Informatique temps réel

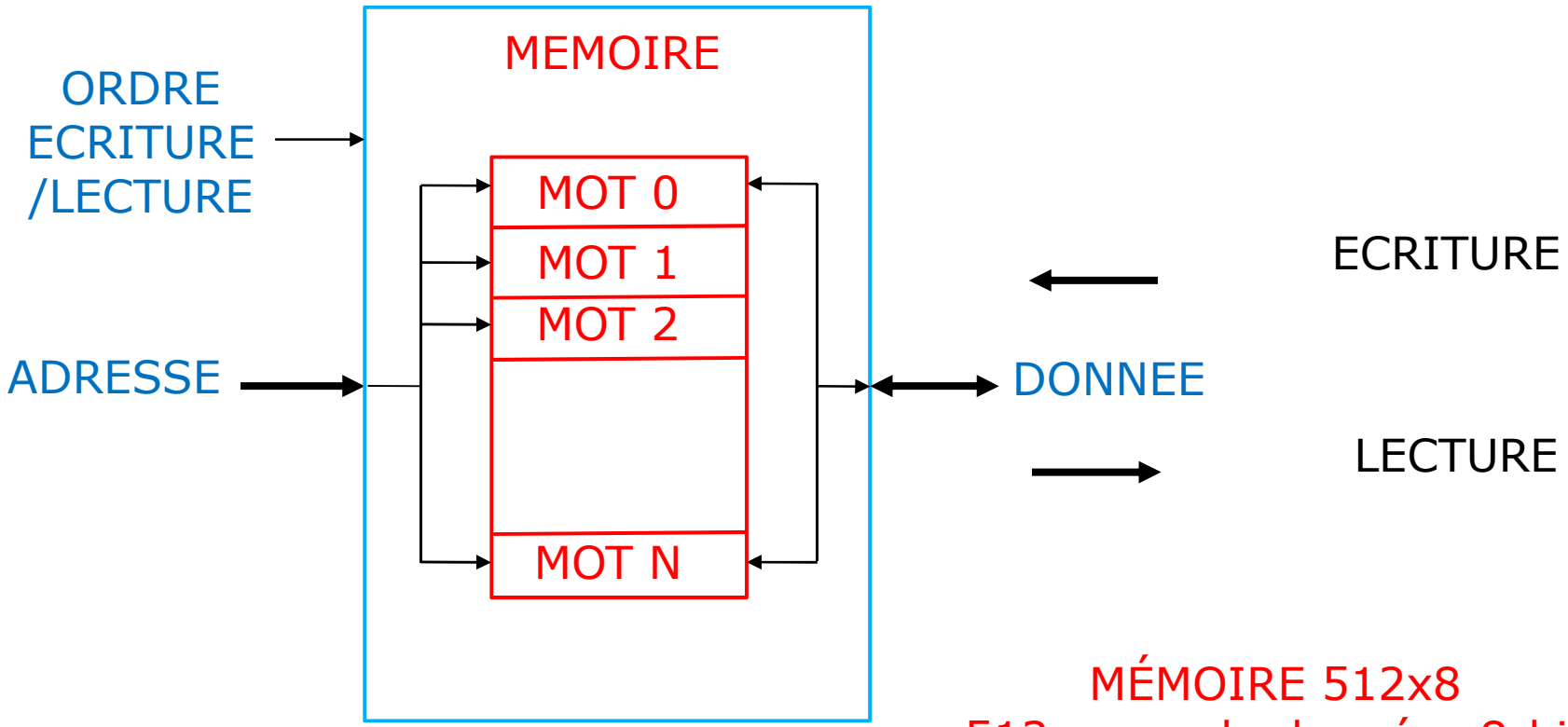
Hervé Le Provost

DSM/IRFU - CEA Saclay
herve.le-provost@cea.fr

avec la participation de Shebli Anvar
shebli.anvar@cea.fr

Version 2.2 – 11 Février 2011

Circuits numériques : la mémoire



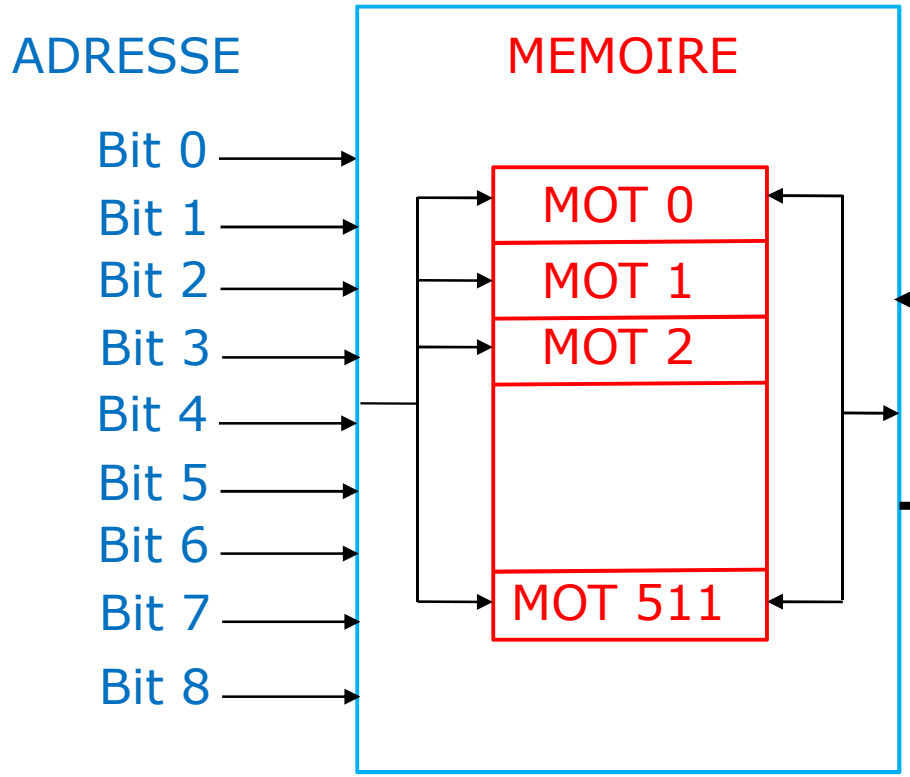
MÉMOIRE 512x8
512 cases de données 8-bit
Largeur du bus d'adresse ?

