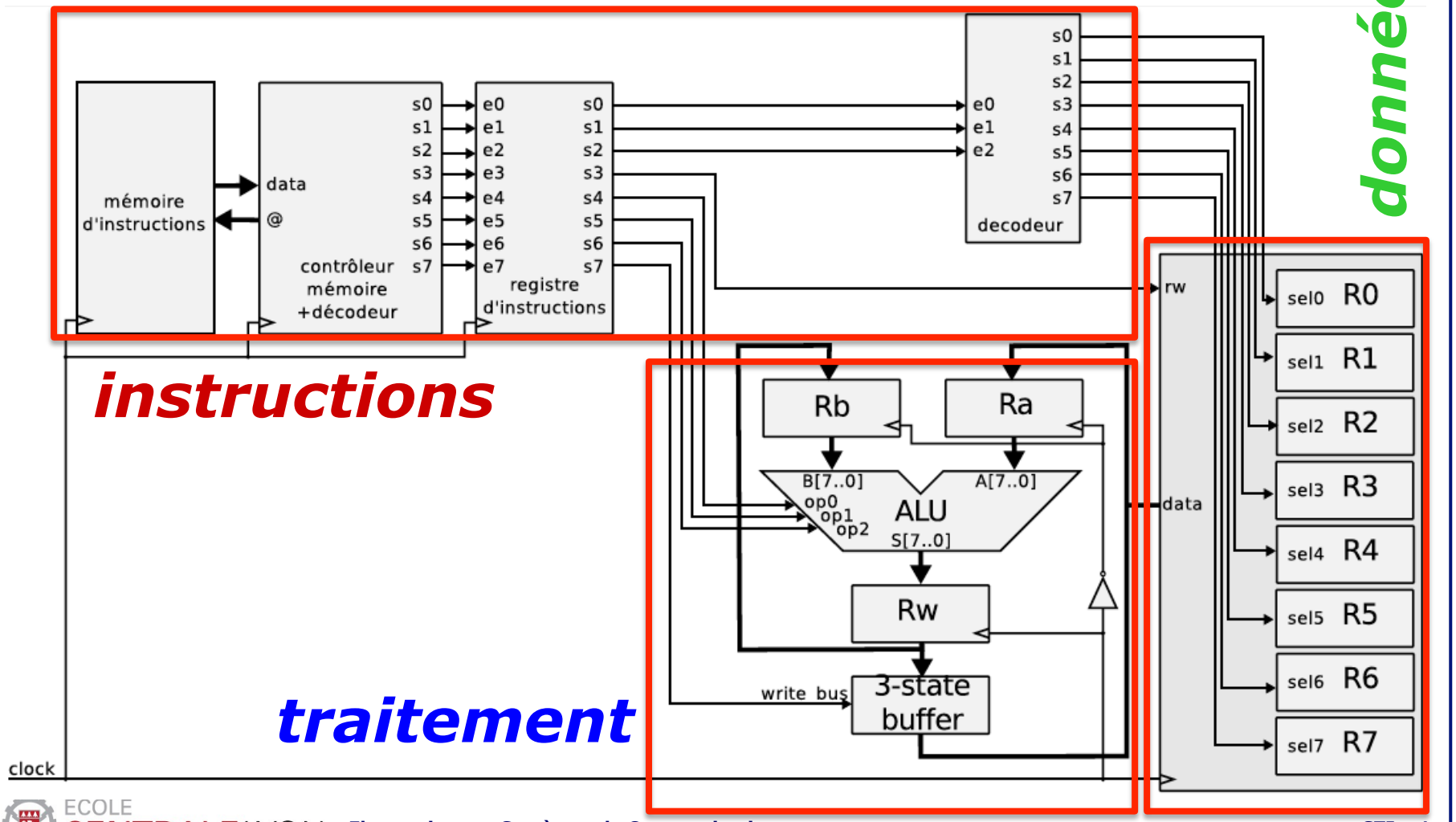
Architectures de microprocesseurs

Plan

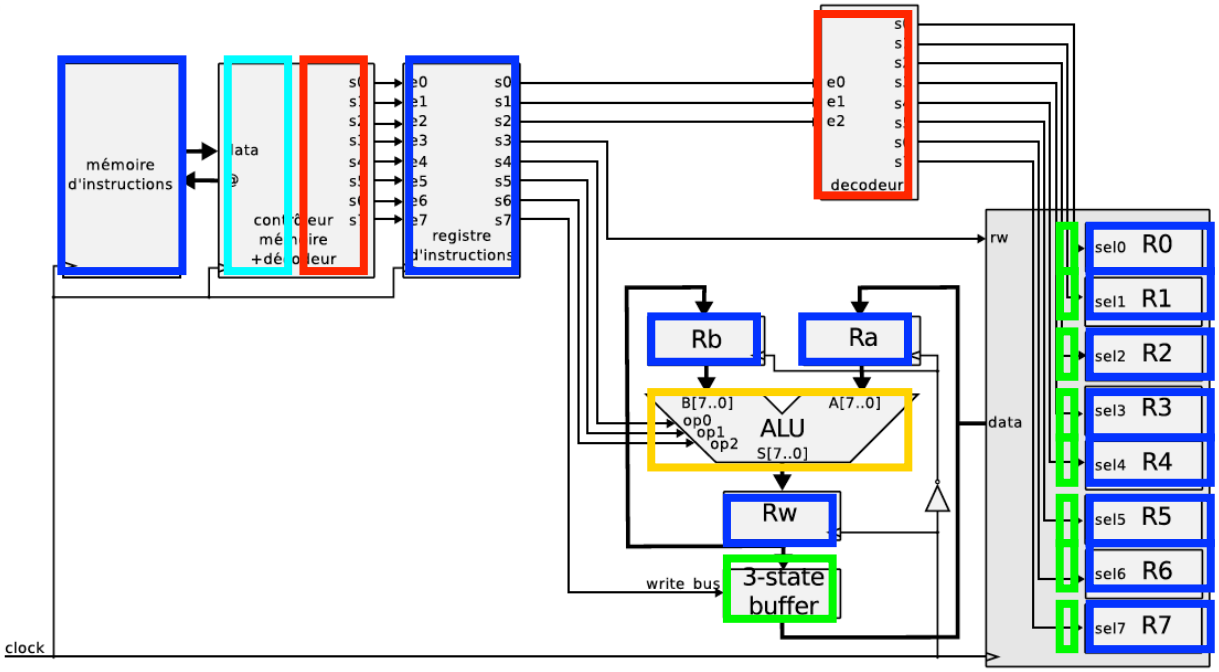
- Structure d'un microprocesseur
- Principe de fonctionnement
- Présentation de la platine TP
- Programmation
- Architectures récentes
- Enseignements proposés en S7, S8 et 39

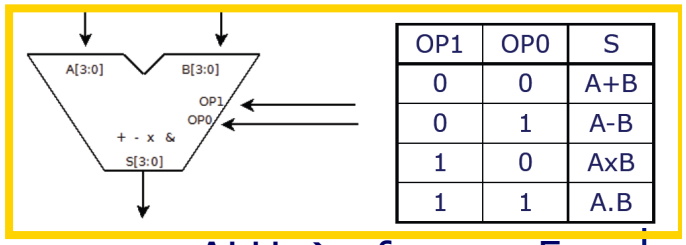
Structure d'un microprocesseur simplifié

- le microprocesseur *traite* des *données* selon un programme *d'instructions*



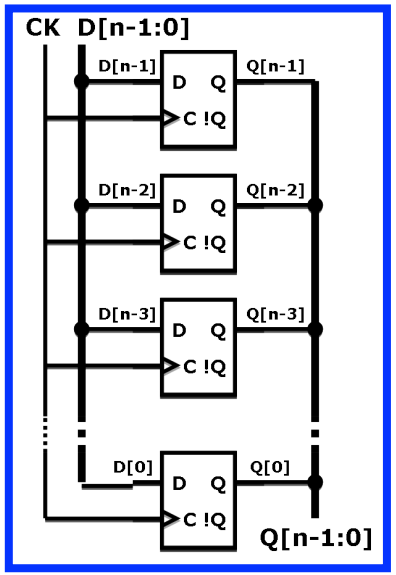
Quelques éléments de base



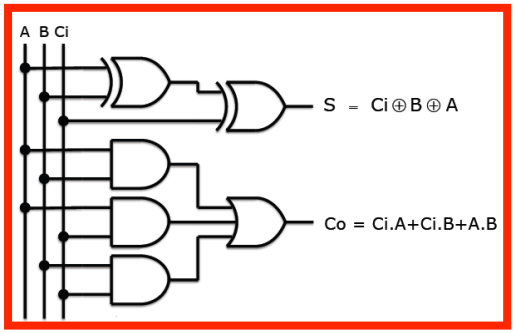


OP1	OP0	S
0	0	A+B
0	1	A-B
1	0	AxB
1	1	A.B

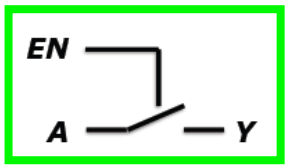
ALU → cf cours 5



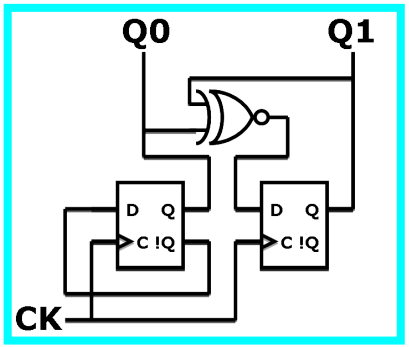
n bascules D,
avec ou sans LD
→ cf cours 6



