Performance de processeurs

B. Miramond – Polytech Nice

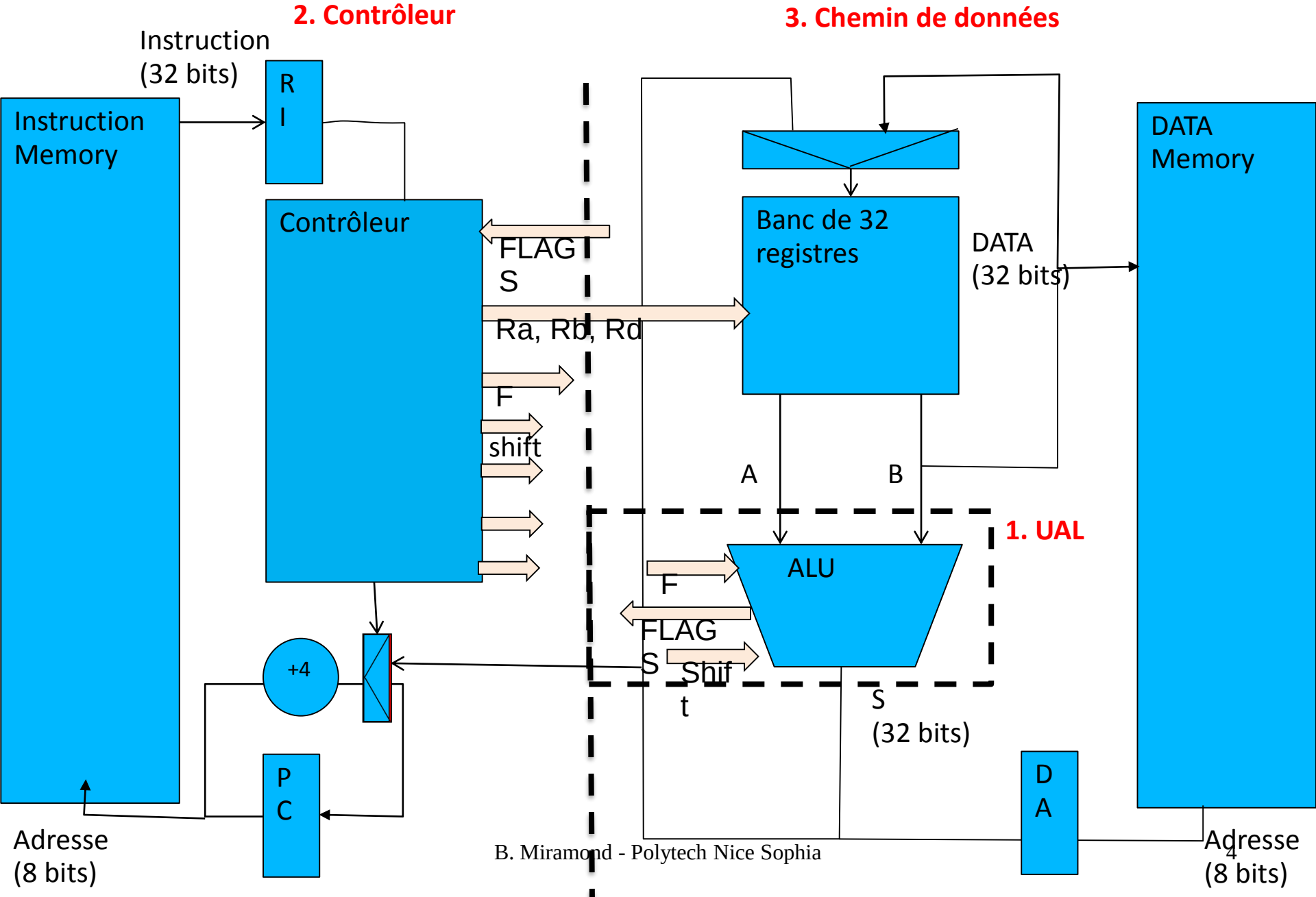
Synthèse du projet

Répartition des rôles

Chaque groupe dispose de 4 séances :

- G1 – Réalisation de l'UAL
- G2 - Contrôleur-MémoireI
- G2 – Registres-MémoireD
- G3 - Assembleur
  - Remplissage mémoire
- G4 - FPGA
  - Déploiement sur FPGA
  - Environnement de test
- Ecriture de jeux de test

Architecture générale



4 Types d'instructions

a) **Shift, add, sub, mov,**
   - 8 instructions

b) **Data Processing,**
   - 16 instructions

c) **Load/Store,**
   - 2 instructions

d) **Stack pointer,**
   - 2 instructions

e) **Branch**
   - 1 instruction

Table A5-1 16-bit Thumb instruction encoding

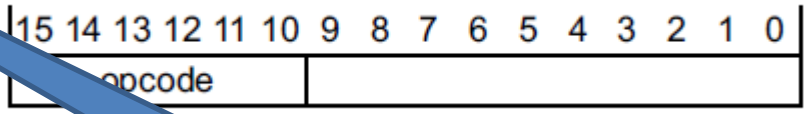
opcode	Instruction or instruction class
00xxxx	*Shift (immediate), add, subtract, move, and compare* on page A5-128
010000	*Data processing* on page A5-129
010001	*Special data instructions and branch and exchange* on page A5-130
01001x	Load from Literal Pool, see *LDR (literal)* on page A7-254
0101xx 011xxx 100xxx	*Load/store single data item* on page A5-131
10100x	Generate PC-relative address, see *ADR* on page A7-197
10101x	Generate SP-relative address, see *ADD (SP plus immediate)* on page A7-193
1011xx	*Miscellaneous 16-bit instructions* on page A5-132
11000x	Store multiple registers, see *STM, STMIA, STMEA* on page A7-422
11001x	Load multiple registers, see *LDM, LDMIA, LDMFD* on page A7-248
1101xx	*Conditional branch, and supervisor call* on page A5-134
11100x	Unconditional Branch, see *B* on page A7-207

Code d'instruction		Catégorie A	Catégorie B	Catégorie C	Catégorie D	Catégorie E
00 XX XX	Shift, add, sub…	1				
01 00 00	Data processing		1			
01 10 XX	Load/Store			1		
10 11 XX	Stack pointer				1	
11 01 XX	Branch					1

Codage Thumb2 16 bits

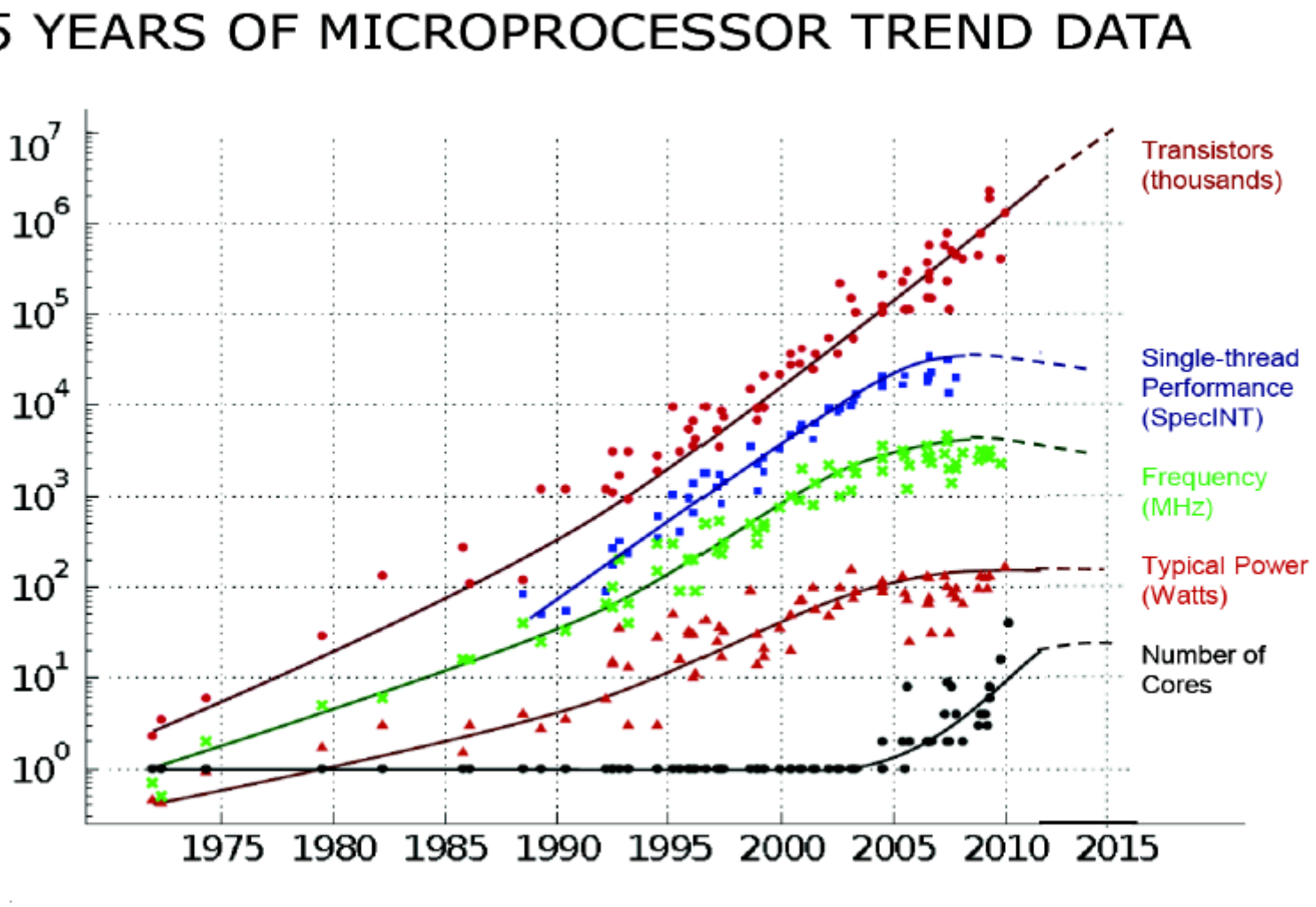
Instructions pour le projet

1. **00xxxx** **Shift (immediate), add, subtract, move, and compare**
2. **010000** ***Data processing***
- 010001 *Special data instructions and branch and exchange*
- 01001x Load from Literal Pool
3. **0101xx**
   011xxx ***Load/store single data***
   100xxx
- 10100x Generate PC-relative address
- 10101x Generate SP-relative address
- 1011xx *Miscellaneous 16-bit instructions*
- 11000x Store multiple registers
- 11001x Load multiple registers
4. **1101xx** ***Conditional branch, and supervisor call***
- 11100x Unconditional Branch



Commençons par le type le plus simple

Evolution de la performance des CPU



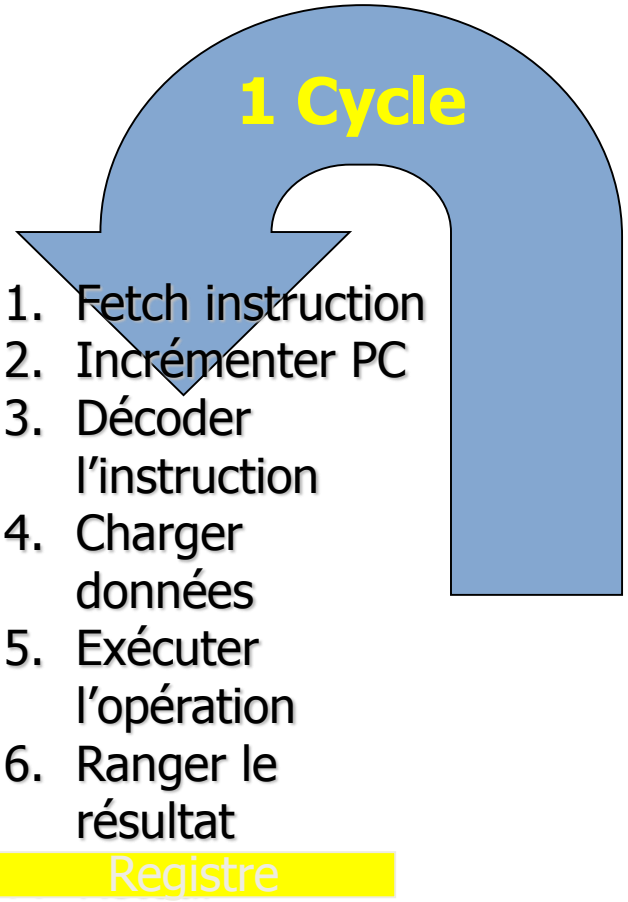
Mécanismes d'amélioration de performances

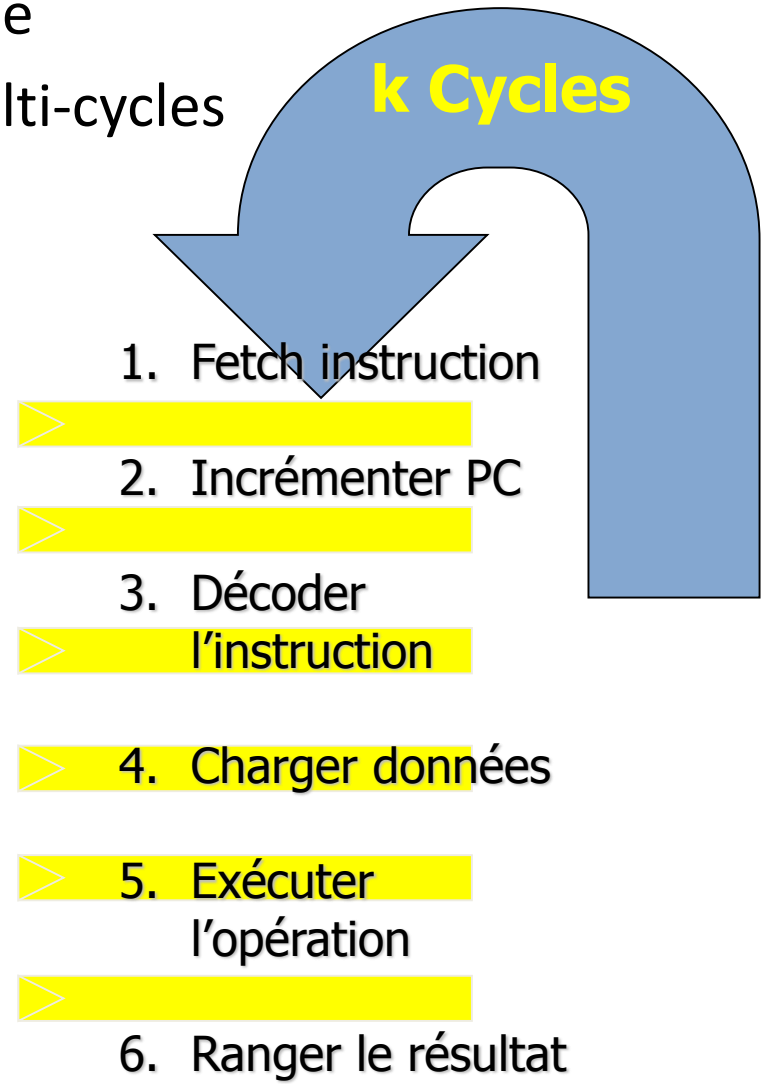
- Implémentation du contrôleur : Pipeline
- Accès mémoire : le cache
- Exécution des opérations : parallélisme

1. Choix d'implémentation du contrôleur

L'exécution du cycle machine peut prendre

- Un cycle – implémentation mono-cycle
- Plusieurs cycles – implémentation multi-cycles