herve.le-provost@cea.fr

Circuits numériques : la mémoire

ECRITURE à l'adresse 48 de la valeur 6
/ LECTURE

```
int main() {
char *adresse;
char donnee=6;
adresse=(char*)0x30;
*adresse=donnee; //ecriture
....
donnee=*adresse;//lecturE
}
```



8 ECRITURE

DONNEE

8 LECTURE

MÉMOIRE 512x8
Largeur du bus d'adresse
2exp(9)=512

herve.le-provost@cea.fr

Circuits numériques : la mémoire

- Mémoires non Volatiles
  - ROM (Read Only Memory), programmée à la fabrication
  - PROM (Programmable ROM), programmée une seule fois par l'utilisateur.
  - EPROM (Erasable PROM), programmable plusieurs fois par l'utilisateur sur des machines dédiées
  - EEPROM (Electrically Erasable PROM), programmable plusieurs fois par l'utilisateur in-situ
    - Les plus utilisées : FLASH, BIOS carte mère PC. Rapides et haute densité (plusieurs Méga octets)
  - NVRAM (Non volatile Random Access Memory)
- Mémoires Volatiles
  - RAM (Random Access Memory), SRAM, SDRAM, DDR2, DDR3.

herve.le-provost@cea.fr

Circuits numériques : codage des valeurs

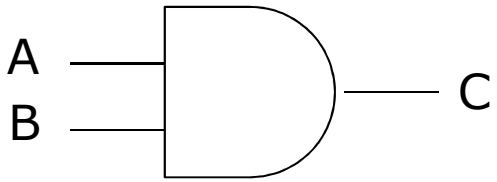
décimal	binaire	Hexa-décimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

décimal	binaire	Hexa-décimal
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Valeur décimale 47, codage binaire et hexadécimal ?

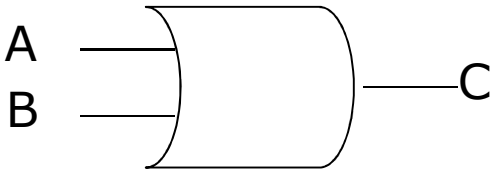
herve.le-provost@cea.fr

Circuits numériques : portes logiques



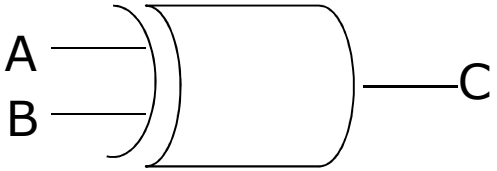
ET

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1



OU

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

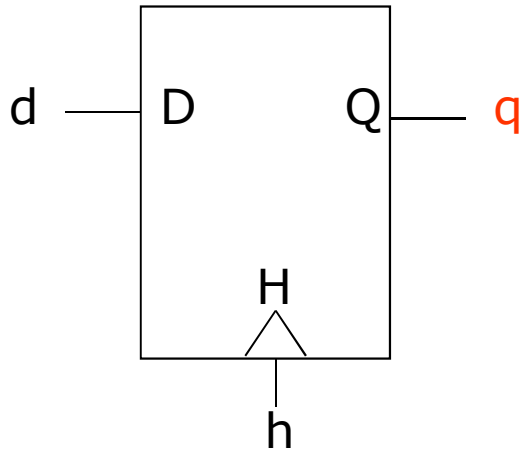


OU EXCLUSIF

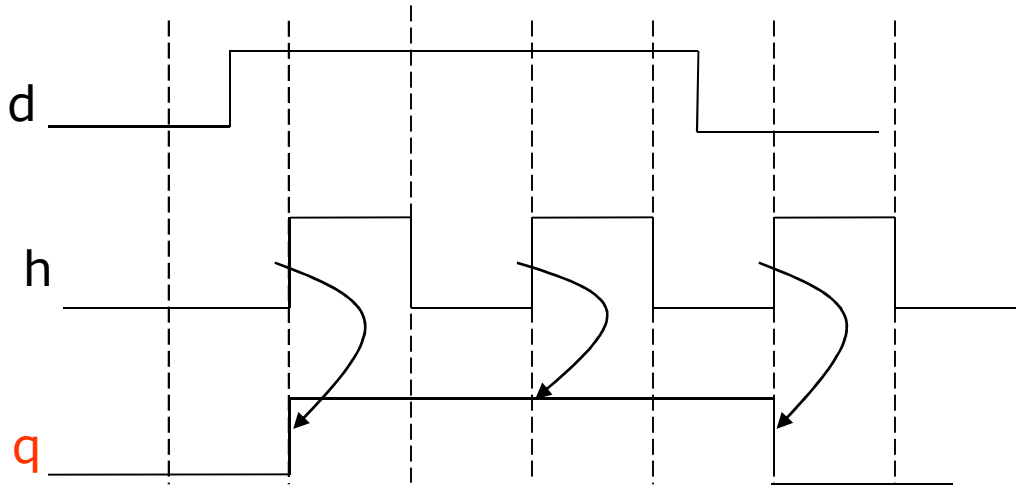
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