Logique combinatoire
→ cf cours 5

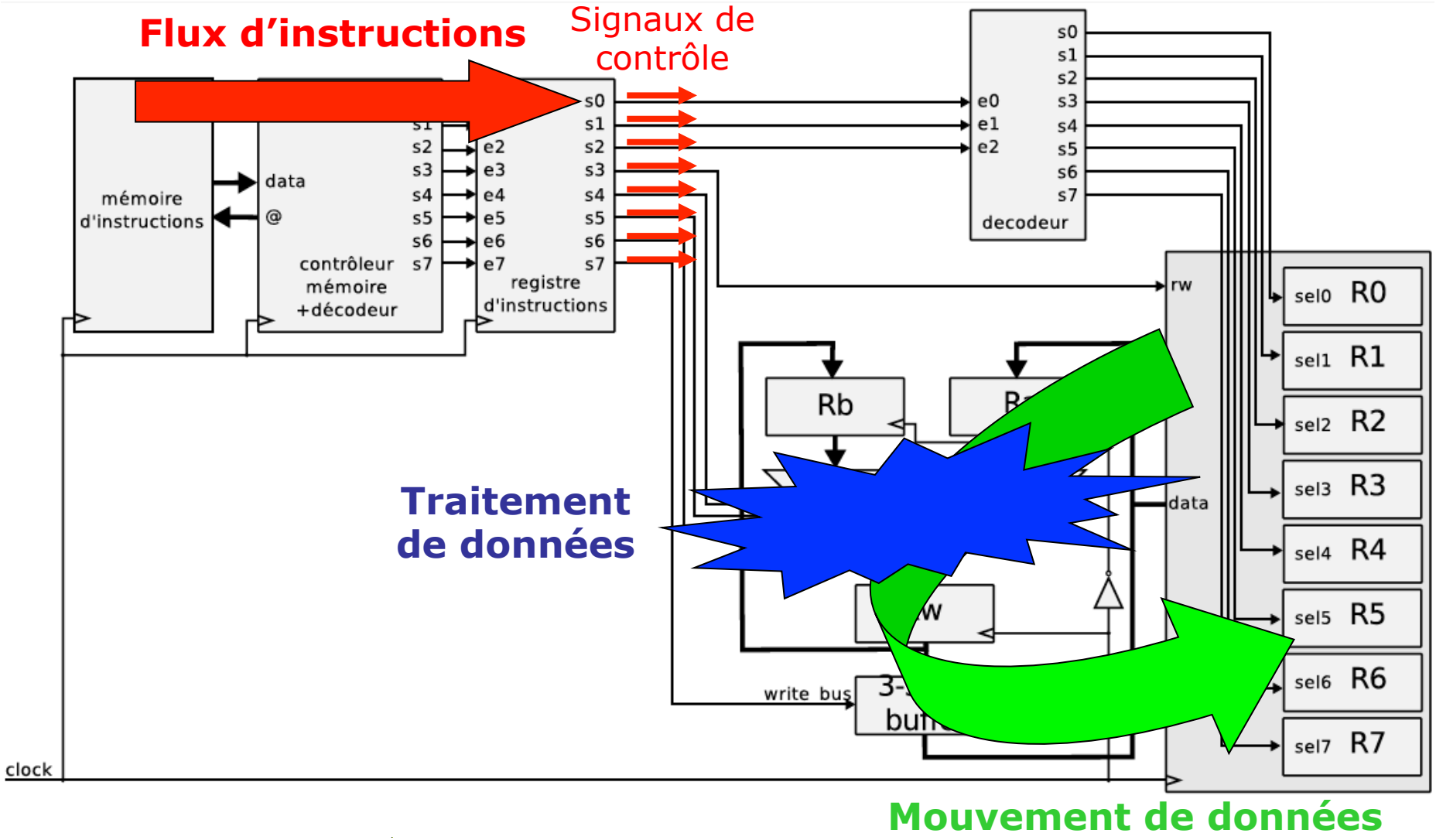


Buffer 3 états
→ cf cours 5

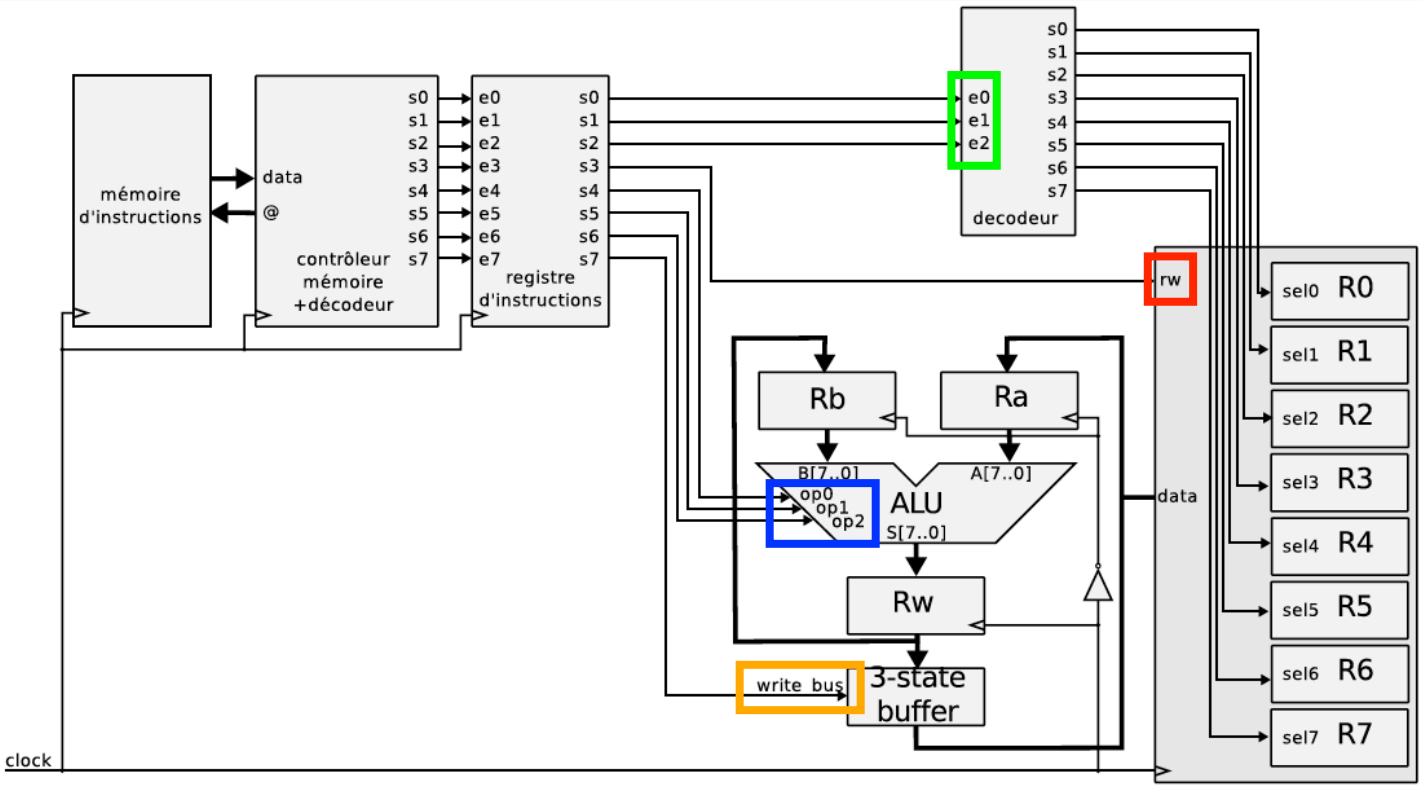


Compteur
→ cf cours 6

Logistique de fonctionnement

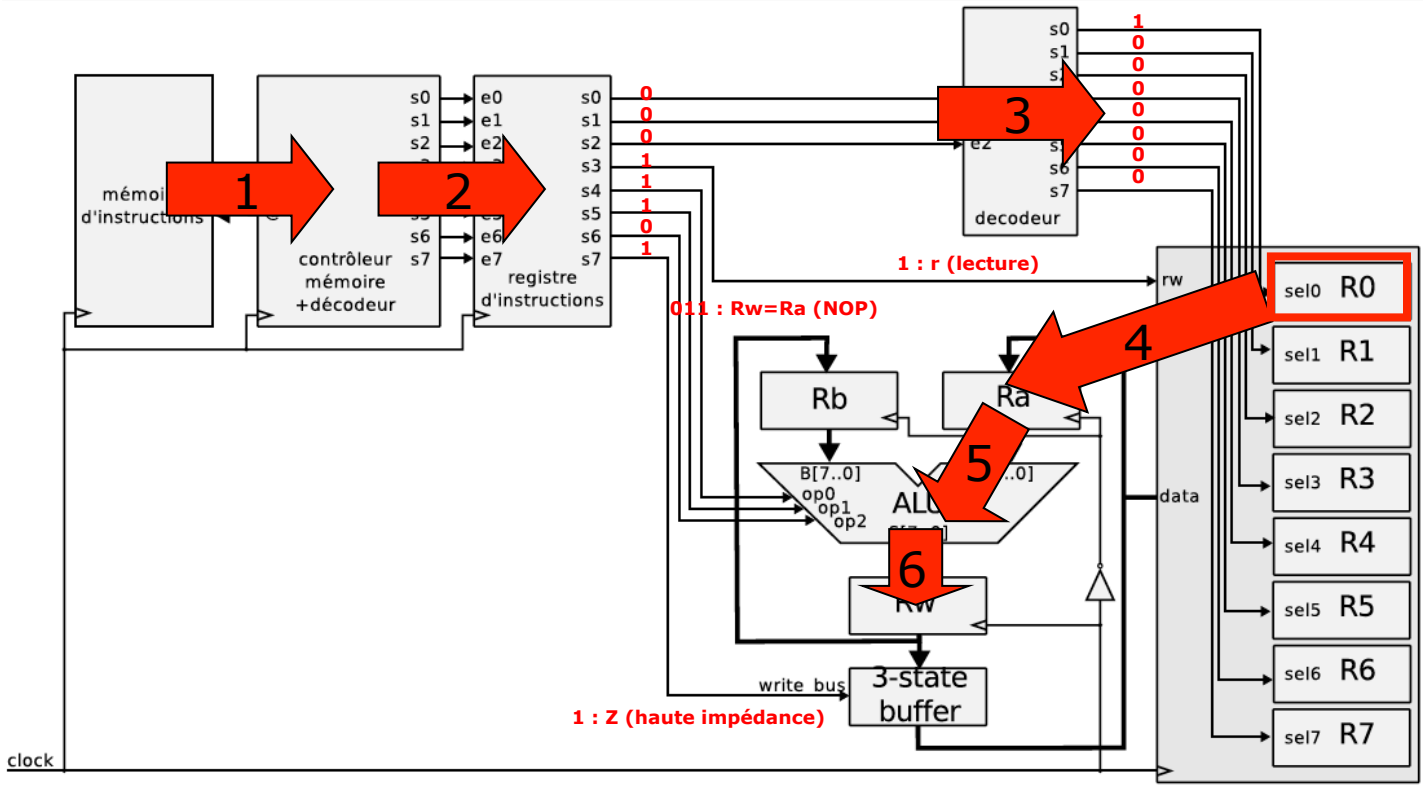


Jeu d'instructions



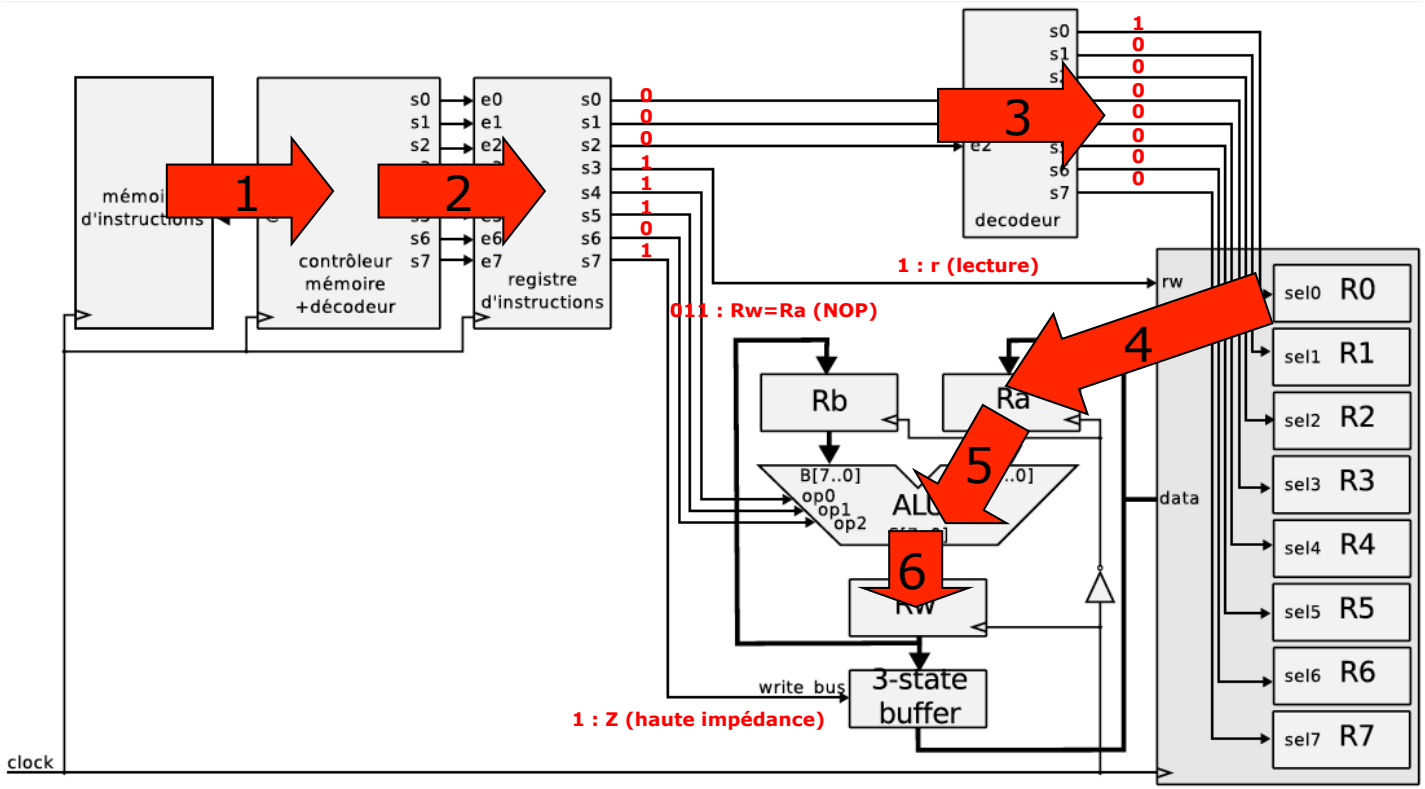
$\overline{WriteBus}$	OP_2 OP_1 OP_0	$R\overline{W}$	e_2 e_1 e_0
0 Écriture **1** Haute impédance	**000 ADDI** **001 SUB** **010 MULT** **011 W=A (NOP)** **100 DIV** **101 !A** **110** *A+B+1* **111** *Réservé*	Sens de communication avec les registres R_0 à R_7 **1** lecture **0** écriture	**000** Registre 0 **001** Registre 1 **010** Registre 2 **011** Registre 3 **100** Registre 4 **101** Registre 5 **110** Registre 6 **111** Registre 7

Exemple: exécution de l'instruction LD(R0)



- 1. Lecture instruction et incrémentation compteur ordinal
- 2. Décodage instruction et placement dans IR
- 3. Décodage registre: LD (R0)
- 4. Transfert du contenu de R0 vers Ra
- 5. Exécution de l'instruction (NOP)
- 6. Stockage du résultat dans Rw