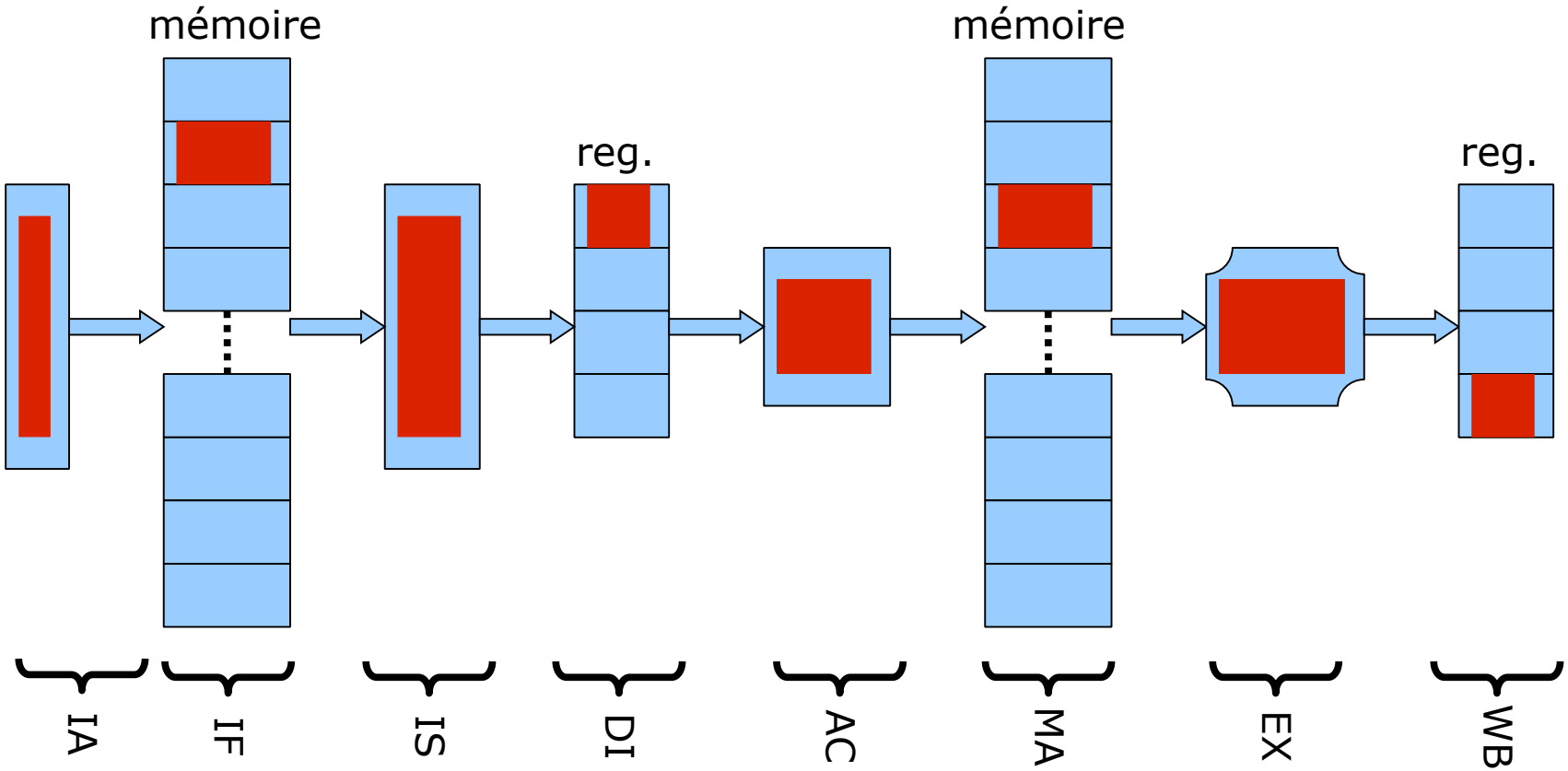


1.1 La technique du pipeline

L'exécution complète d'une instruction dans un processeur met en jeu plusieurs unités du processeur (unité d'adressage, unité de décodage, unité d'exécution, etc...) qui ne sont pas **toutes** utilisées simultanément.

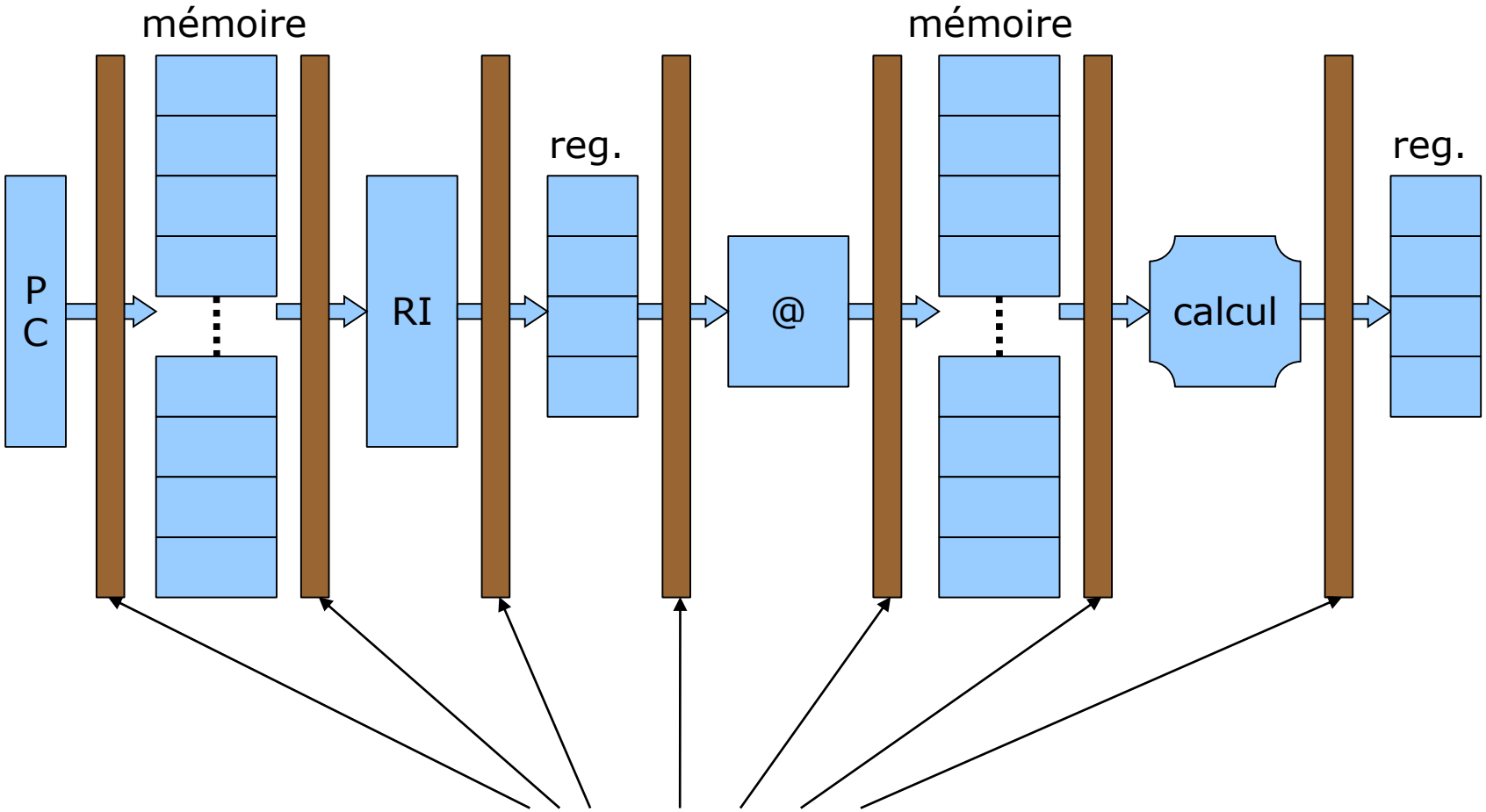
La technique du **pipeline** consiste à améliorer les performances du processeur en même temps que son **rendement**.

Le **pipeline** permet de superposer les sous-cycles d'exécution des instructions.



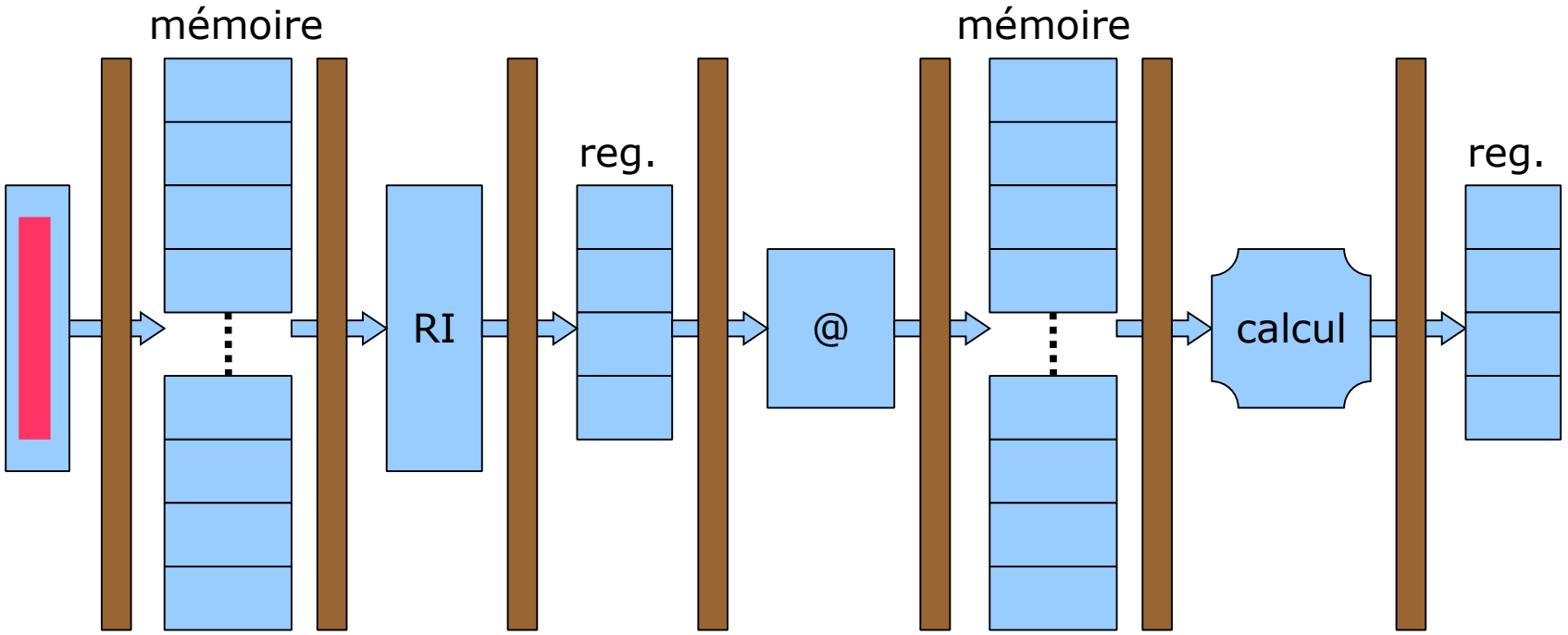
Cycles du processeur

1. Accès Instruction (IA)
2. Recherche Instruction (IF)
3. Stockage Instruction (IS)
4. Décodage opérandes (DI)
5. Calcul d'adresse (AC)
6. Accès mémoire (MA)
7. Exécution (EX)
8. Ecriture résultat (WB)

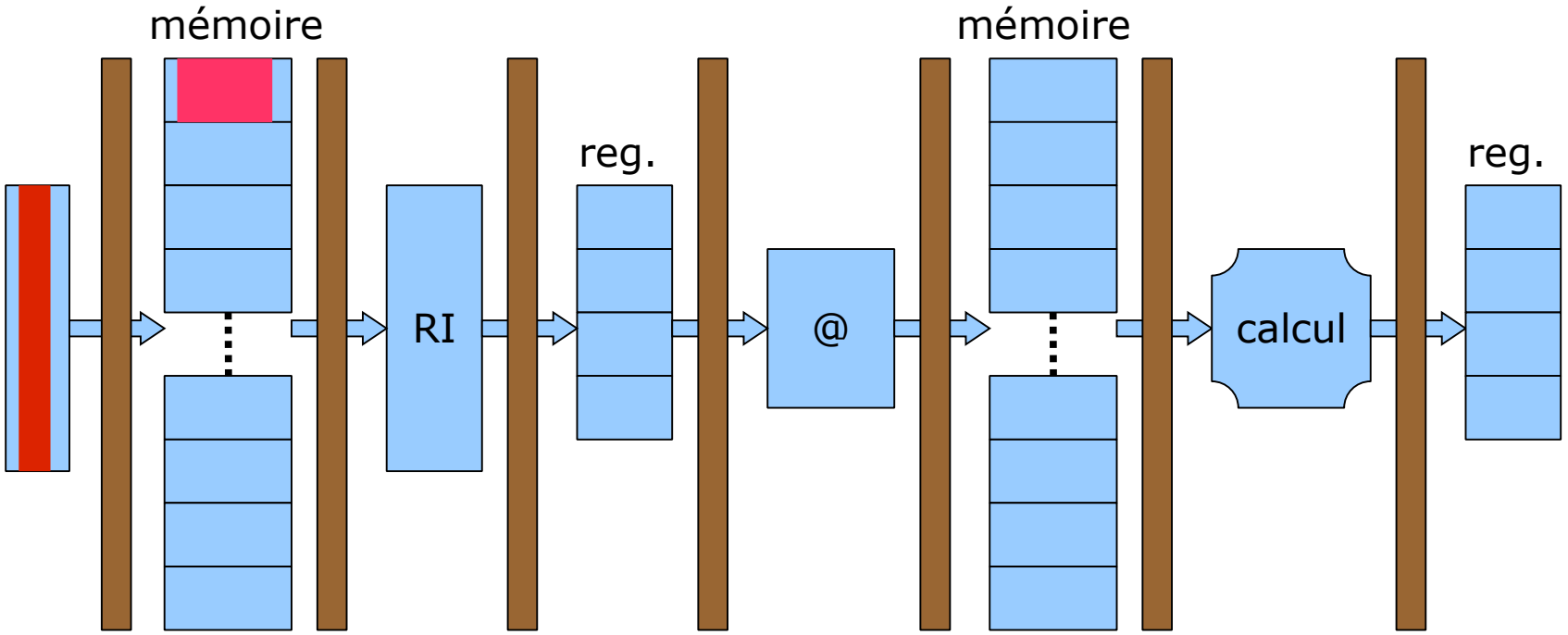


Insertion de registres d'étages
(les niveaux du pipeline)