herve.le-provost@cea.fr

Circuits numériques : Bascule D



Bascule D



Chronogramme

h	D	Q_{n+1}
↑	0	0
↑	1	1
↓	x	Q_n
0	x	Q_n
1	x	Q_n

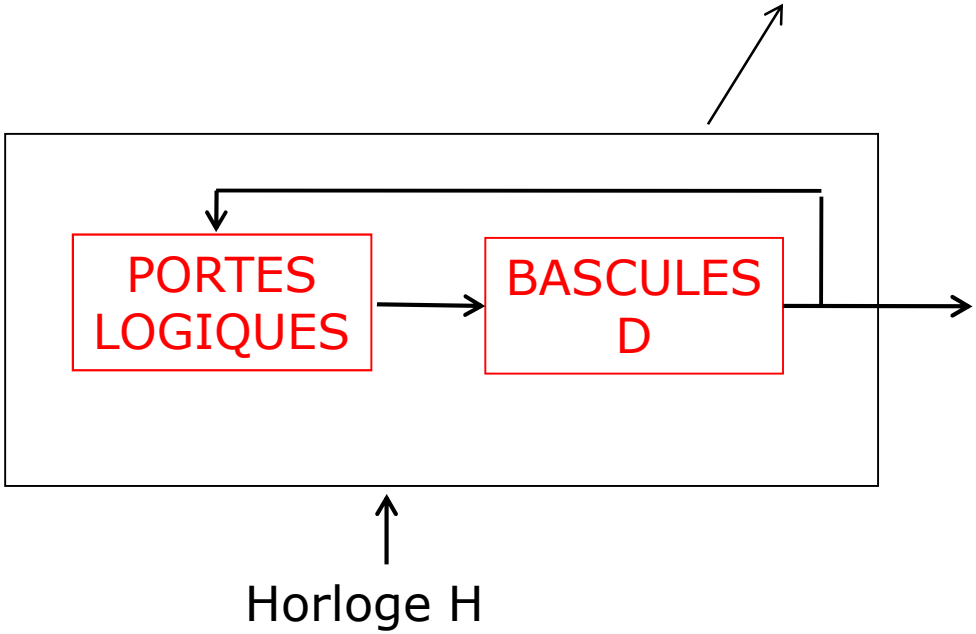
Table de vérité

herve.le-provost@cea.fr

Circuits numériques : Design Synchrone

EXEMPLE : UN COMPTEUR

FPGA :
FIELD PROGRAMMABLE GATE ARRAY



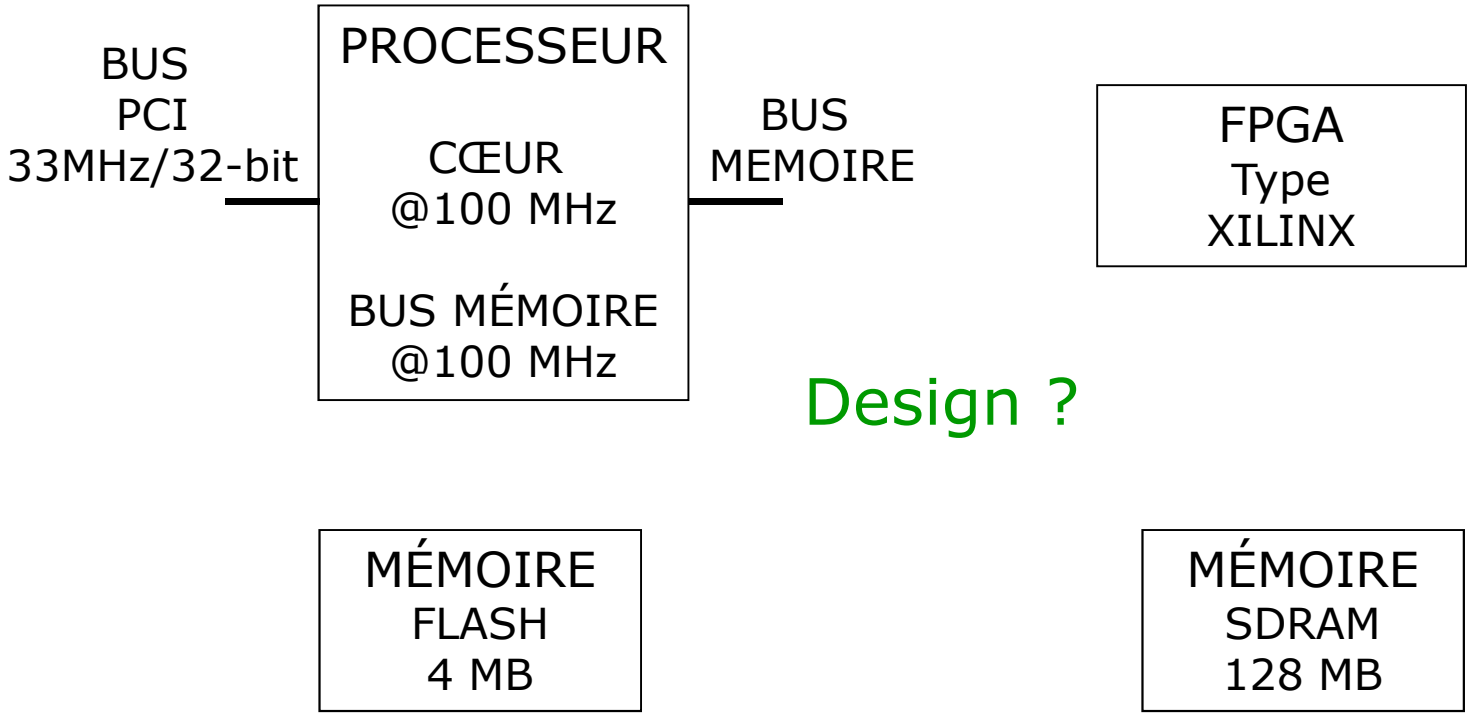
SORTIE
S=S+1
EVOLUTION SUR CHAQUE FRONT MONTANT DE L'HORLOGE H

FREQUENCE F=50 MHz, COMPTEUR S CODE SUR 8 BIT => QUELLE EST LA DUREE ECOULEE ENTRE 2 PASSAGE A 0 DU COMPTEUR ?

herve.le-provost@cea.fr

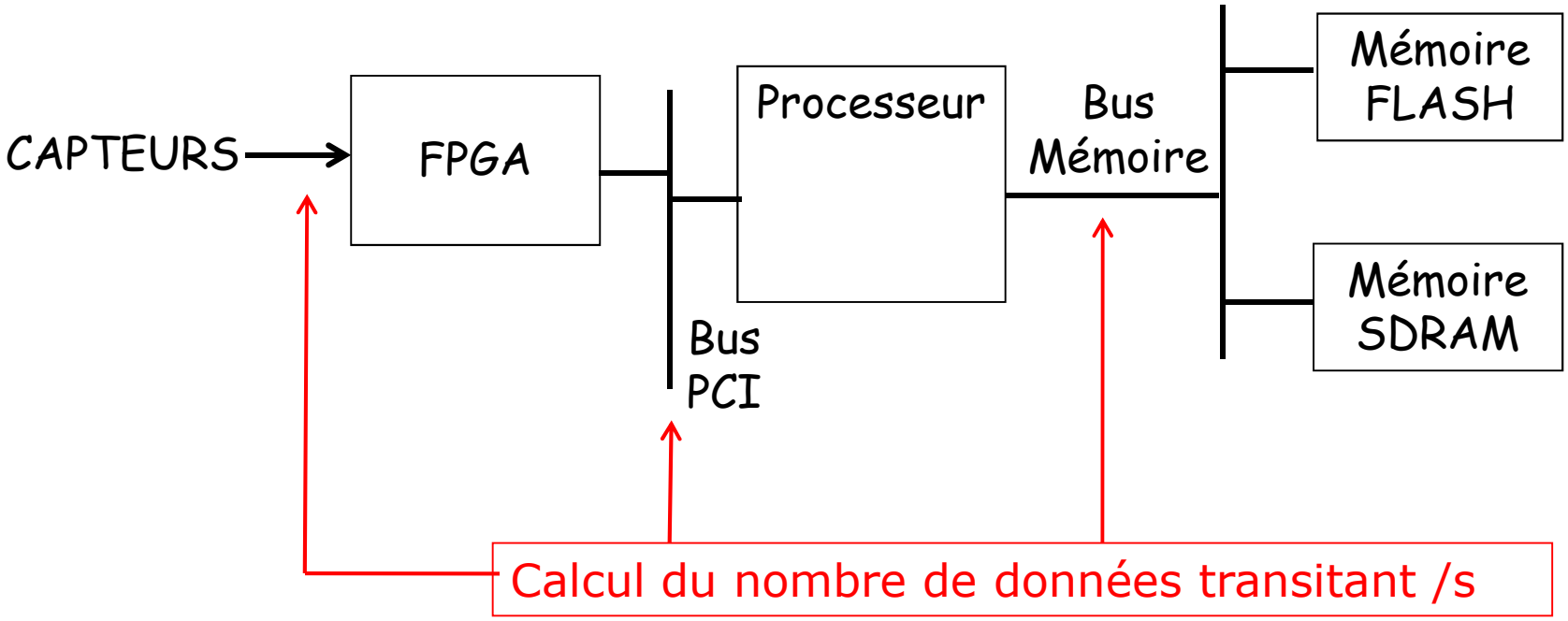
Circuits numériques : un design, un LEGO