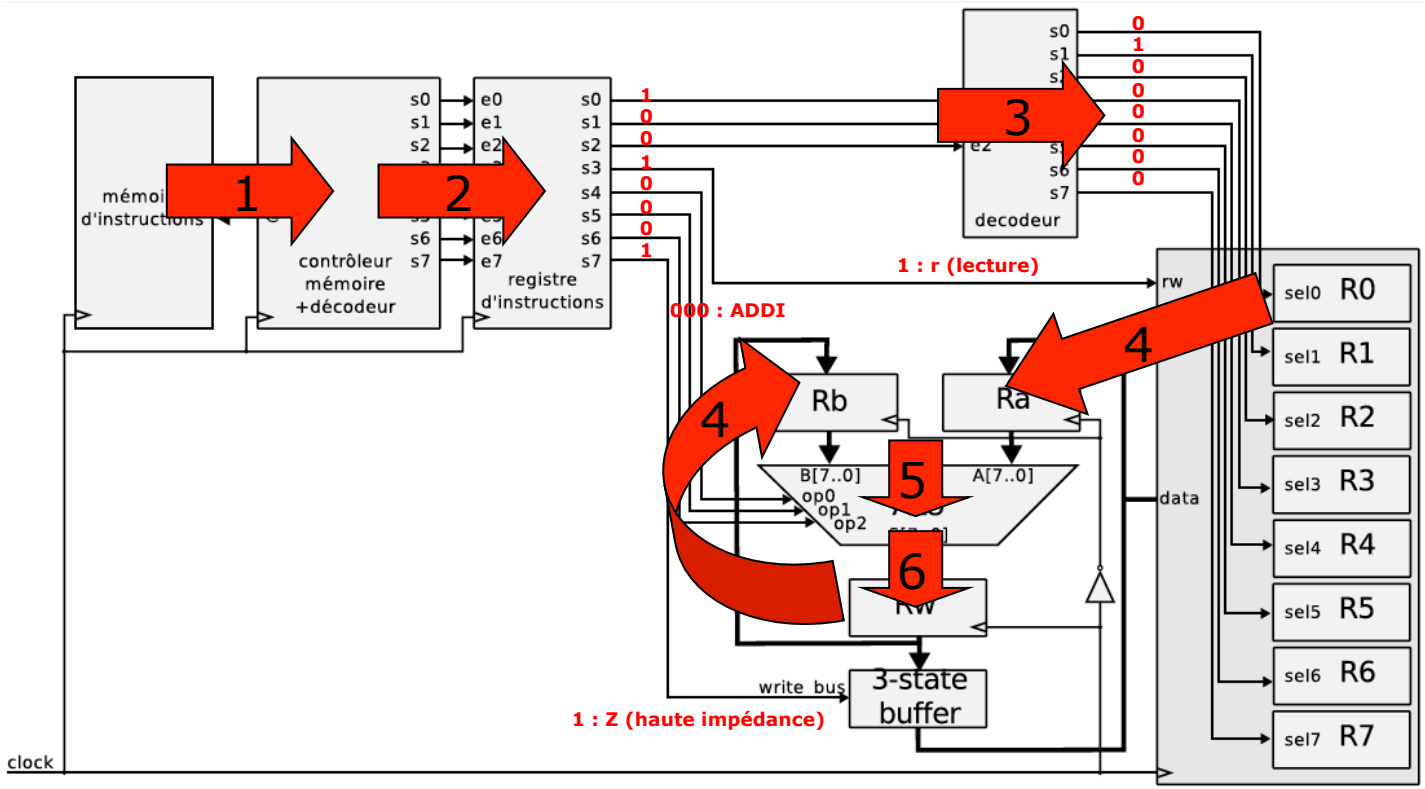
Exemple: exécution de l'instruction LD(R0)



LD (R0) → $(10111000)_2 = (B8)_{16}$

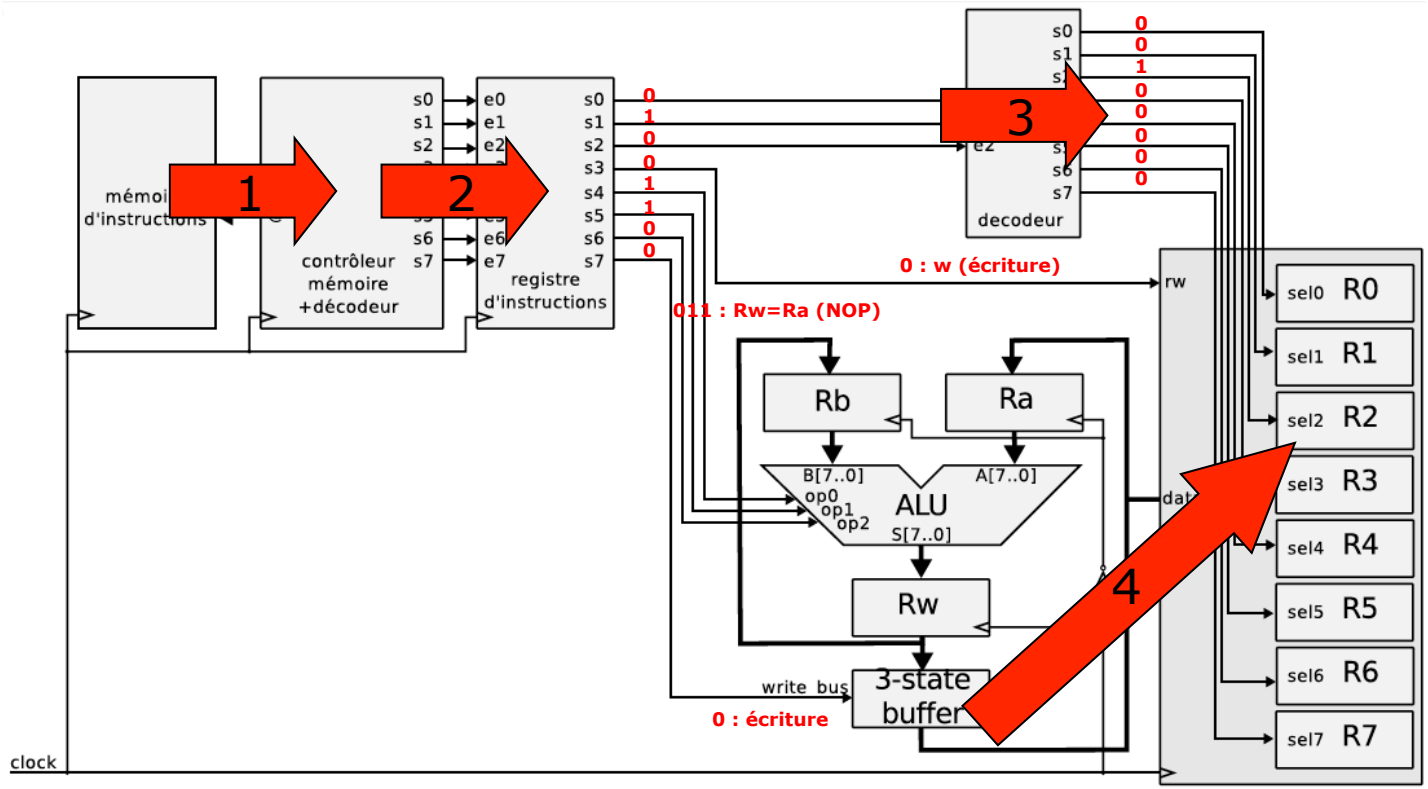
$\overline{WriteBus}$	OP_2	OP_1	OP_0	$R\overline{W}$	e_2	e_1	e_0
0 Écriture **1** Haute impédance	**000 ADDI** **001 SUB** **010 MULT** **011 W=A (NOP)** **100 DIV** **101 !A** **110** *A+B+1* **111** *Réservé*			Sens de communication avec les registres R_0 à R_7 **1** lecture **0** écriture	**000** Registre 0 **001** Registre 1 **010** Registre 2 **011** Registre 3 **100** Registre 4 **101** Registre 5 **110** Registre 6 **111** Registre 7		

ADD(R1) : $(10001001)_2$: $(89)_{16}$



- 1. Lecture instruction et incrémentation compteur ordinal
- 2. Décodage instruction et placement dans IR
- 3. Décodage registre: ADD (R1)
- 4. Transfert du contenu de R1 vers Ra et de Rw vers Rb
- 5. Exécution de l'instruction (ADD)
- 6. Stockage du résultat dans Rw