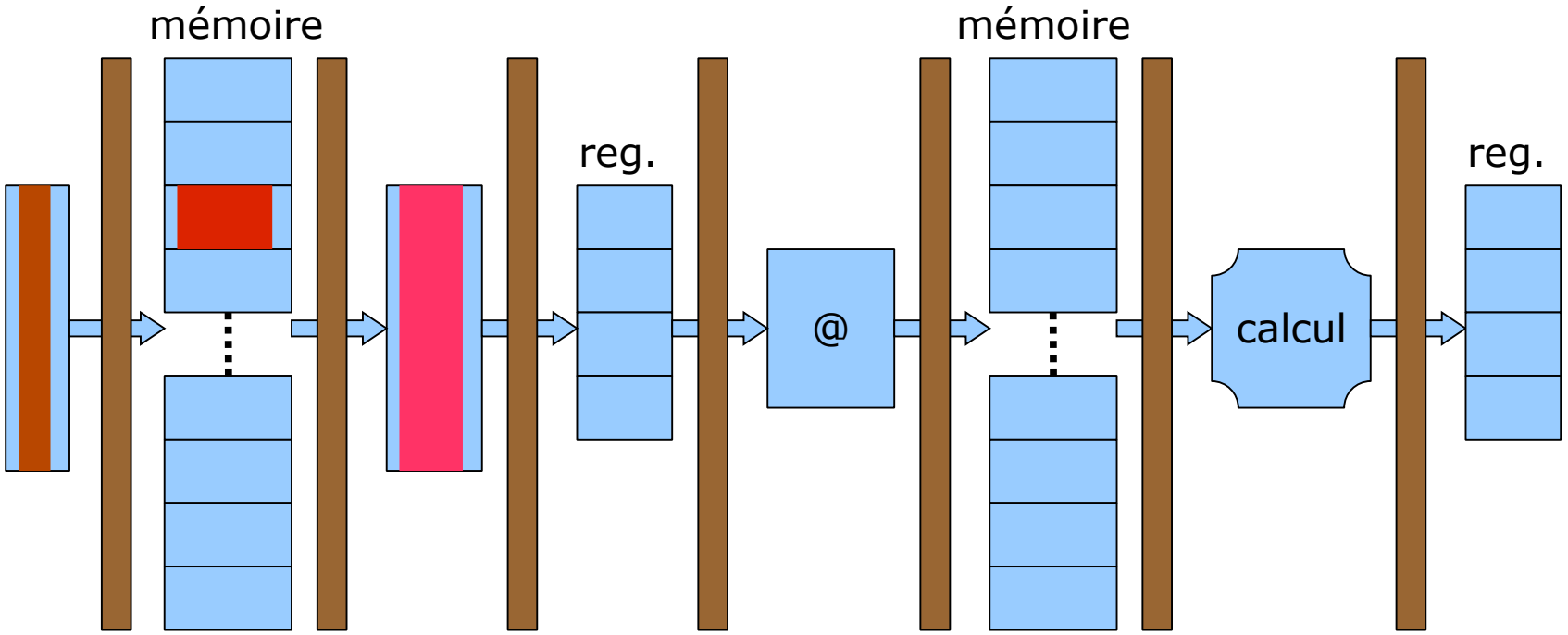
Les données et tous les signaux de contrôle sont mémorisés et propagés à chaque niveau



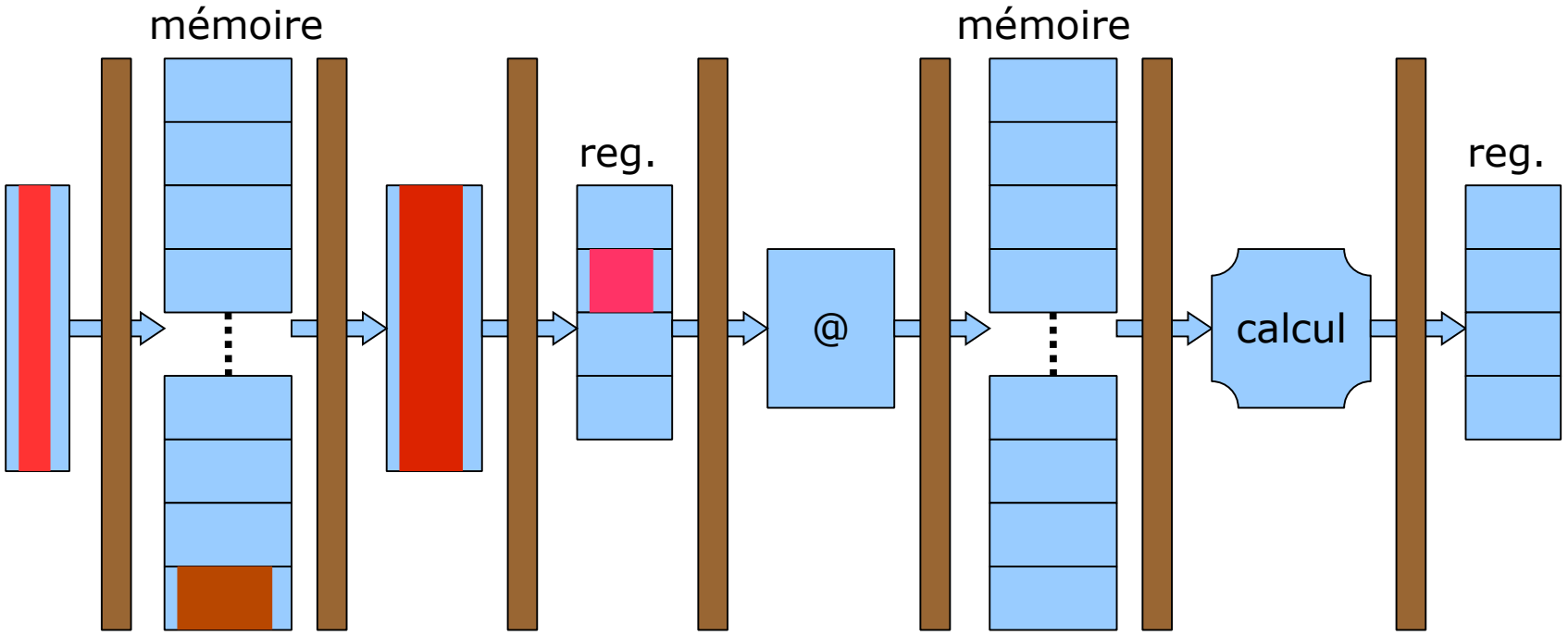
Instruction i : IA



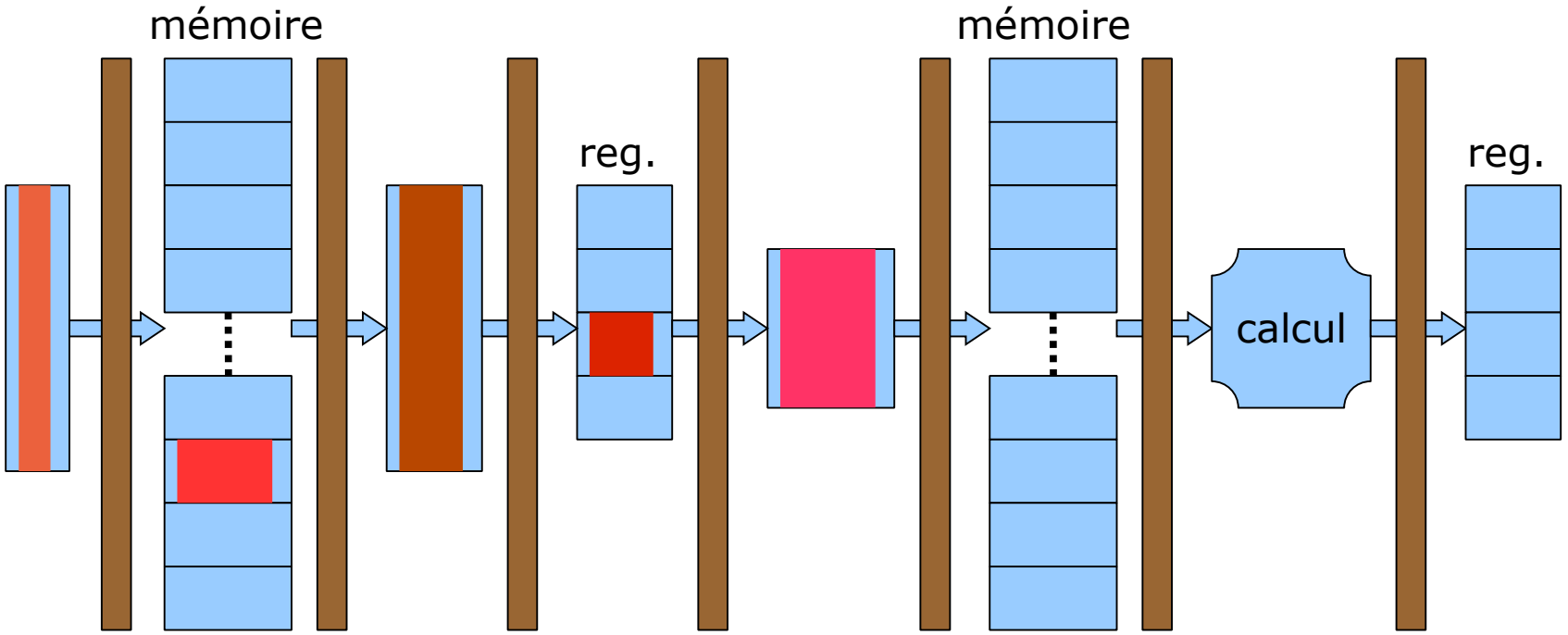
Instruction i : IA IF
Instruction i+1 : IA



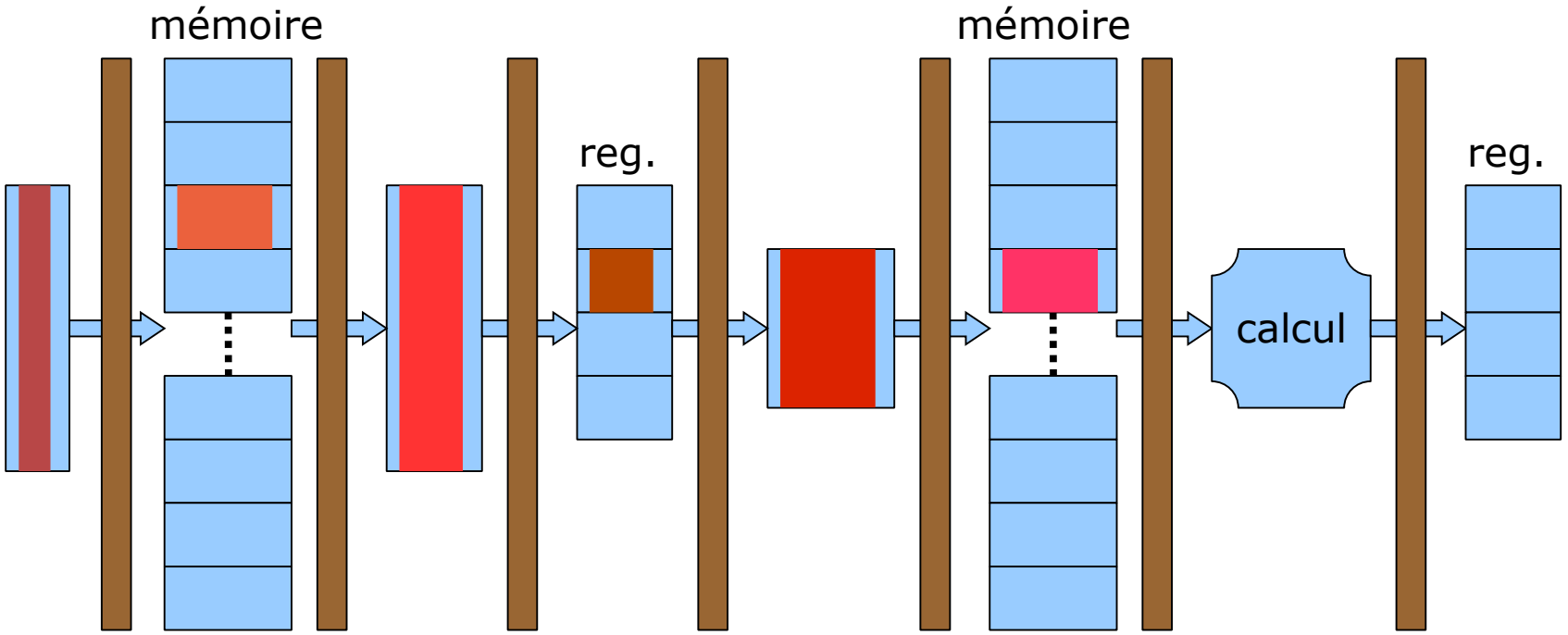
Instruction i :	IA	IF	SI
Instruction i+1	:	IA	IF
Instruction i+2	:	IA	



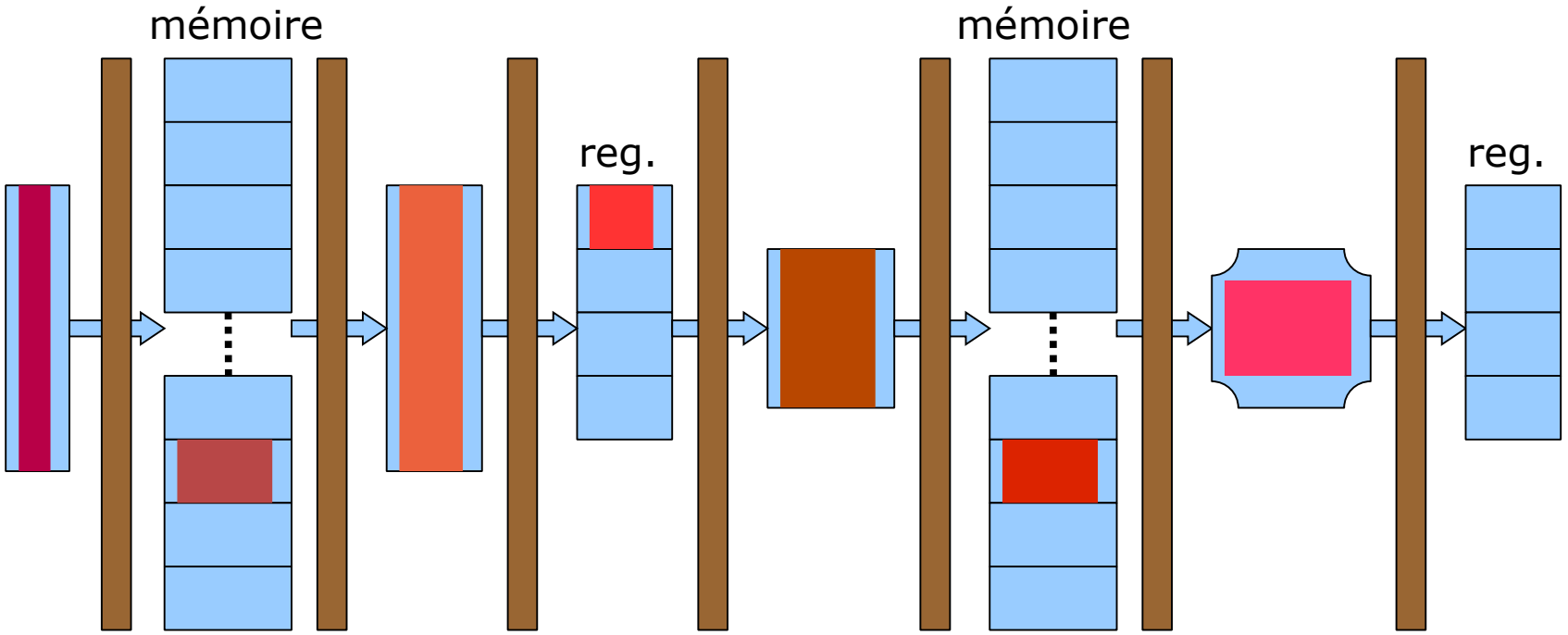
Instruction i :	IA	IF	SI	DI
Instruction i+1	:	IA	IF	SI
Instruction i+2	:	IA	IF	
Instruction i+3	:	IA		