DESIGN : LECTURE DE CAPTEURS (FORMAT PROPRIETAIRE)
ET TRAITEMENT DES DONNEES PAR UN PROCESSEUR
TOURNANT UN SYSTÈME D'EXPLOITATION



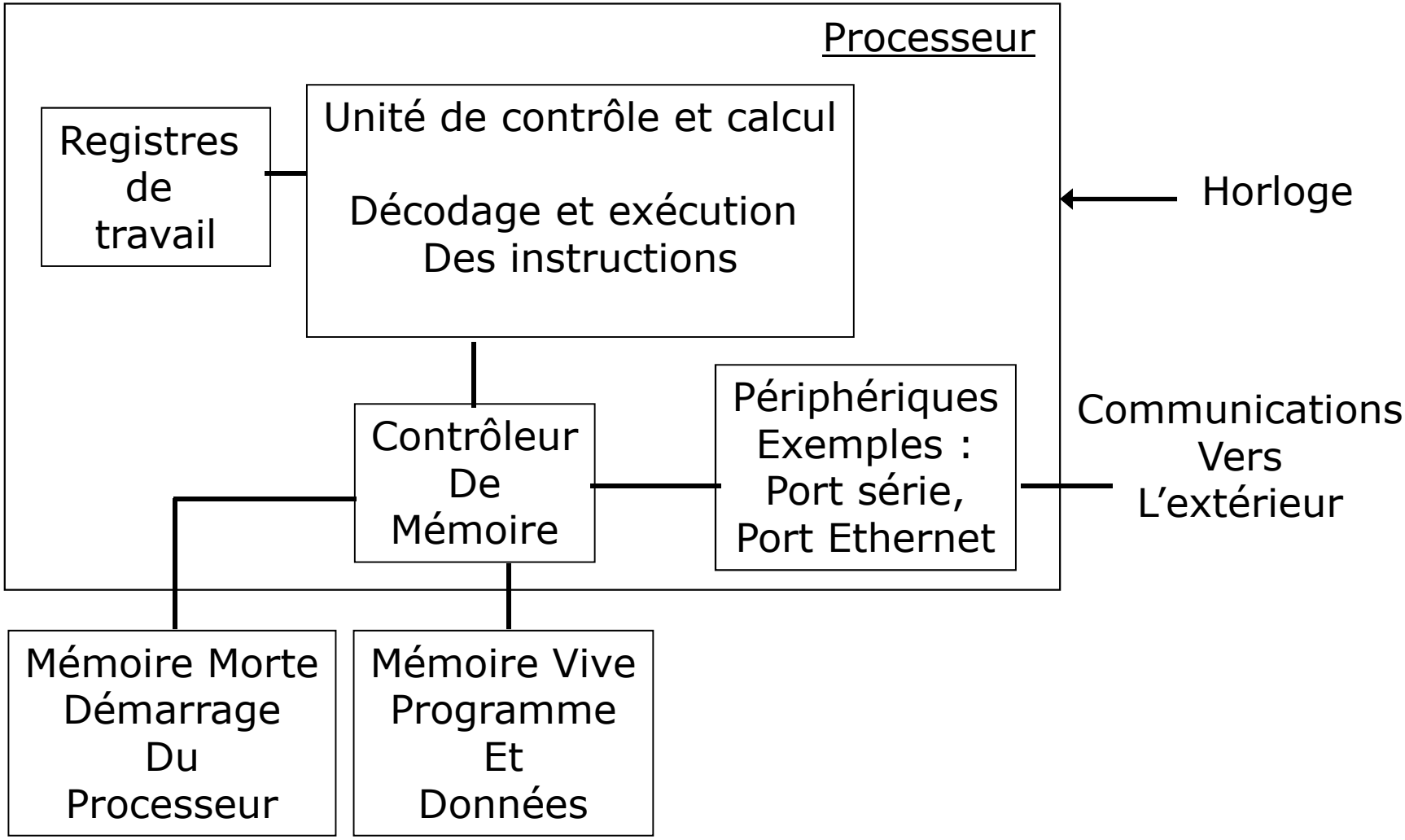
herve.le-provost@cea.fr

Circuits numériques : un design, un LEGO



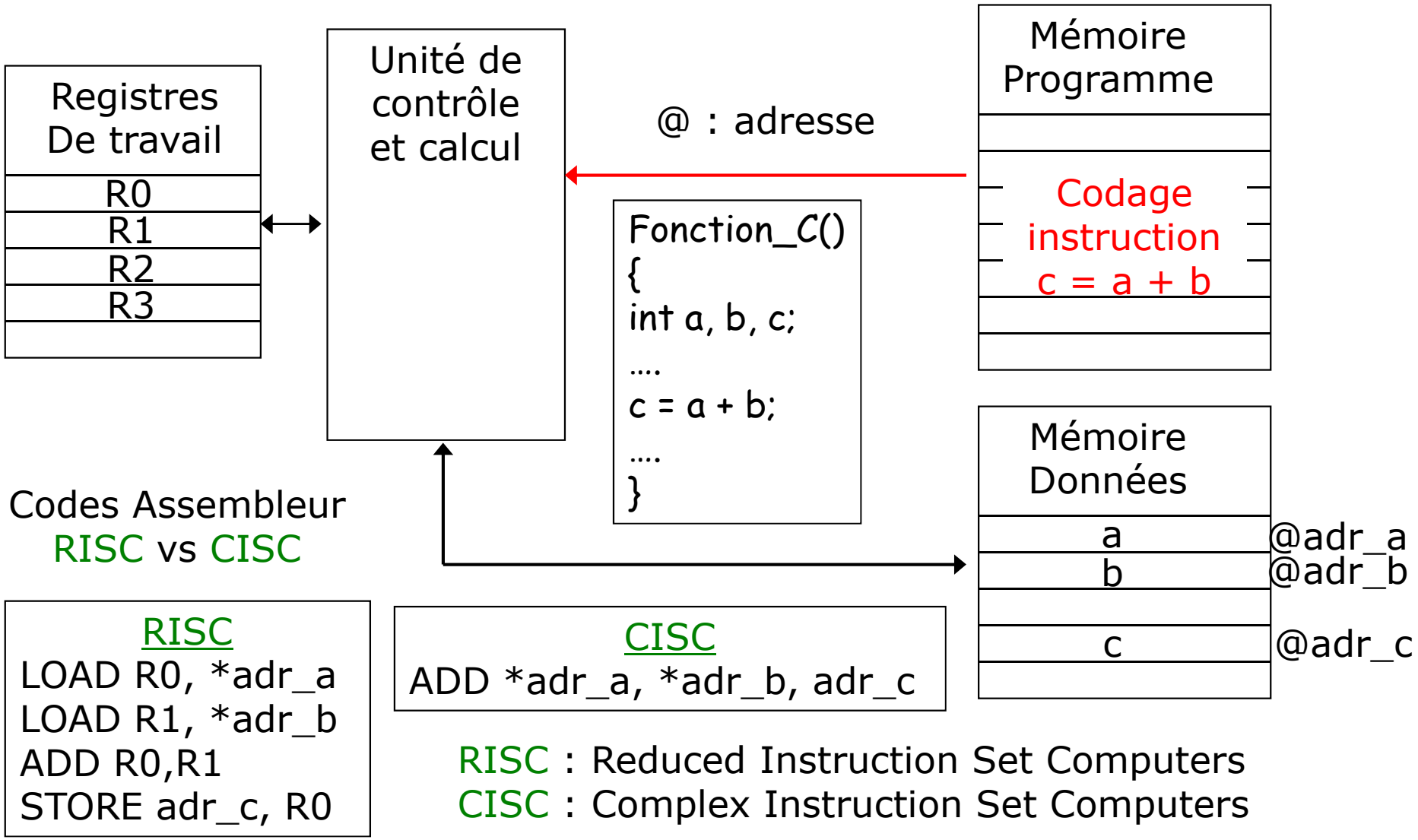