SW(R2) : $(00110010)_2$: $(32)_{16}$



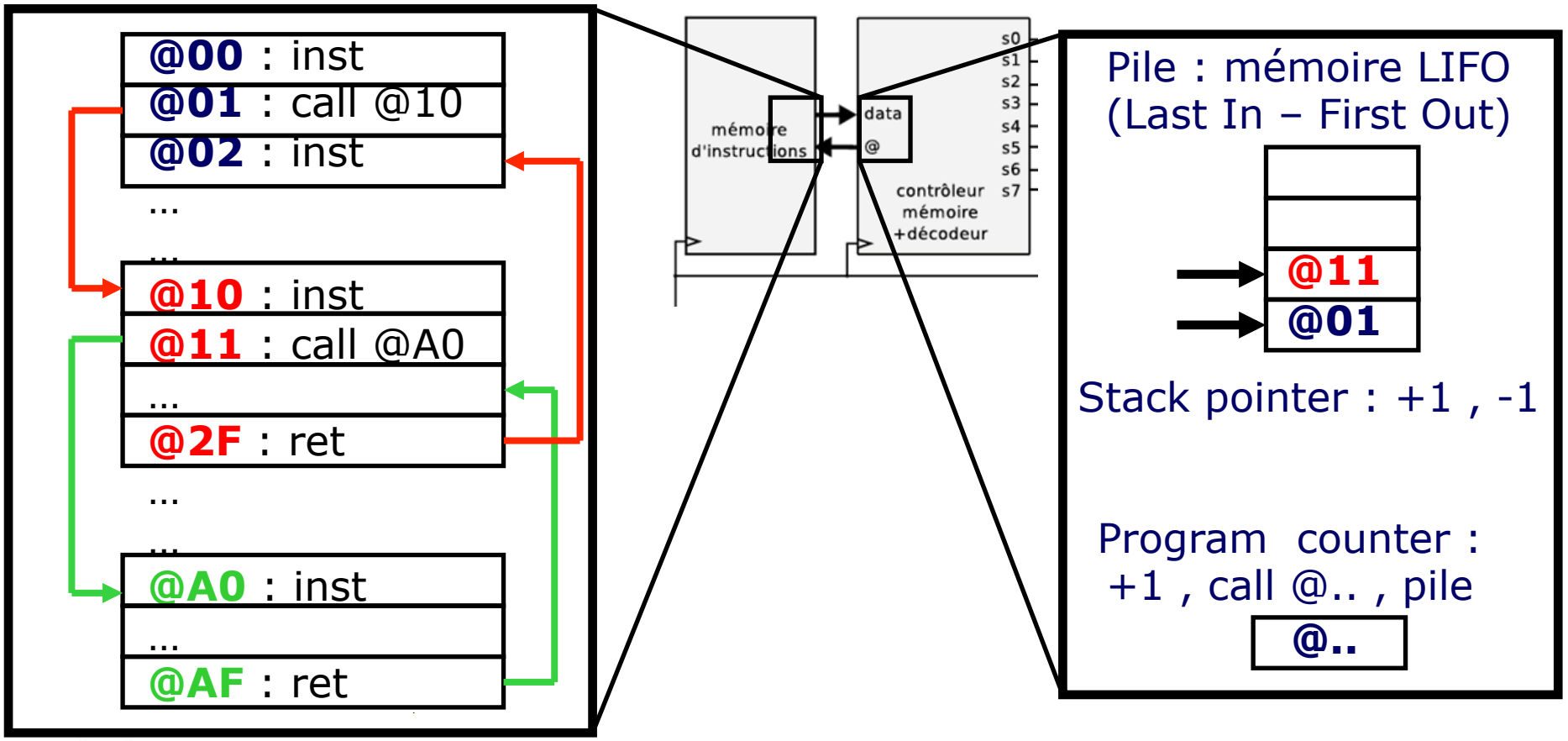
- 1. Lecture instruction et incrémentation compteur ordinal
- 2. Décodage instruction et placement dans IR
- 3. Décodage registre: SW (R2)
- 4. Transfert du contenu de Rw vers R2

Programme

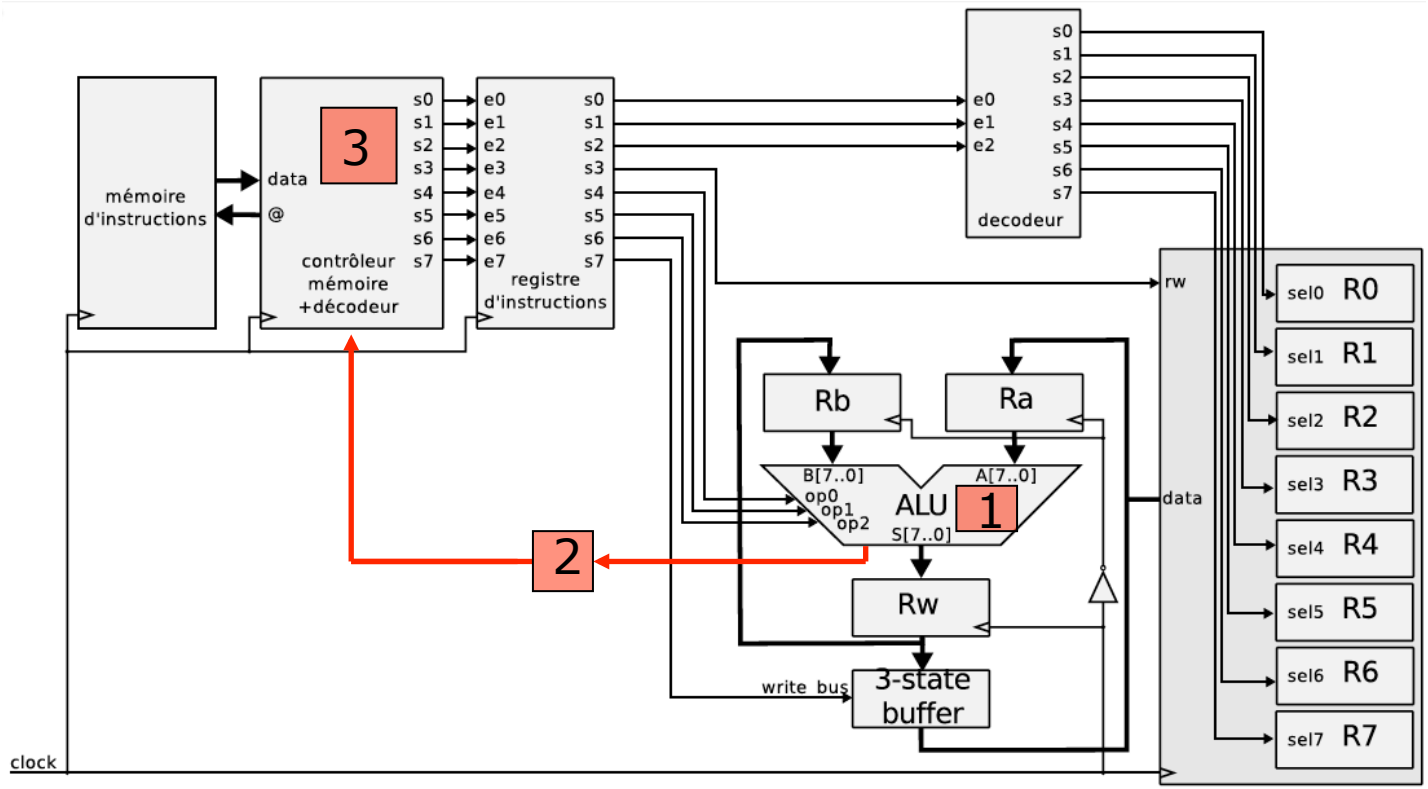
- Programme = suite d'instructions (code hexadécimal) stockées en mémoire
  - Ex:
    - @00 : B8
    - @01 : 89
    - @02 : 32
    - …
- Mais le microprocesseur simplifié ne supporte pas :
  - Décisions (if … then … else)
  - Boucles (for, while)
  - Opérations « complexes » (mult, div, sqrt, ln …)
  - Formats de nombre précis (virgule flottante)

Exécution d'un programme

- Un programme réaliste
  - Appel de fonction (branchement, goto)
  - Registres d'états
  - Donc architectures plus complexes