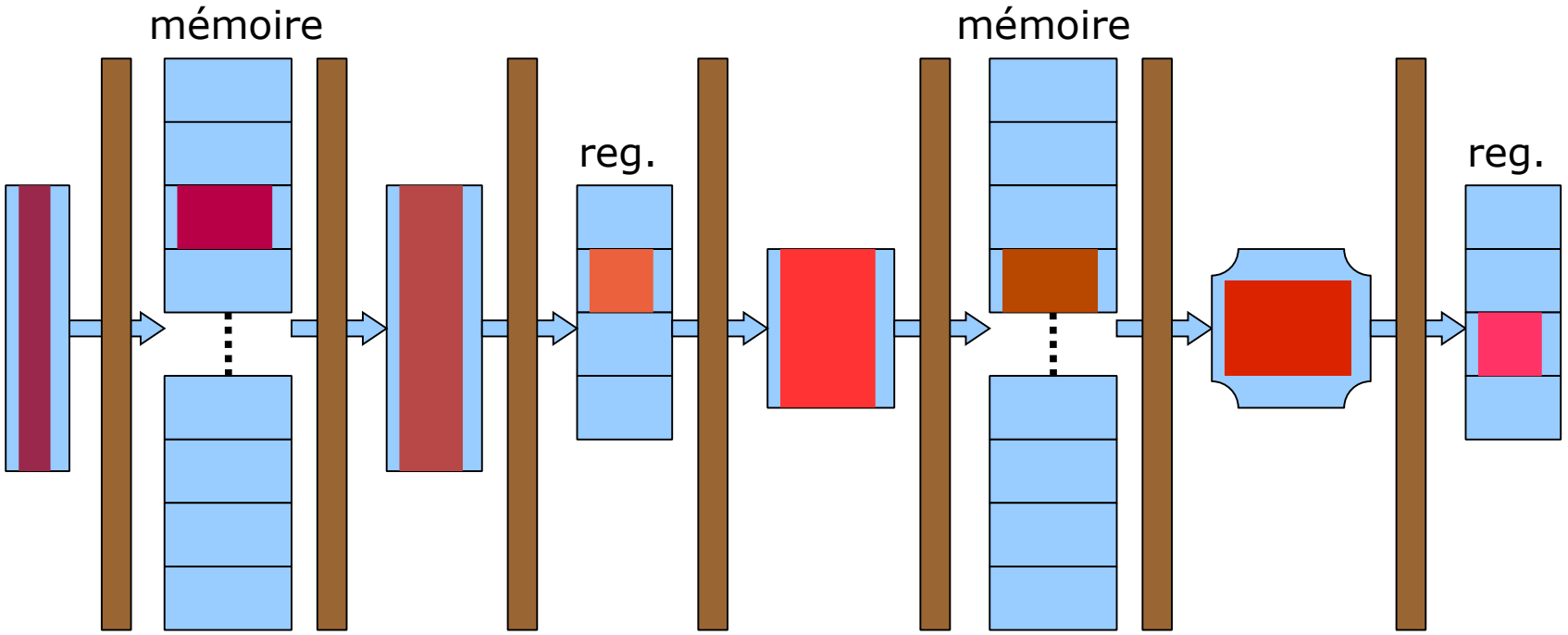
Instruction i :	IA	IF	SI	DI	AC
Instruction i+1	:	IA	IF	SI	DI
Instruction i+2	:	IA	IF	SI	
Instruction i+3	:	IA	IF		
Instruction i+4	:	IA			



Instruction i :	IA	IF	SI	DI	AC	MA
Instruction i+1	:	IA	IF	SI	DI	AC
Instruction i+2	:	IA	IF	SI	DI	
Instruction i+3	:	IA	IF	SI		
Instruction i+4	:	IA	IF			
Instruction i+5	:	IA				



Instruction i :	IA	IF	SI	DI	AC	MA	EX
Instruction i+1	:	IA	IF	SI	DI	AC	MA
Instruction i+2	:	IA	IF	SI	DI	AC	
Instruction i+3	:	IA	IF	SI	DI		
Instruction i+4	:	IA	IF	SI			
Instruction i+5	:	IA	IF				
Instruction i+6	:	IA					



Toutes les unités du processeur sont utilisées à 100 %

Instruction i :	IA	IF	SI	DI	AC	MA	EX	**WB**
Instruction i+1	:	IA	IF	SI	DI	AC	MA	**EX**
Instruction i+2	:	IA	IF	SI	DI	AC	**MA**	
Instruction i+3	:	IA	IF	SI	DI	**AC**		
Instruction i+4	:	IA	IF	SI	**DI**			
Instruction i+5	:	IA	IF	**SI**				
Instruction i+6	:	IA	**IF**					
Instruction i+7	:	**IA**						

Effet du pipeline

L'effet du pipeline rentabilise l'utilisation du silicium

=> Augmentation du rendement silicium (proche de 100%)

Le pipeline augmente le **débit** des instructions sans modifier le **temps d'exécution** des instructions

=> augmentation de la fréquence d'exécution des instructions

=> augmentation de la performance de la machine

La présence du pipeline permet d'augmenter la fréquence **apparente** d'exécution des instructions

=> plus de niveaux, niveaux plus «courts», donc plus rapides

Exemple :

Pentium IV = 20 niveaux de pipeline !!

1.2 Inconvénients du pipeline

1. Il est impossible d'assurer **en permanence** l'exécution d'une instruction par cycle d'horloge

Le débit **effectif** est inférieur à 1 inst/s (IPC < 1)

2. Un pipeline doit être **rempli** et **vidé** en cas de rupture de séquence

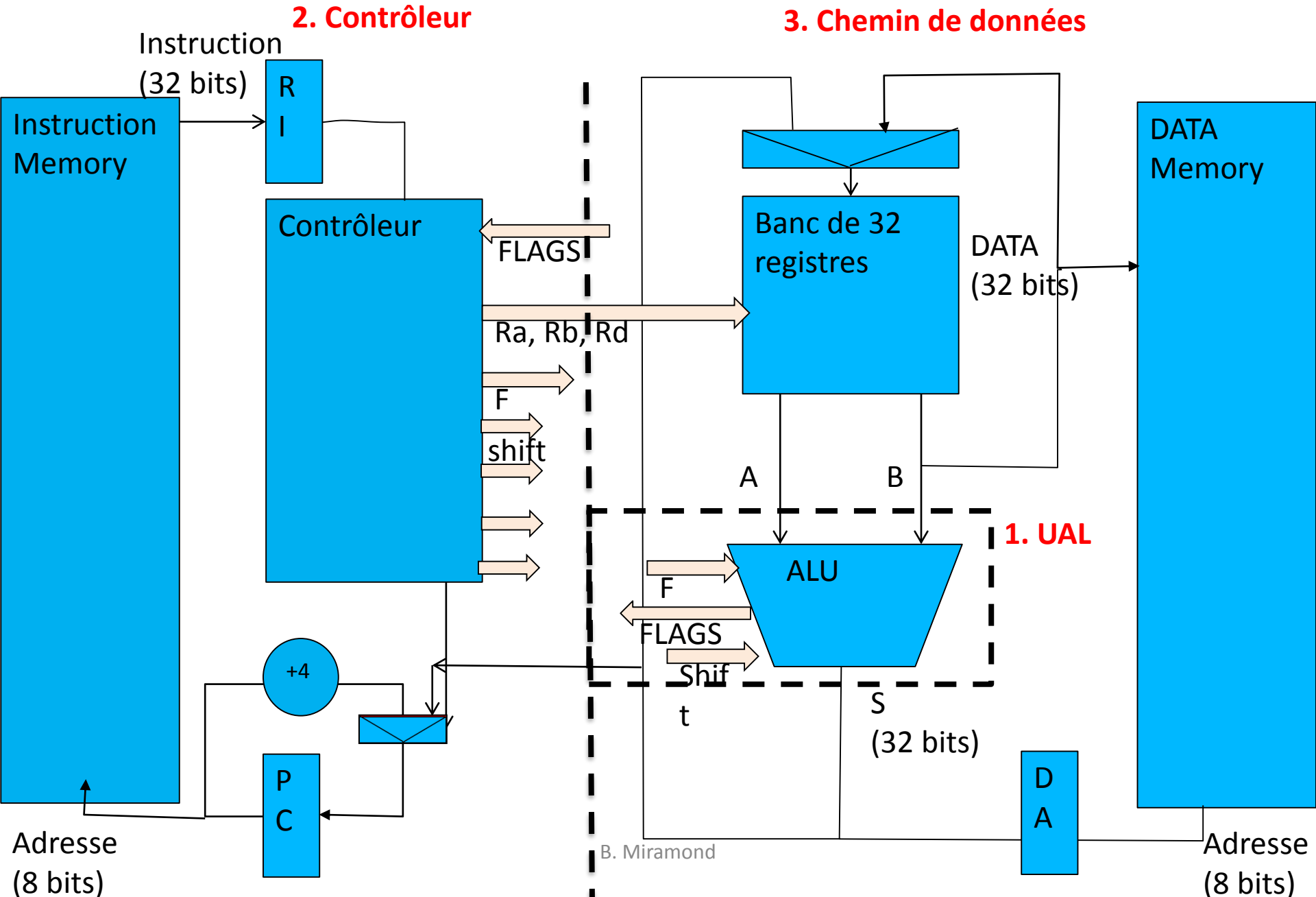
3. Il peut apparaître des dépendances entre les instructions dans le pipeline (les **aléas de pipeline**) :

- Conflit d'accès à des ressources : **aléas structurels**
- Dépendances de données entre instructions : **aléas de données**
- Rupture de séquence : **aléas de contrôle**

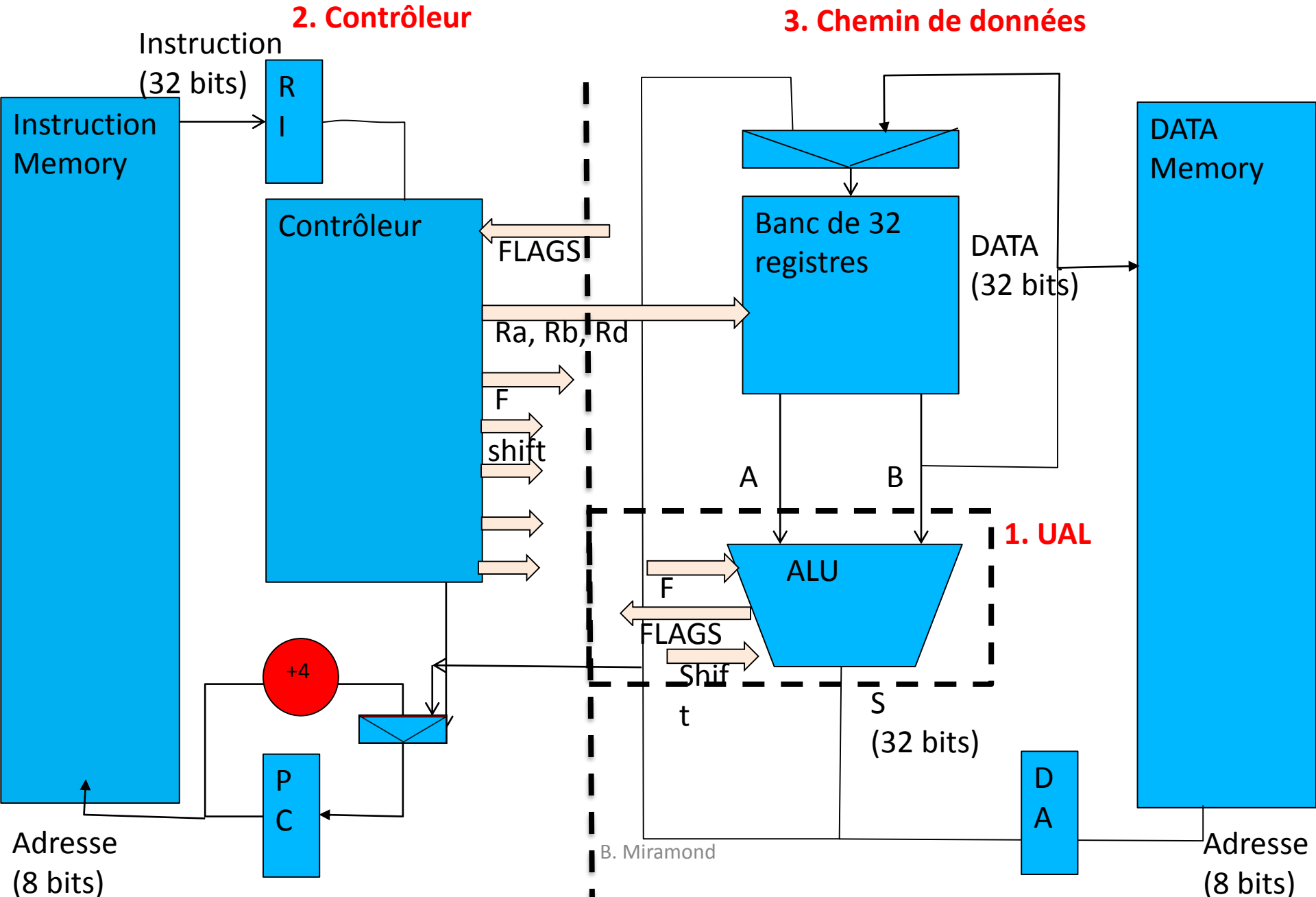
1.3 Synchronisation dans le projet

- Le projet fait apparaître un contrôleur/décodeur qui interprète les instructions
- Il existe un décalage entre le moment du décodage et celui de l'exécution
- Pendant ce temps, une nouvelle instruction a été lue puis décodée en mémoire d'instruction
- Sans autre modification, il apparaît un problème de synchronisation : le branchement est retardé d'un cycle !