herve.le-provost@cea.fr

Architecture générique d'une carte processeur



herve.le-provost@cea.fr

Processeur et codage assembleur



Codes Assembleur
RISC vs CISC

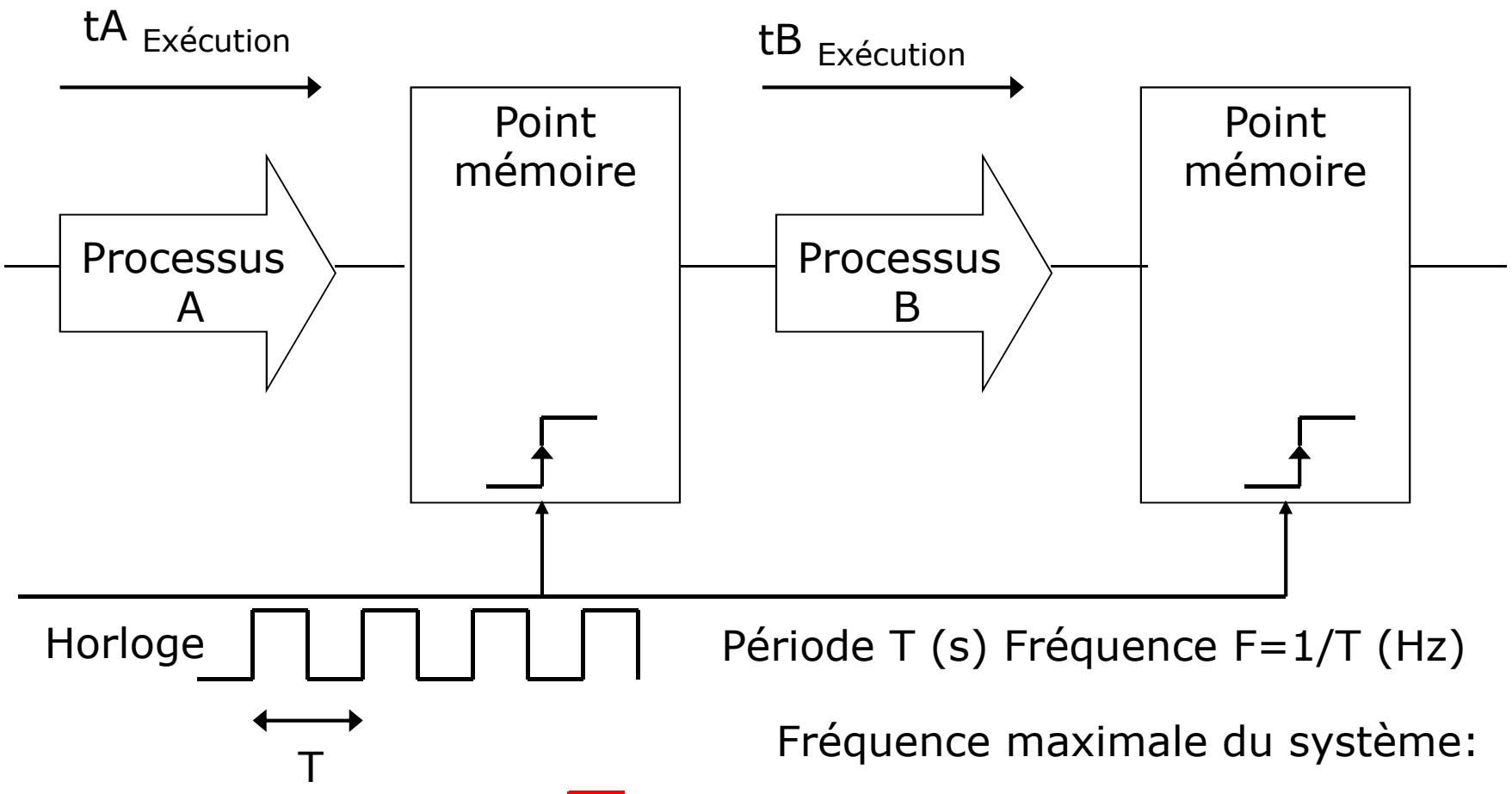
RISC
LOAD R0, *adr_a
LOAD R1, *adr_b
ADD R0,R1
STORE adr_c, R0