Une extension possible

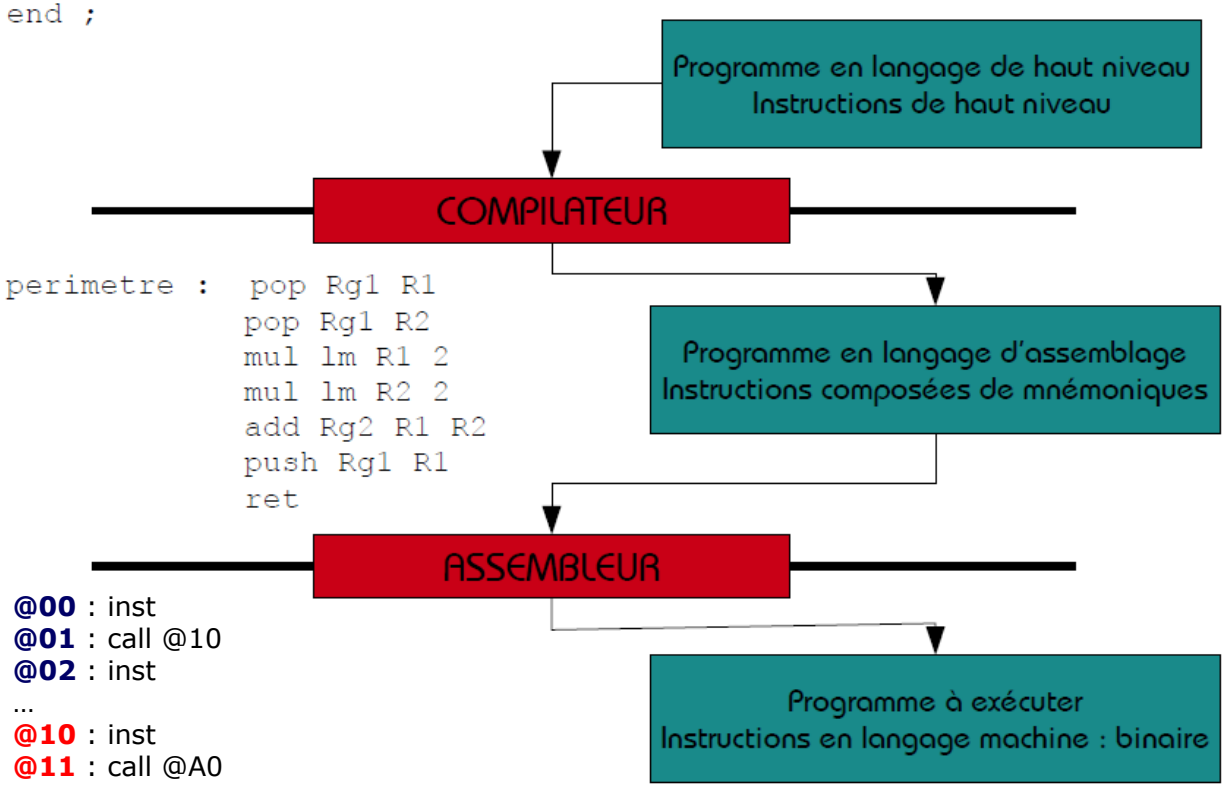


- Gestion du branchement conditionnel (e.g. instruction IF … THEN … ELSE …):
  - 1 : ALU → ajout d'un opérateur de comparaison
  - 2 : registre d'état dédié au stockage des résultats de comparaisons
  - 3 : décodeur → lecture du registre et gestion de la pile d'instructions

La programmation en résumé

- Programmation à partir de langages « évolués » (e.g. C); abstraction de l'architecture matérielle
- Exemple :

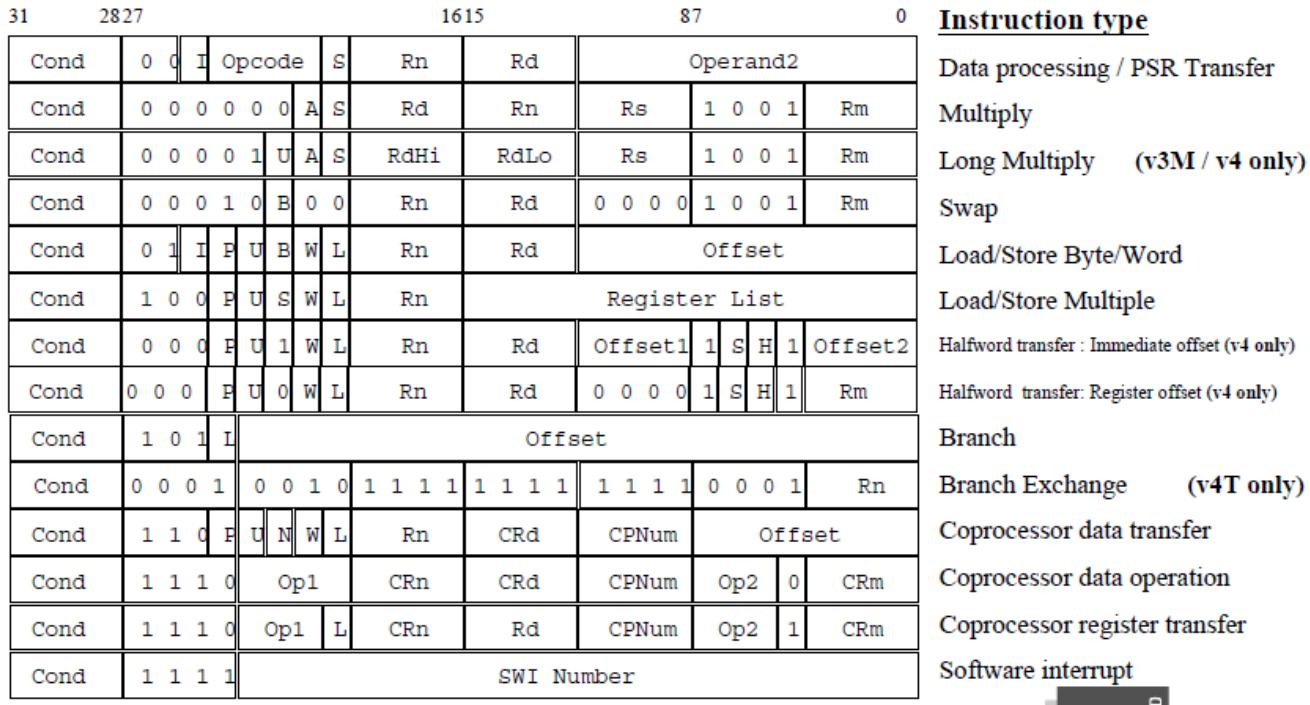
```
function perimetre (a,b : in integer) return integer is
begin
    perimetre:=(2*a)+(2*b)
end ;
```



Processeur à jeu d'instructions réduit RISC

- Langage machine : affectation, itération, appel de procédure et branchement (conditionnel ou pas)
  - 60% des instructions sont de l'affectation
  - 75% des références mémoires portent sur des données simples (constantes ou variables scalaires)
- Conséquences
  - Diminution de la partie contrôle. Séquenceur en logique câblée donc plus rapide.
  - Diminution de la surface du séquenceur → plus de place pour les caches et les registres → possibilité de séparer données et instructions sur le composant...
  - Modes d'adressages et instructions simplifiées → compilateur plus simple et instructions de longueur fixe → donc pipeline possible

Exemple : format du jeu d'instruction ARM



- ARM Advanced RISC Machine : iPhone, plupart des opérations : un cycle
- Les instructions ont une longueur constante et beaucoup d'adressage registre

Processeurs réels

- Une opération = N phases distinctes, N cycles d'horloge
- Processeur en attente de la mémoire externe (distante, lente, grande taille)
- N tâches par processeur, horloge rapide

- Un résultat par cycle d'horloge : chemin de données et **pipeline**
- Copier des parties utilisées de la mémoire externe vers **mémoire cache** (proche, rapide, petite taille)
- 1 tâche par processeur, N processeurs, horloge ralenti : **multi-coeur**

Pipeline

Sans Pipeline

Cycle	1	2	3	4	5	6	7	8	9	10	11	…
Fetch	C1				C2				C3			
Decode		C1				C2				C3		
Read/Write			C1				C2				C3	
Execute				C1				C2				C3