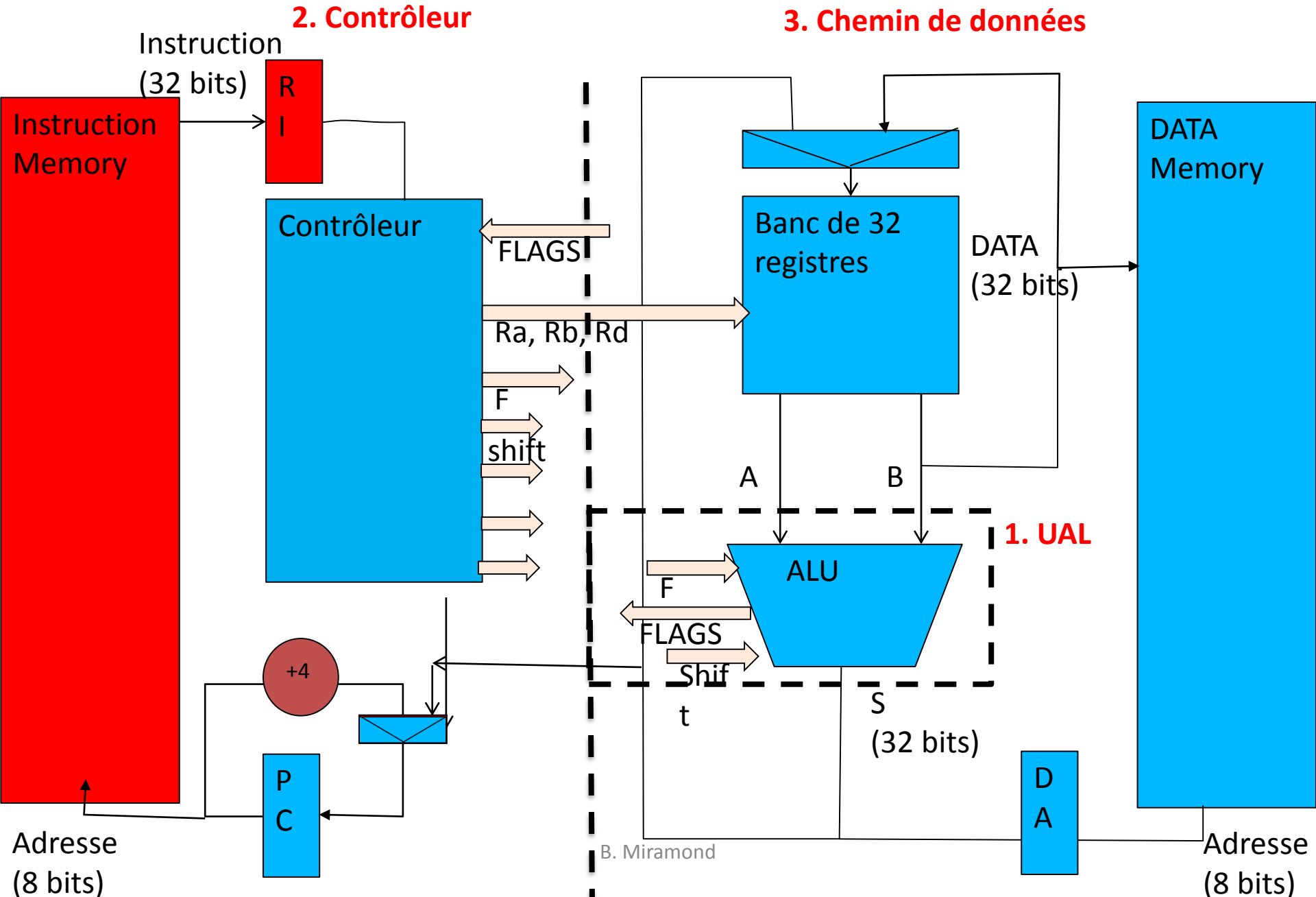
Architecture générale



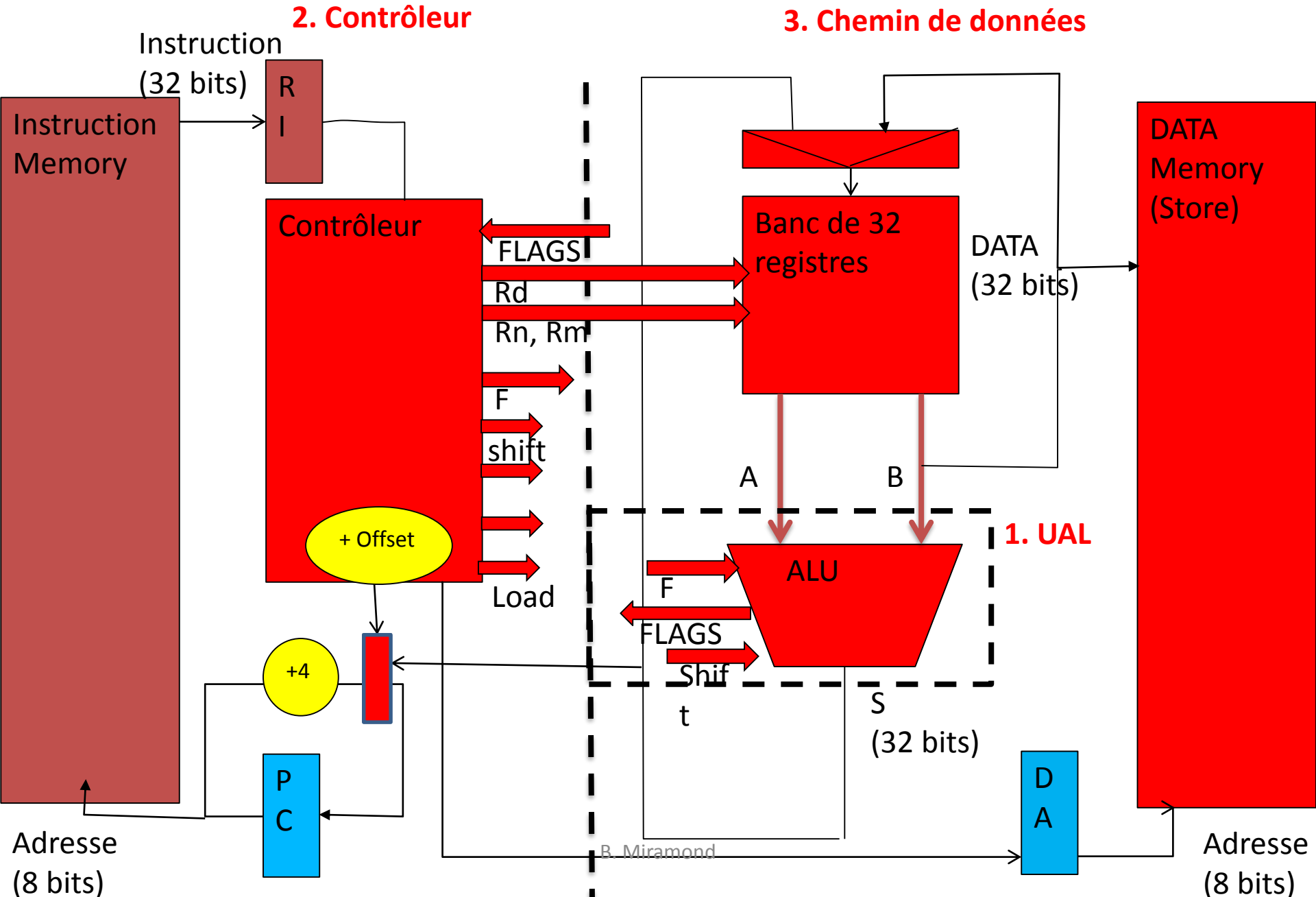
Architecture générale



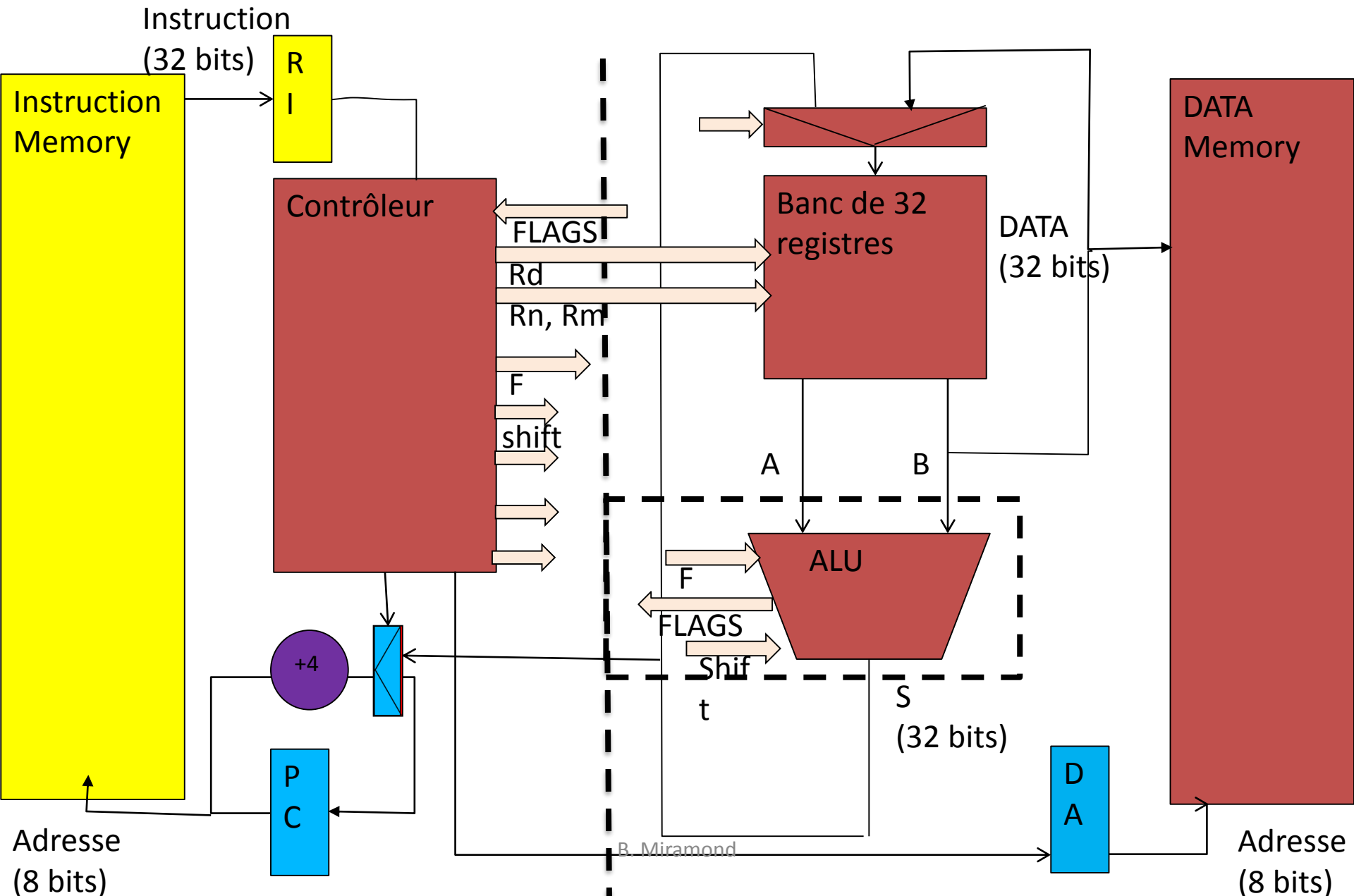
Architecture générale



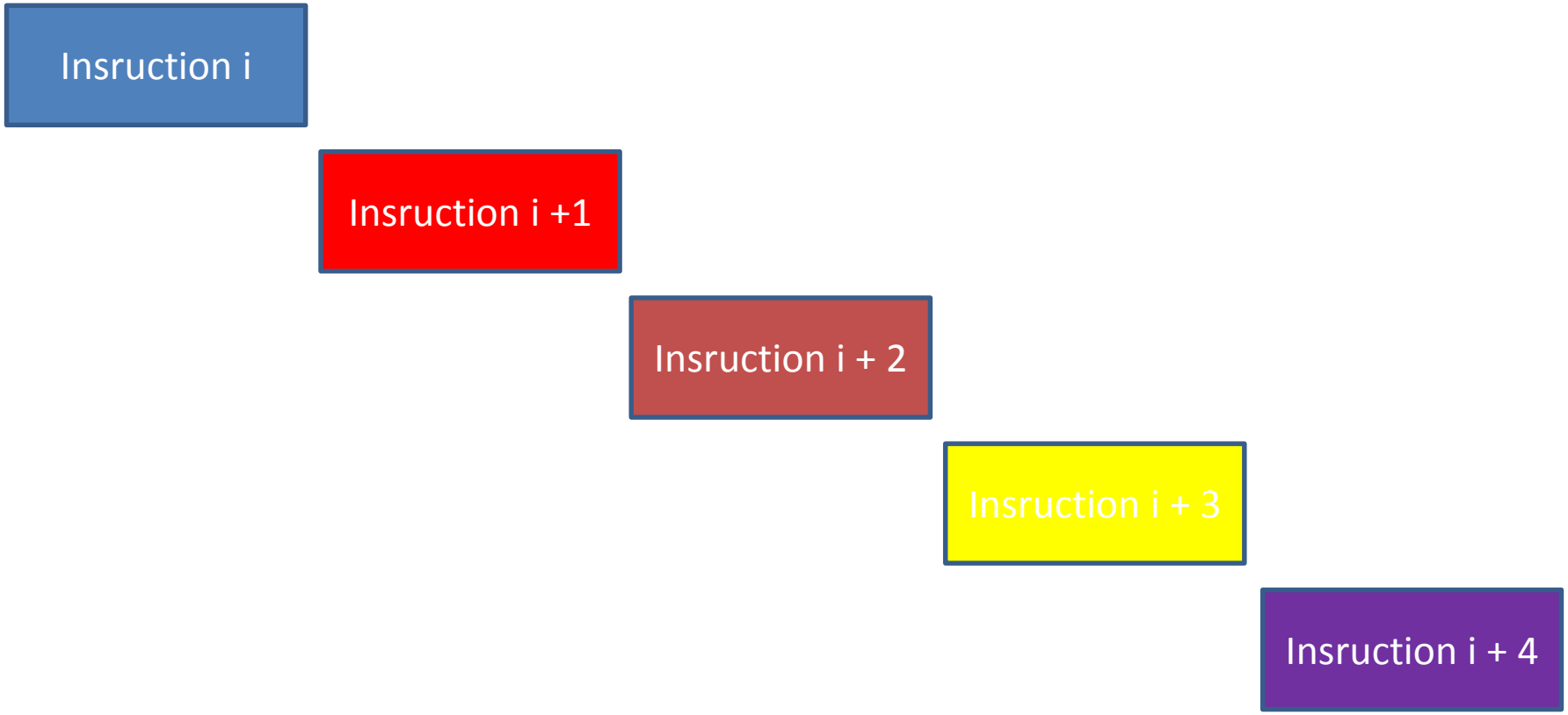
Architecture générale



1 instruction est déjà entrée dans le pipeline

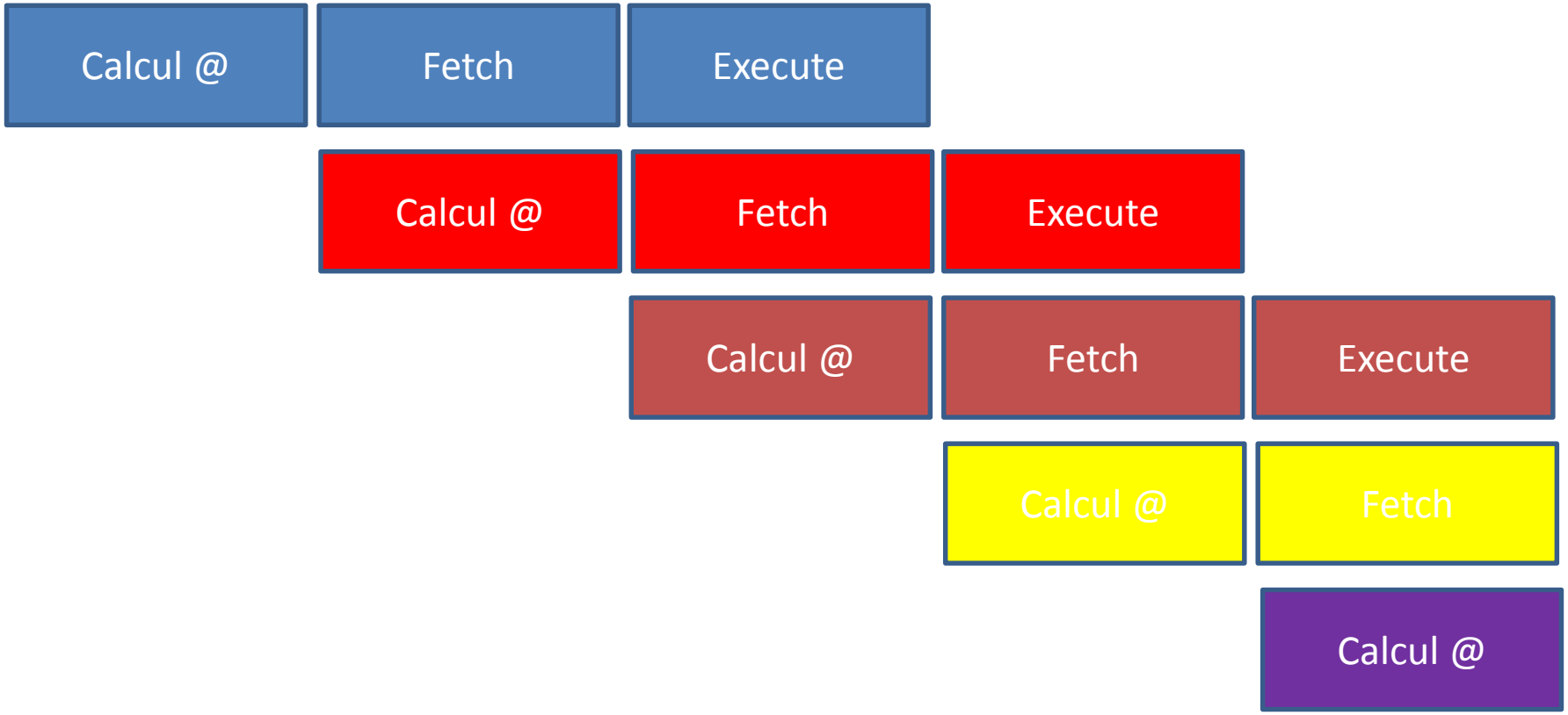


Exécution des instructions
sans pipeline



Le rendement du processeur est donc augmenté, sans modification de la latence d'exécution
On augmente le Débit
On augmente pas la Latence !