CISC
ADD *adr_a, *adr_b, adr_c

RISC : Reduced Instruction Set Computers
CISC : Complex Instruction Set Computers

herve.le-provost@cea.fr

Fréquence maximale de fonctionnement



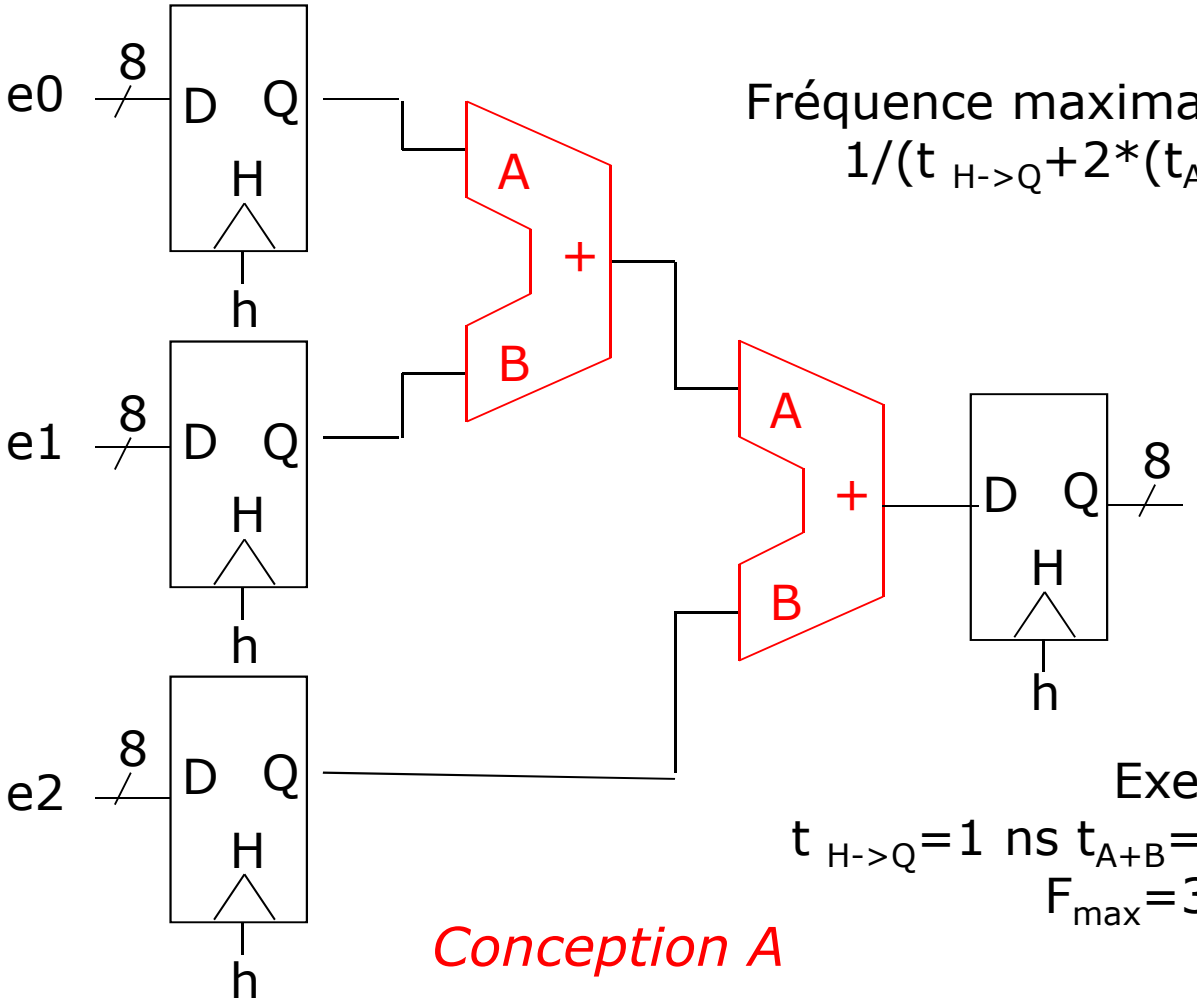
Période T (s) Fréquence F=1/T (Hz)

Fréquence maximale du système:

F_{max}=1/max(tA Exécution, tB Exécution)

herve.le-provost@cea.fr

Fréquence maximale de fonctionnement



Fréquence maximale du système :

$1/(t_{H->Q}+2*(t_{A+B})+t_{D\text{-}setup})$

Exemple:

$t_{H->Q}$=1 ns t_{A+B}=15 ns $t_{D\text{-}setup}$= 1 ns

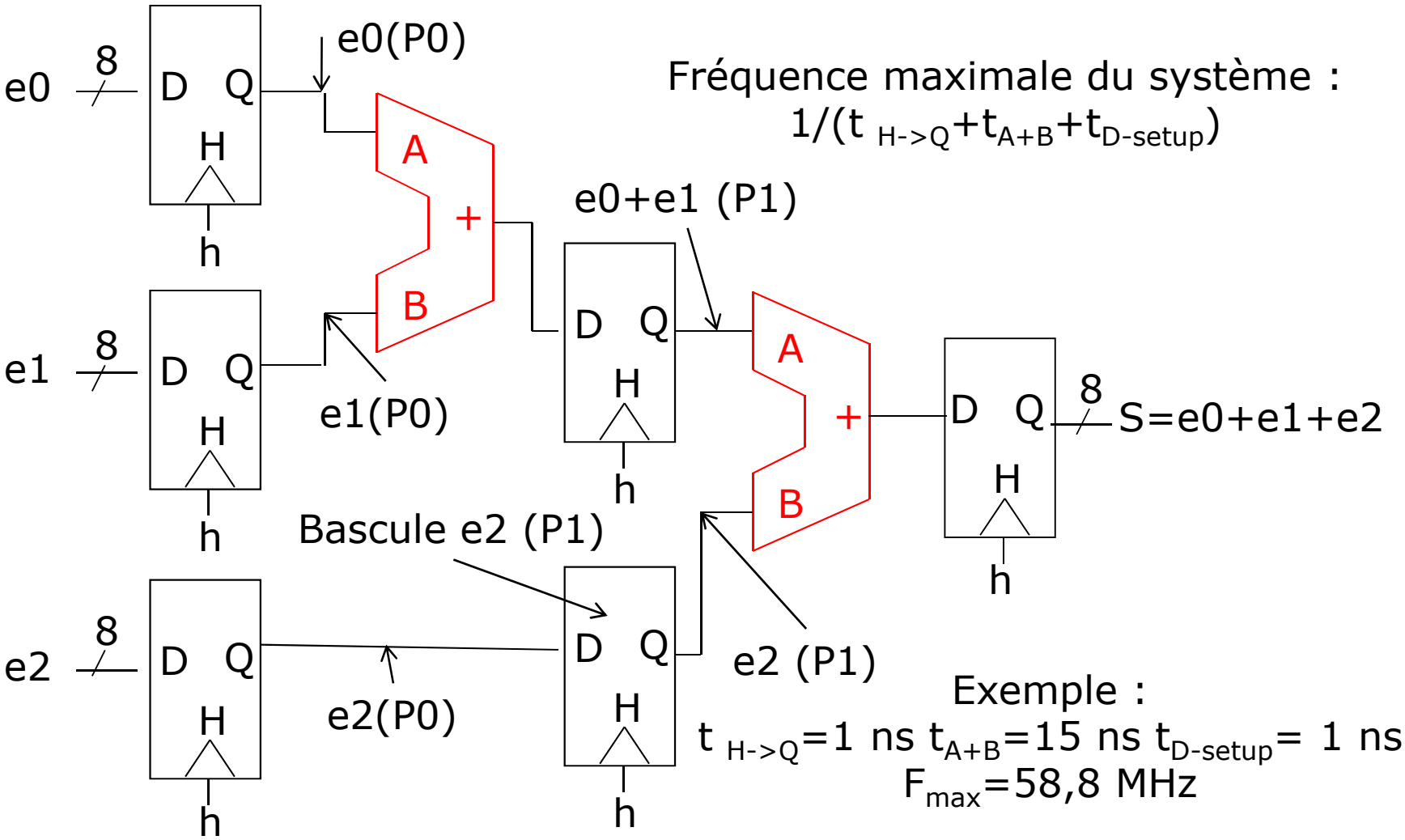
F_{max}=31,2 MHz

Conception A

Proposer une solution pour augmenter la fréquence maximale de fonctionnement

herve.le-provost@cea.fr

Fréquence maximale de fonctionnement



Fréquence maximale du système :

$1/(t_{H->Q}+t_{A+B}+t_{D-setup})$

Exemple :

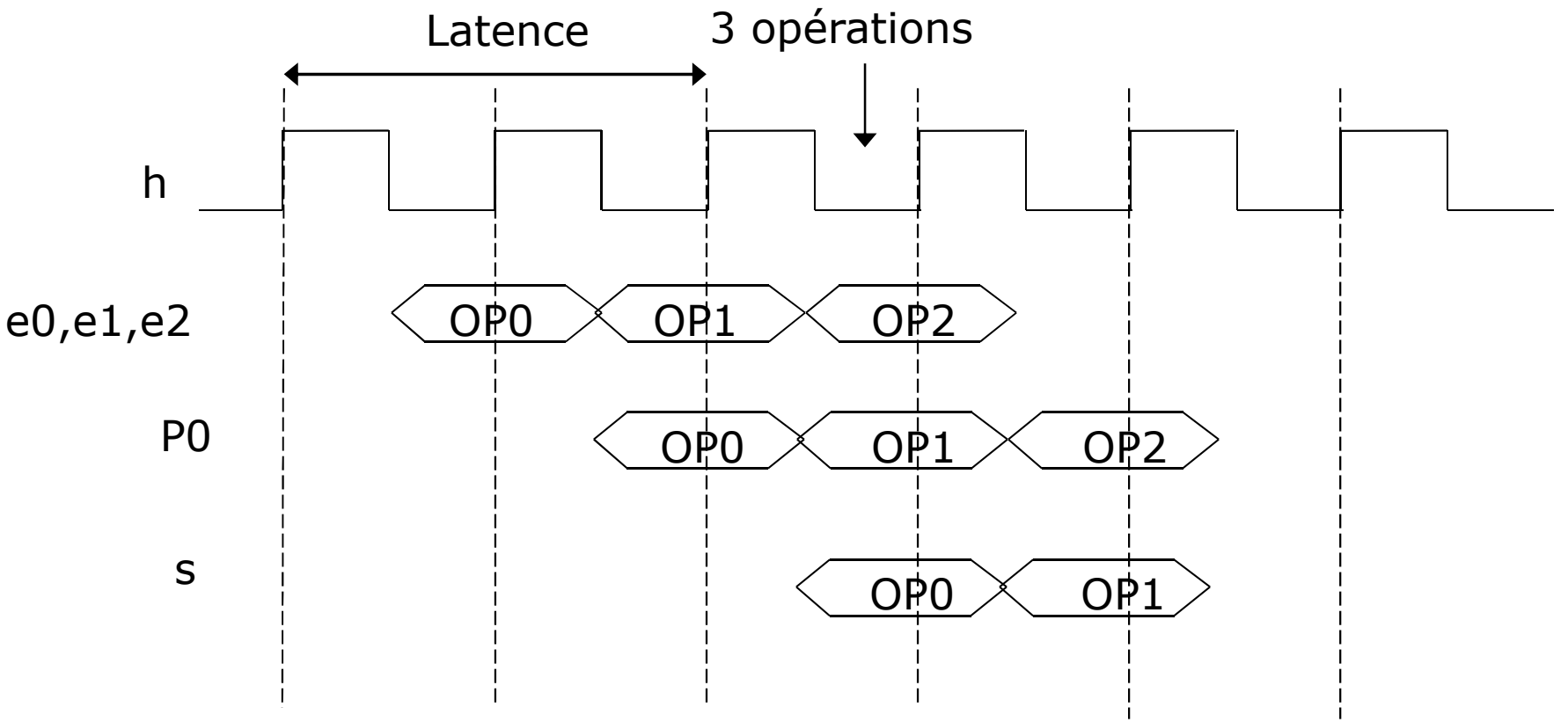
$t_{H->Q}$=1 ns t_{A+B}=15 ns $t_{D-setup}$= 1 ns