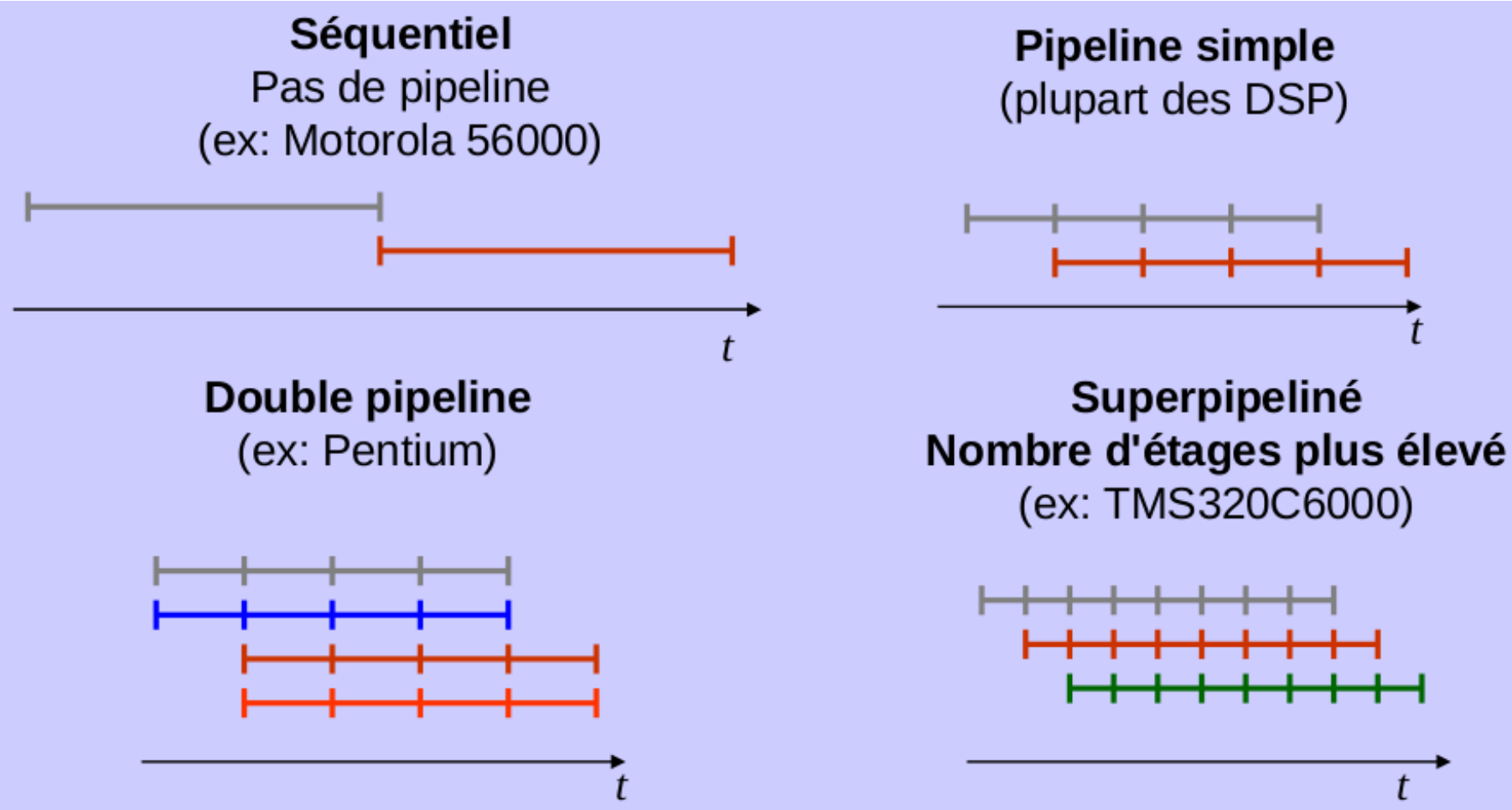
Fin C1 Fin C2 Fin C3

Avec pipeline

Cycle	1	2	3	4	5	6	7	8	9	10	11	…
Fetch	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	
Decode		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	…
Read/Write			C1	C2	C3	C4	C5	C6	C7	C8	C9	…
Execute				C1	C2	C3	C4	C5	C6	C7	C8	…

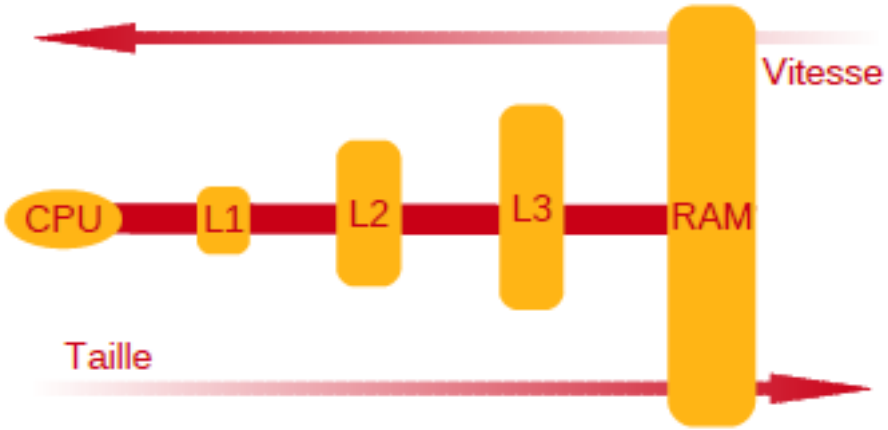
Fin C1 Fin C3

Exemples d'architectures



- 31 étages pour l'Intel Pentium 4 Prescott !

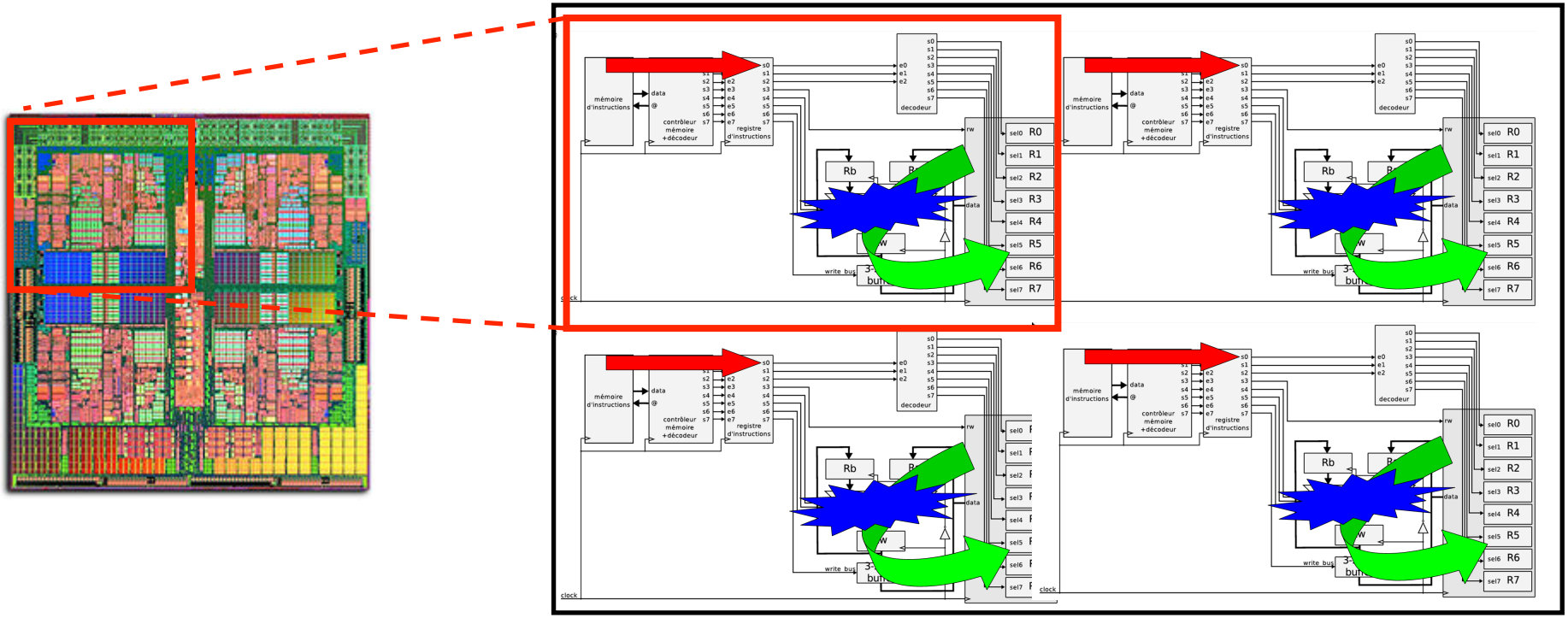
Hiérarchie mémoire et mémoire cache



Mémoire	Taille	Latence
L1	16kB	1 cycle
L2	256kB	20 cycles
L3	8MB	50 cycles
RAM	8GB	200 cycles

- Indépendant de l'architecture du processeur (gestion automatique)
- Transfert par blocs (e.g. 64B)
- Performances d'un algorithme : nombre de transferts de blocs
- Localité des accès mémoires
  - Localité spatiale : utiliser toutes les données d'un bloc
  - Localité temporelle : réutiliser le plus rapidement possible

Architectures Multi-coeur



AMD Magny-Cours

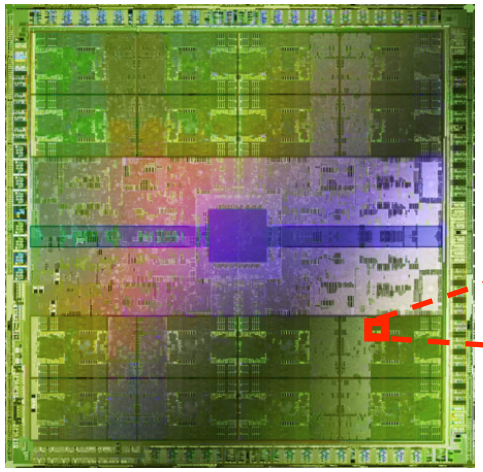
- 2x6 cœurs
- 1.7 - 2.3 Ghz
- 12x512KB L2 privés
- 2x6MB L3 partagé