Exécution des instructions
avec pipeline



Le rendement du processeur est donc augmenté, sans modification de la latence d'exécution
On augmente le Débit
On augmente pas la Latence !

2 Les mémoires cache

Le principe consiste à interposer entre le processeur et la mémoire centrale une mémoire de taille plus petite mais beaucoup plus rapide (SRAM)

- les temps de cycle du processeur sont toujours plus courts que ceux de la mémoire principale (SDRAM) : +60%/an contre +9%/an.

Le processeur teste la présence de ses données dans le cache :

- Donnée présente = succès = *cache hit*
- Donnée non présente = défaut = *cache miss*

L'amélioration des performances dépend du taux de succès dans le cache (de l'algorithme de mise à jour du cache) et du temps d'accès au cache

Une mémoire cache exploite les propriétés de localité temporelle et spatiale des **données** :

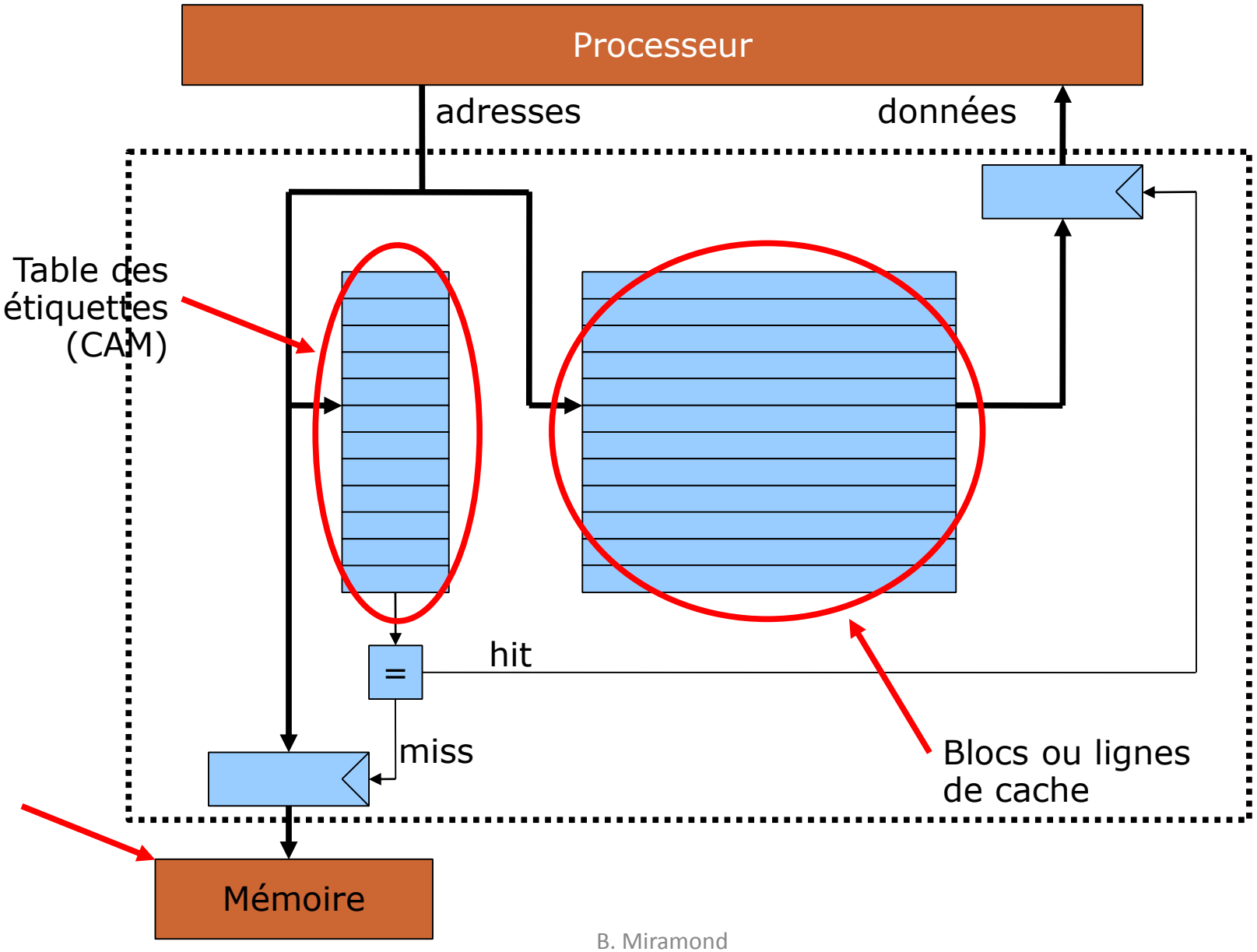
Localité temporelle

Une donnée adressée à l'instant *t* possède une forte probabilité d'être adressée à nouveau dans un avenir proche

Localité spatiale

Si une donnée a été adressée à l'instant *t*, alors une donnée voisine possède une forte probabilité d'être adressée dans un avenir proche

=> Les **instructions** d'un programme possèdent de fortes caractéristiques de localité temporelle et spatiale (boucles, branchements courts)


Processeur
adresses
données
Table des
étiquettes
(CAM)
=
hit
miss
Blocs ou lignes
de cache
Mémoire

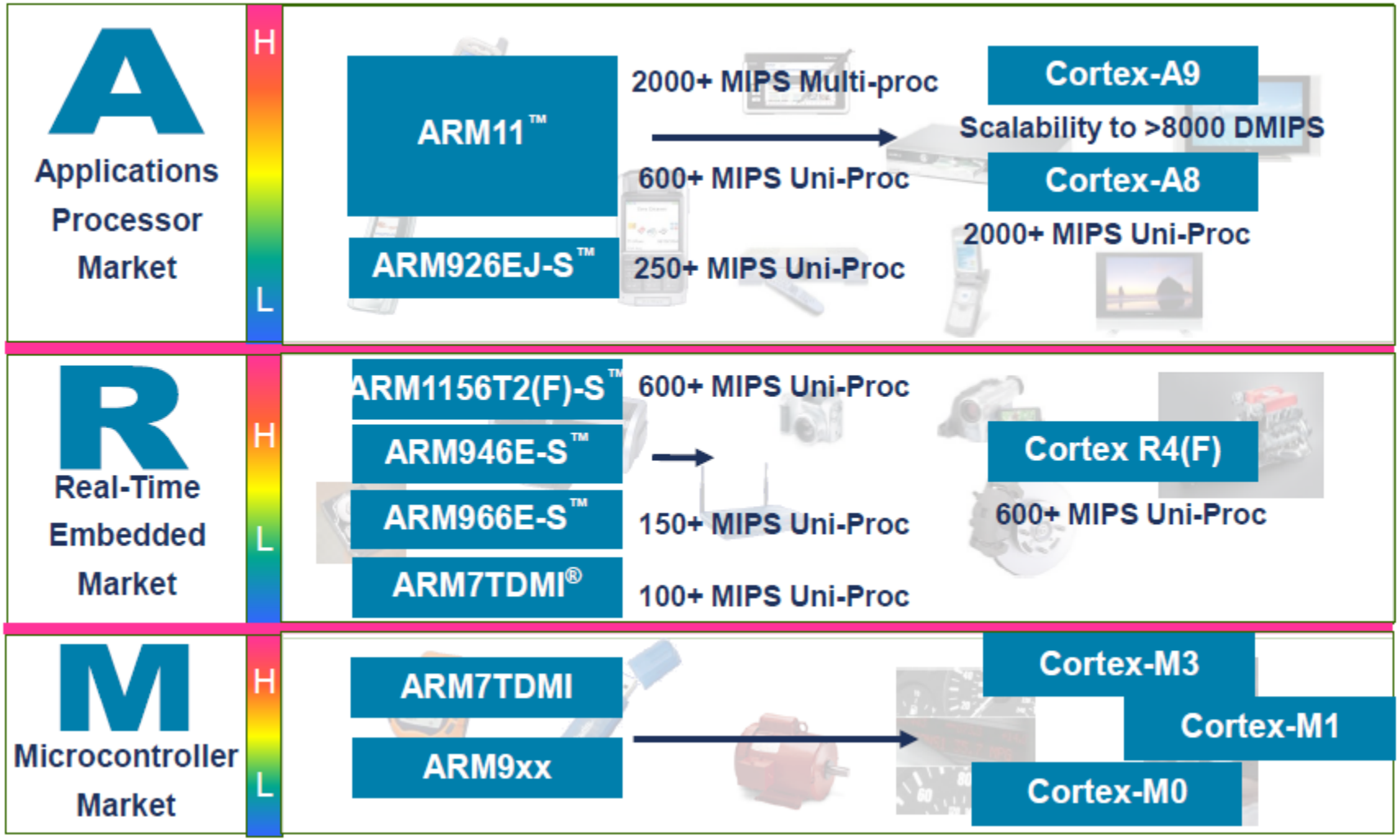
Utilisation des mémoires cache :

1) Avec l'augmentation du nombre de niveaux de pipeline et de la fréquence de fonctionnement des processeurs, les cycles d'accès aux mémoires sont de plus en plus courts. Les mémoires caches permettent d'**accélérer les accès** à la mémoire.
2) Pipeline = 1 instruction par cycle = aléas de ressources (mémoire). Une mémoire cache permet de résoudre une partie des **aléas structurels** dans le pipeline (cas des DSP avec des caches «simples».

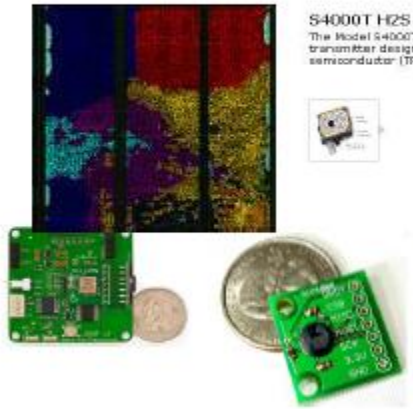
Inconvénients :

- Les temps de calculs sont difficiles à anticiper : dépendance aux compilateurs
- Gestion difficile de la cohérence des données (caches multiples)
- Ressources supplémentaires pour la mise à jour des caches

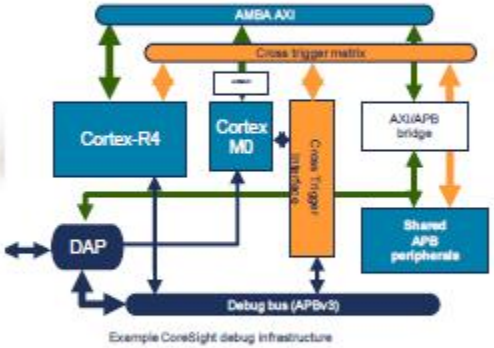
Comparaison dans la famille ARM



Cortex-M Family – Application Breadth



Storage



Example CoreSight debug infrastructure

Mixed Signal

System on Chip

Smart Sensors

Automotive

Ultra LowPower Connectivity

Human Interface

- Touch screen controller
- Accelerometer
- GPS / Bluetooth
- Battery monitor
- Joystick/touchpad controller

faultRobust

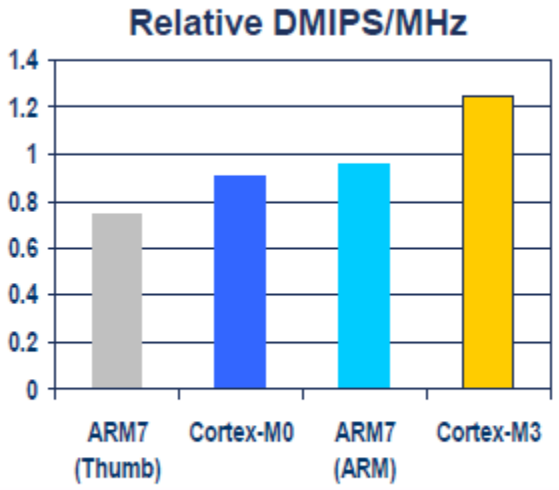
Bluetooth®

ZigBee™

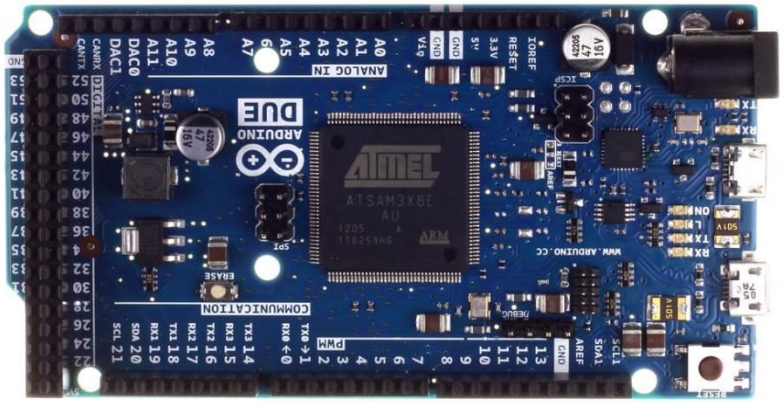
Cortex-M0 and Cortex-M3 Processors

- **The Cortex-M3 processor is ARM's flagship Cortex-M class processor**
  - Offering superior efficiency and flexibility
- **The Cortex-M0 processor is an ultra low-power, streamlined subset of the Cortex-M3**
  - With easy migration to the Cortex-M3 processor
- **Coherent architecture spanning cost/performance points**

Comparison	Cortex-M0	Cortex-M3
DMIPS/MHz	0.9	1.25
Gate count	12k	43k
Number interrupts	1-32 + NMI	1-240 + NMI
Interrupt priorities	4	256
Breakpoints, Watchpoints	4/2, 2/1	8/4, 2/1
MPU, integrated trace option	No	Yes
Hardware Divide	No	Yes



ARM dans les cartes Arduino



Arduino Due

Architecture

ARMv7 Cortex-M3

Processor

Atmel SAM3X8E – 84MHz

RAM

96KB

Arduino Zero

Architecture

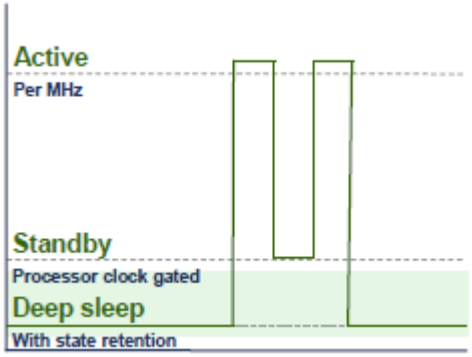
ARMv7 Cortex-M0

Processor

Atmel SAMD21 – 48MHz

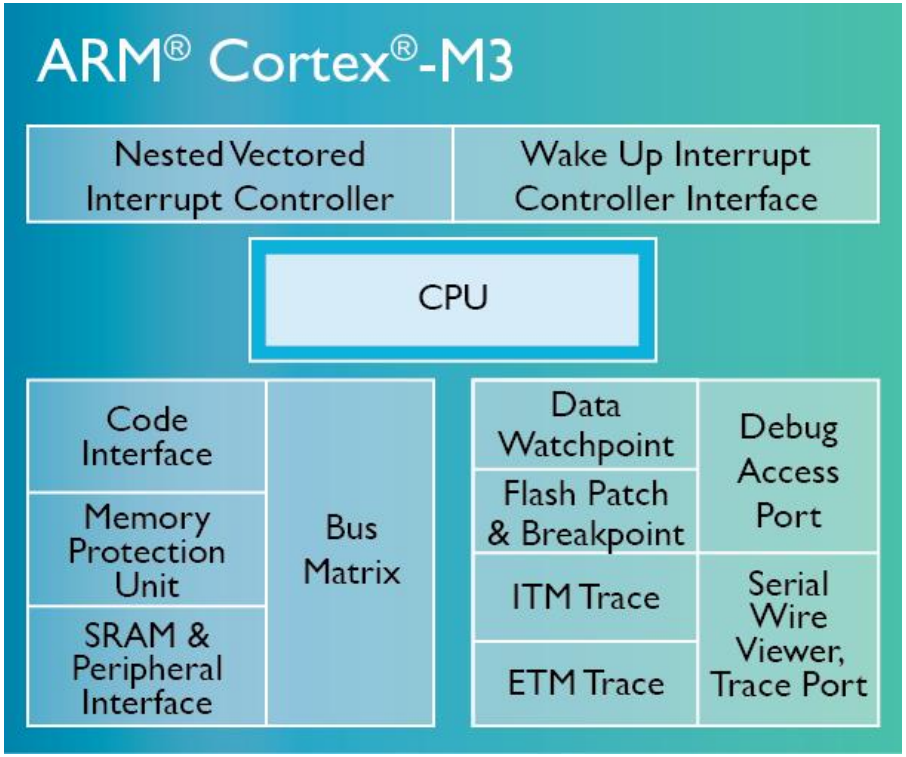
RAM

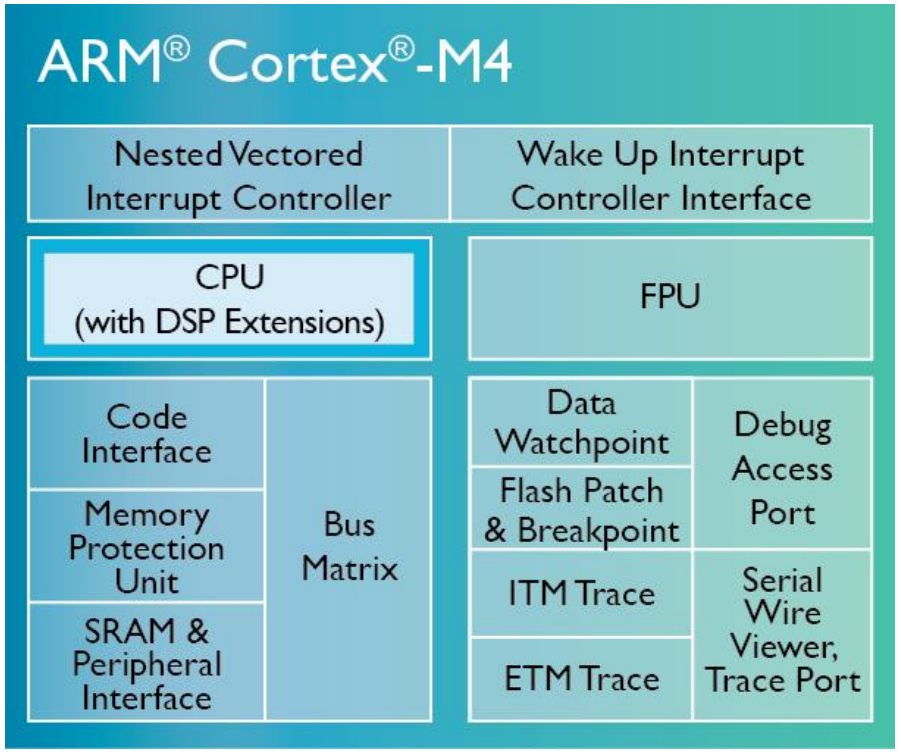
32 KB



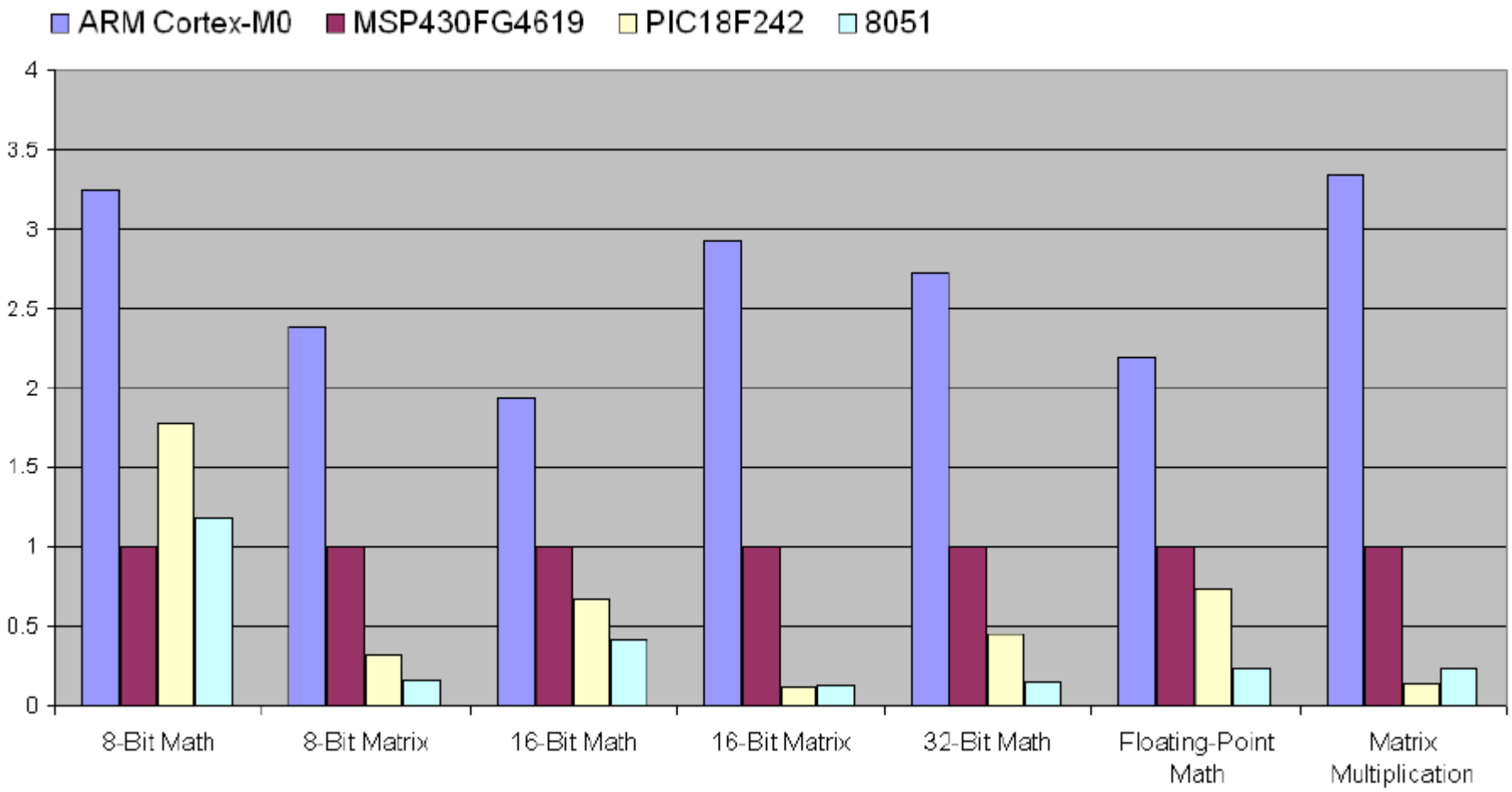
Retour sur la famille Cortex M3

ISA Support	Thumb® / Thumb-2
Pipeline	3-stage
Performance Efficiency	1.25 / 1.50 / 1.89 DMIPS/MHz**
Memory Protection	Optional 8 region MPU
Interrupts	1 to 240 physical interrupts
Interrupt Priority Levels	8 to 256 priority levels

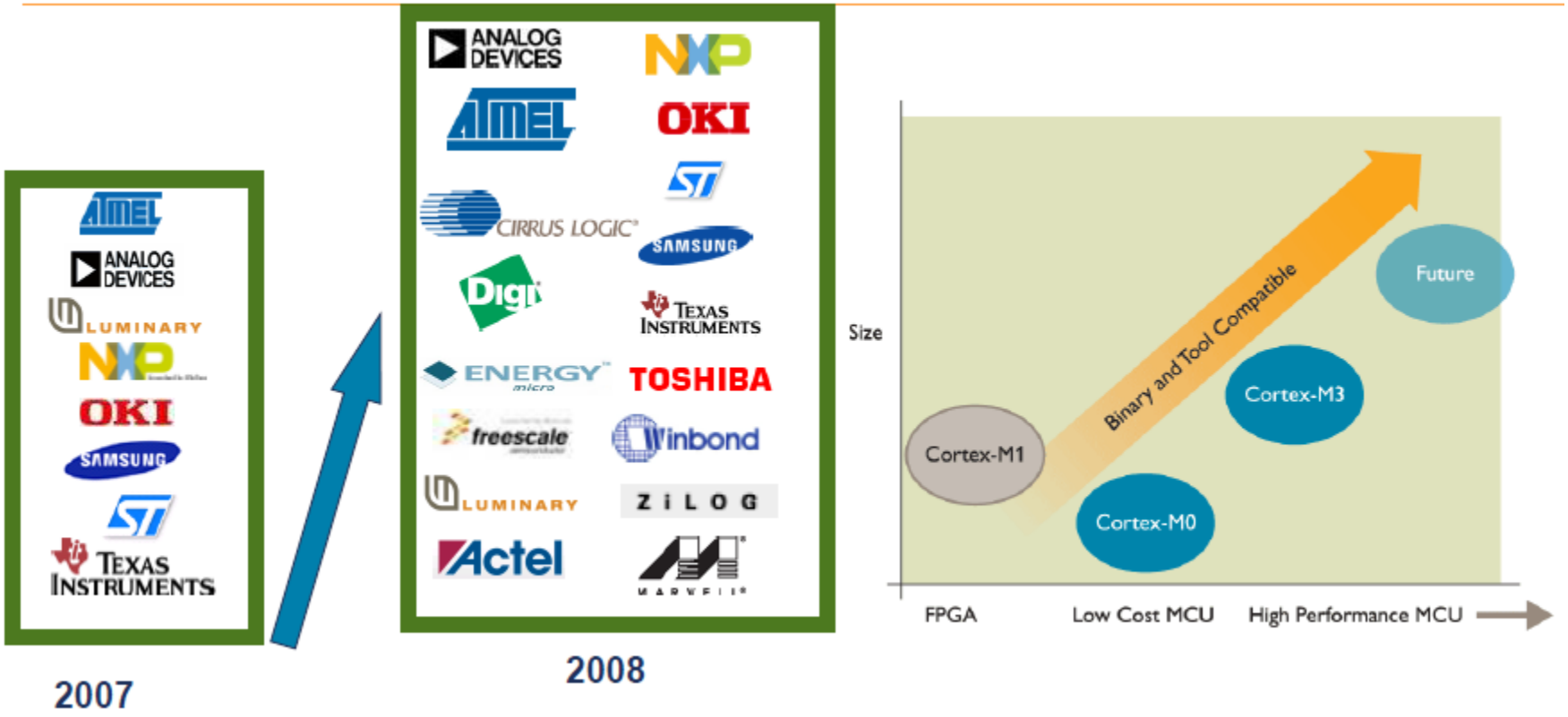




Comparaison de performances



ARM principal fournisseur de solution MCU pour les fondeurs



3. Parallélisme d'exécution

Types de parallélisme :

- ILP : Instruction Level Parallelism
  - Au sein du processeur
- TLP : Task Level Parallelism
  - Entre plusieurs processeurs

3.1 Parallélisme d'instructions - ILP

- Il consiste à multiplier le nombre d'unités de calcul (ALU)
- Au détriment d'un surcout important de contrôle de ces architectures
- On distingue deux types d'architectures selon le temps de la prise de décision des instructions à paralléliser :
  - Superscalaire : parallélisme à l'exécution
  - VLIW : parallélisme à la compilation => Very Long Instruction Width
  - SIMD : Tous les cœurs exécutent la même instruction sur des données différentes (processeurs graphiques)

3.2 Parallélisme de tâches

- Le problème devient différent
- Il ne s'agit plus d'extraire le parallélisme d'un code séquentiel, mais de paralléliser des threads indiqués par le programmeur
  - Le parallélisme est explicité par le programmeur
  - Entre chaque exécution d'un thread une phase de changement de contexte est nécessaire
  - Les variables manipulées par les threads doivent être protégées