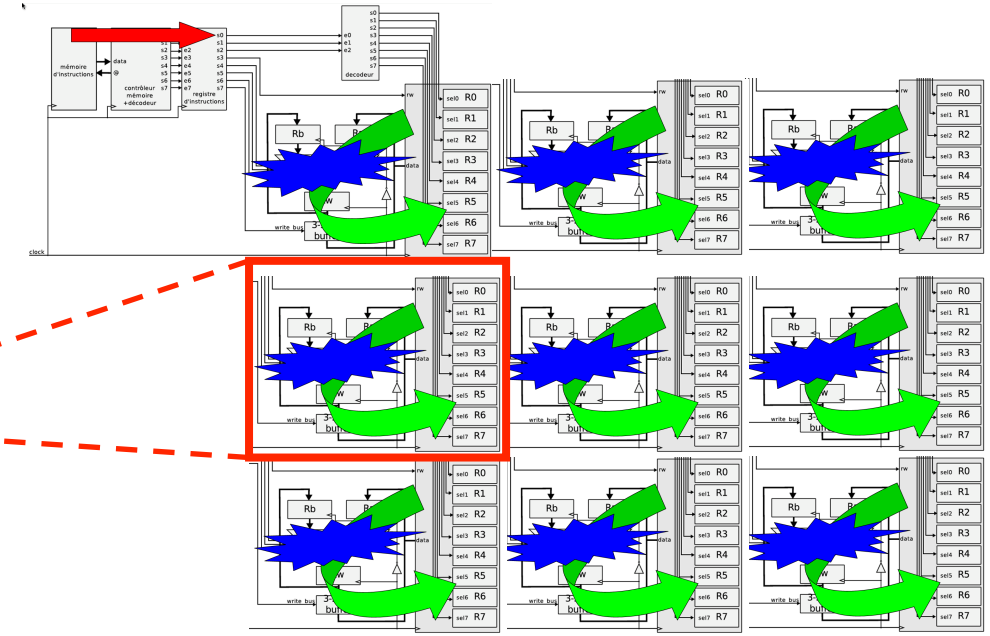
Intel Nehalem-EX

- 8 cœurs
- 1.6GHz-3.3GHz
- 8x256KB L2 privés
- 24MB L3 partagé

Processeur graphique

- SIMD (Single instruction, multiple data) : un flux d'instructions et N flux de données
  - Traitement vidéo
  - Calcul scientifique

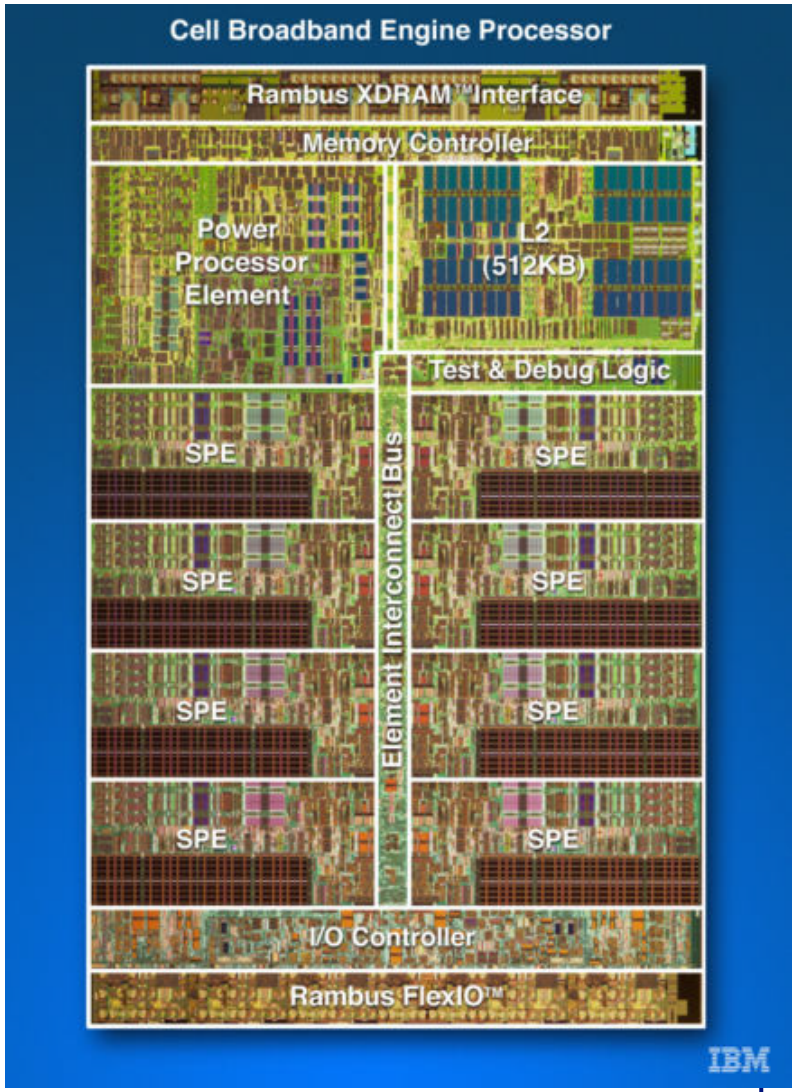




- NVIDIA Fermi
  - 430mm²
  - 3x10⁹ transistors (40 nm)
  - 1,5 GHz
  - 512 cœurs

Processeur Cell (Sony, Toshiba, IBM)

- 9-core processor
  - Power Processing Element (PPE)
    - conventional Power processor
    - not supposed to perform all operations itself, acting like a controller
    - running conventional operating systems
    - 16 KB instruction/data level 1 cache
    - 512 KB level 2 cache
  - Synergistic Processing Elements (SPE)
    - specialized co-processors for specific types of code, i.e., very high performance vector processors
    - local stores
    - can do general purpose operations
    - the PPE can start, stop, interrupt and schedule processes running on an SPE



Processeur Cell (Sony, Toshiba, IBM)

- Cell has in essence traded running everything at moderate speed for the ability to run certain types of code at high speed
- used for example in
  - Sony PlayStation 3:
    - 3.2 GHz clock
    - 7 SPEs for general operations
    - 1 SPE for security for the OS
  - Toshiba home cinema:
    - decoding of 48 HDTV MPEG streams
      - dozens of thumbnail videos simultaneously on screen
  - IBM blade centers:
    - 3.2 GHz clock
    - Linux ≥ 2.6.11

Aide-mémoire

Processeur = assemblage de composants « simples »

- Instruction = configuration de la partie calcul
  - Identification des registres, sens de communication
  - Identification de l'opération
  - Stockage du résultat (interne / externe)
- Mémoire contient les instructions et les données (zones distinctes)
- Compilation = interprétation de programmes haut-niveau en instructions machine

Enseignements S7 (approfondissements)

- STI a 1-FH – Architectures embarquées et informatique industrielle
  - Le processeur dans son environnement, programmation contrainte
- STI a 3-EG – Architectures numériques de traitement de l'information
  - Fonctionnement des processeurs performants, informatique matérielle
- STI a 4-EG – Capteurs Intelligents Communicants : Systèmes d'Interface
  - Instrumentation (capteurs), microcontrôleurs et transmission de l'information

Enseignements S8 (électifs)

- ELC B-2 – Prototypage de systèmes embarqués : circuits programmables
  - FPGA, circuits reconfigurables
- ELC C-3 - Systèmes embarqués collaboratifs
  - Informatique collaborative
- ELC C-4 – Capteurs et traitement d'images
- ELC D-2 – Filtrage adaptatif : application au contrôle actif de bruit
  - Architectures dédiées au traitement du signal
- ELC F-2 – Systèmes mécatroniques intelligents
  - Robotique, instrumentation d'objets mécaniques

Enseignements S9 / M2R

- MOD 3.3 – Systèmes embarqués en environnement hostile
- MOD 3.4 – Microsystèmes autonomes
  - Chaînes d'acquisition de données naturelles – fabrication de capteurs, énergie embarquée
- MOD 7.5 – Green computing
  - Architectures parallèles – réduction de la consommation
- Master EEAP (Electronique Electrotechnique Automatique Procédés) – parcours Recherche ESE (Electronique et Systèmes Embarqués)