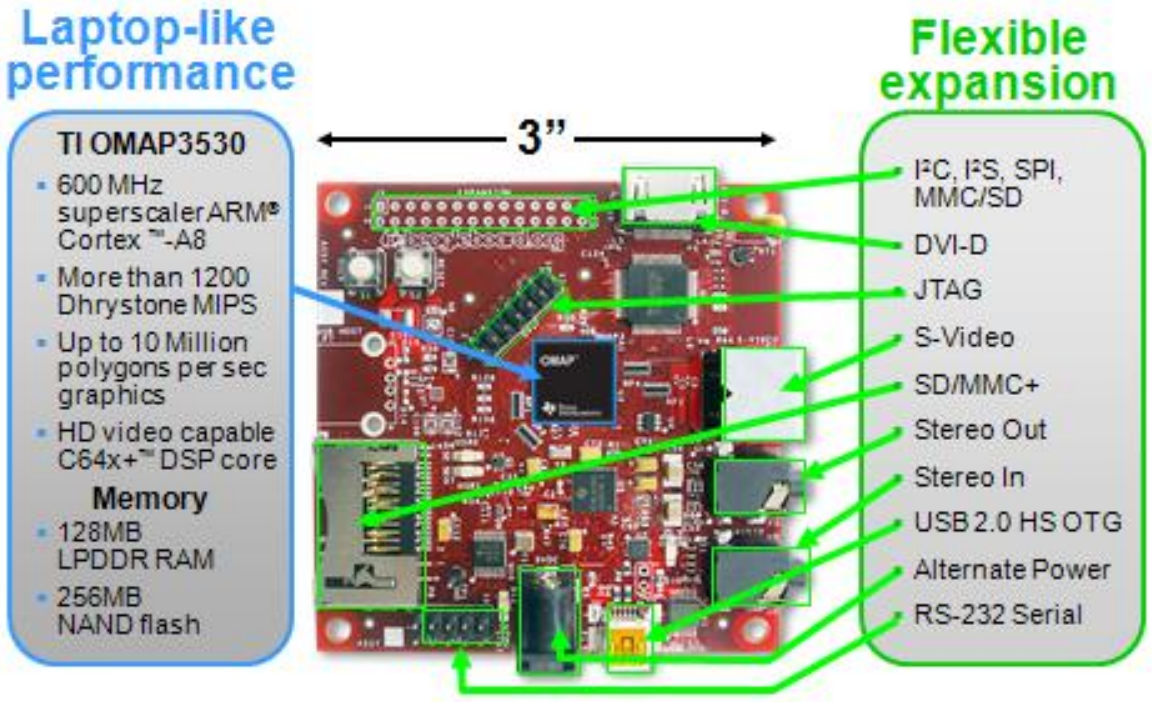
Différents parallélismes de tâches

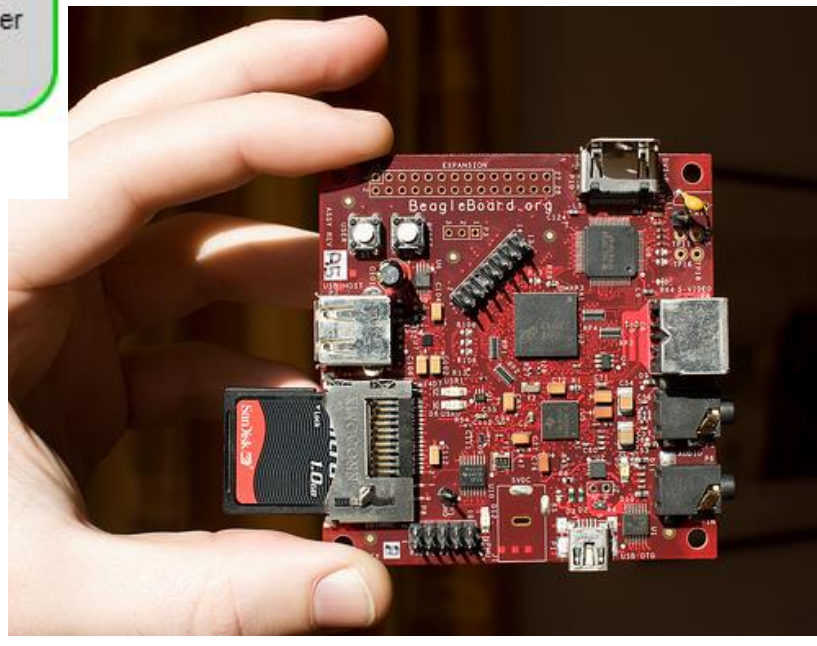
- Les architectures multicoeurs
  - Homogènes, gérées par l'OS
  - Hétérogènes, gérées par le programmeur
  - Manycore, plusieurs centaines de coeurs
- Les processeurs graphiques
  - Également programmables au niveau tâches
- Les architectures matérielles
  - FPGA, en association avec un cœur de processeur

Raspberry Pi 2 B – multicoeur homogènes

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM

BeagleBoard – multicoeur hétérogène

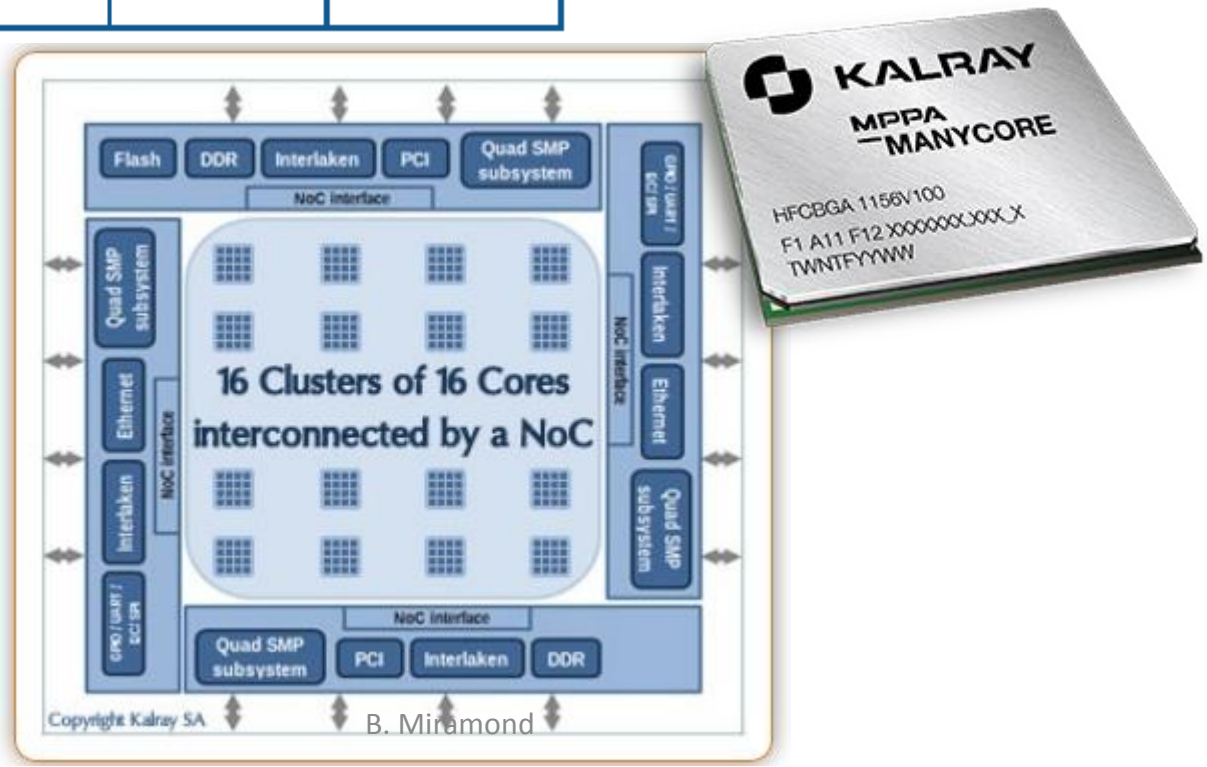




http://beagleboard.org/

Kalray MPPA, manycore

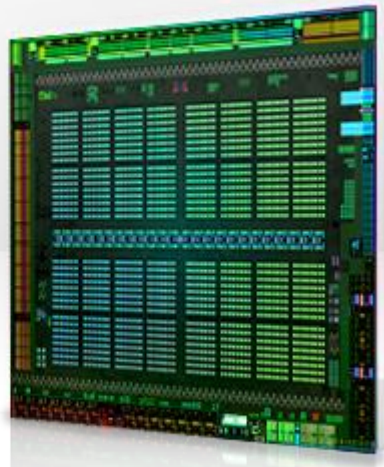
Core Generation	Number of Processing Cores	GFLOPS/W	GOPS/W
Andey	256	25	75
Bostan (2014)	256	50	80
Coolidge (2015)	64/256/1024	75	115



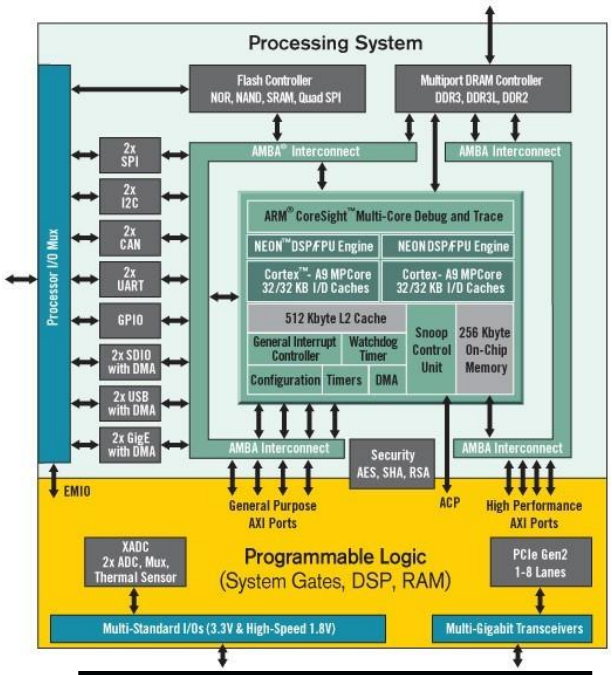
GPU, calcul SIMD

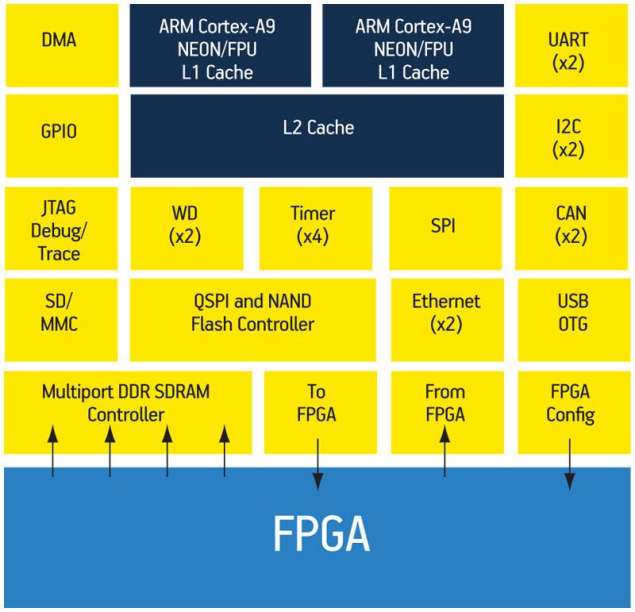
Titan X GPU

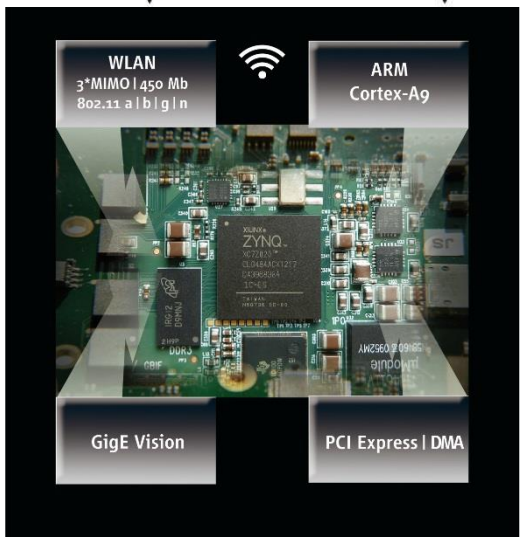
2076 Cores Cuda
1GHz
12GB memory
250 W



FPGA SoC

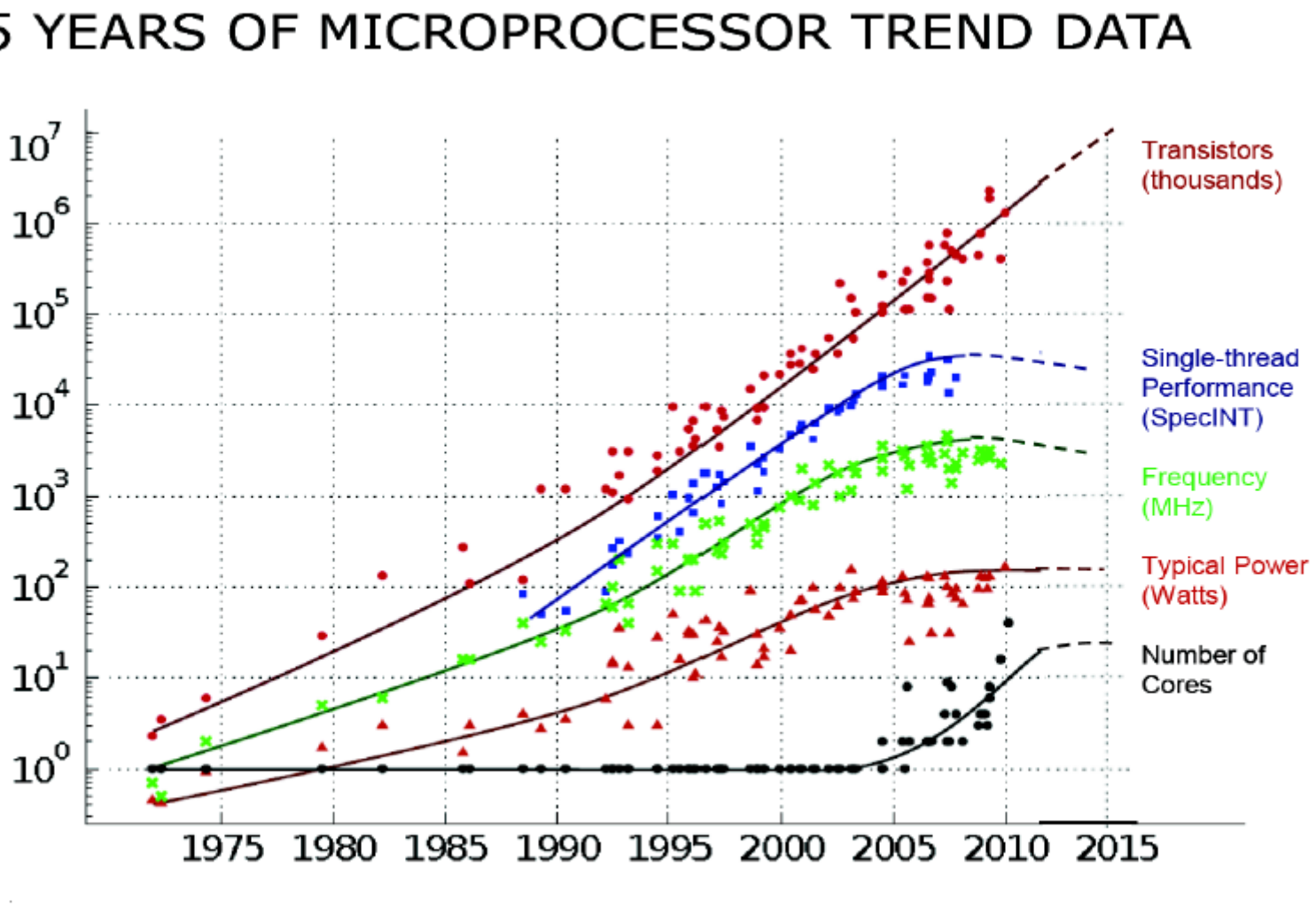

Processing System
Flash Controller
NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller
DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Snoop Control Unit
256 Kbyte On-Chip Memory
Configuration Timers
DMA
AMBA Interconnect
Security AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic
(System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers


DMA
ARM Cortex-A9 NEON/FPU L1 Cache
ARM Cortex-A9 NEON/FPU L1 Cache
UART (x2)
GPIO
L2 Cache
I2C (x2)
JTAG Debug/ Trace
WD (x2)
Timer (x4)
SPI
CAN (x2)
SD/ MMC
QSPI and NAND Flash Controller
Ethernet (x2)
USB OTG
Multiport DDR SDRAM Controller
To FPGA
From FPGA
FPGA Config
FPGA


WLAN
3*MIMO | 450 Mb
802.11 a | b | g | n
ARM
Cortex-A9
ZYNQ
GigE Vision
PCI Express | DMA


ARM
Arria V
Cyclone V
ALTERA
SoC FPGA

Conclusion sur la performance des CPU



Architecture des ordinateurs

The end