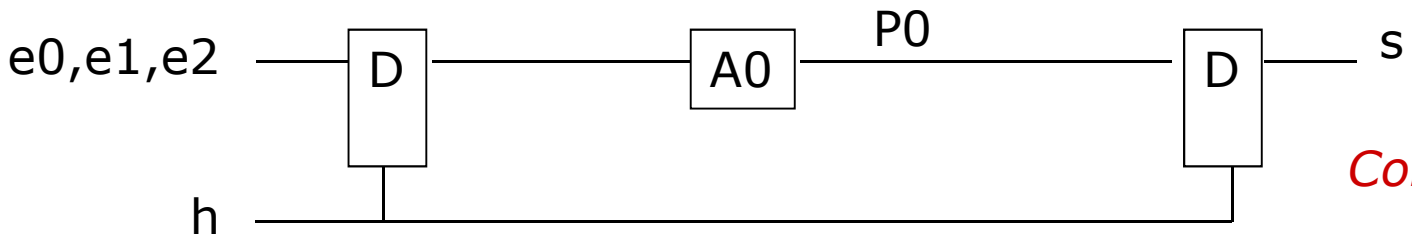
F_{max}=58,8 MHz

Conception B: ajout d'un étage de pipeline

herve.le-provost@cea.fr

Fréquence maximale de fonctionnement

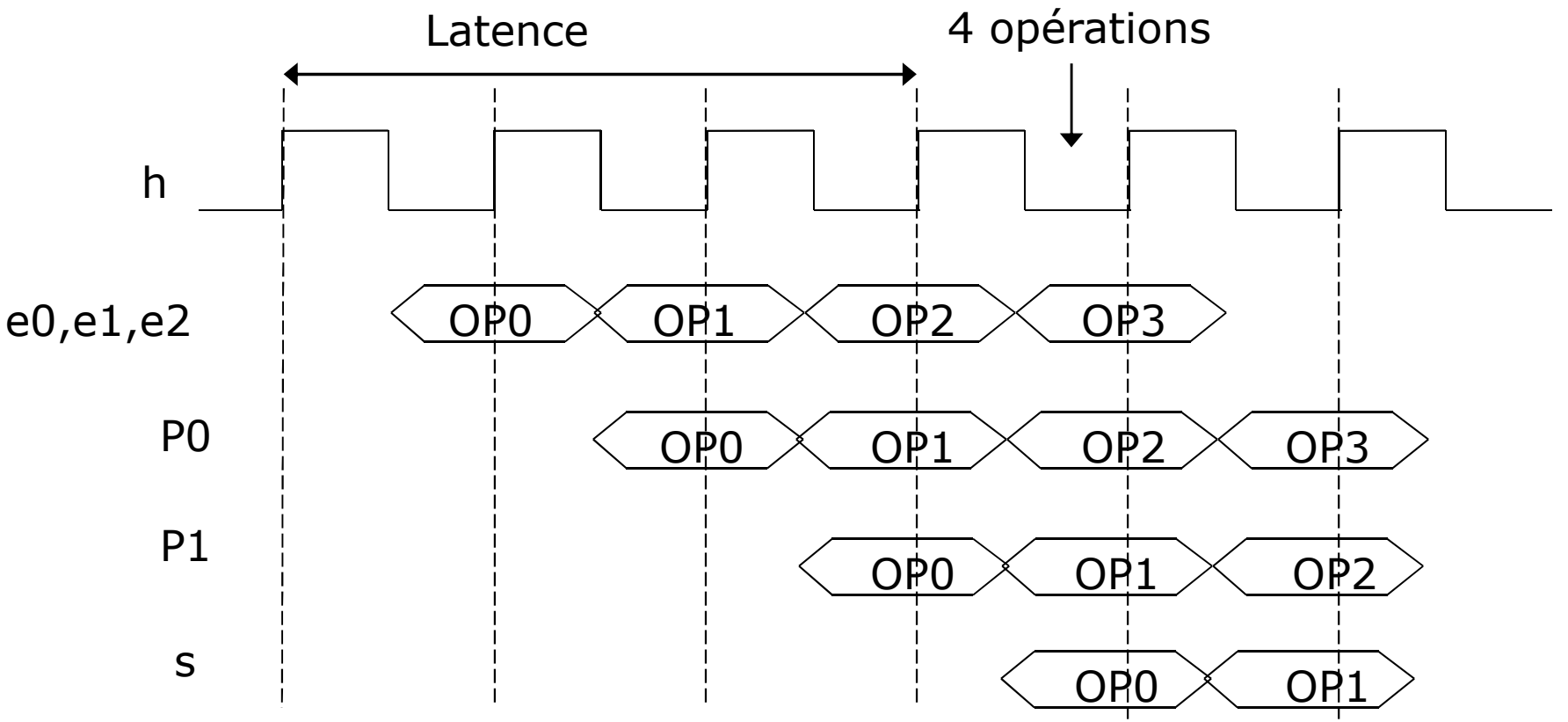


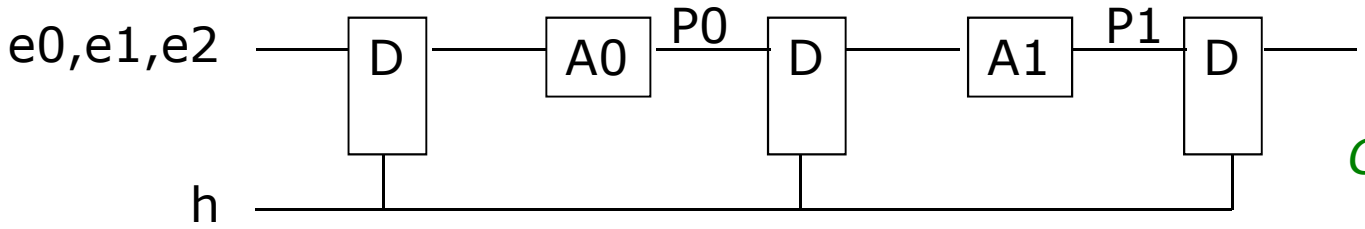


Conception A

herve.le-provost@cea.fr

Fréquence maximale de fonctionnement



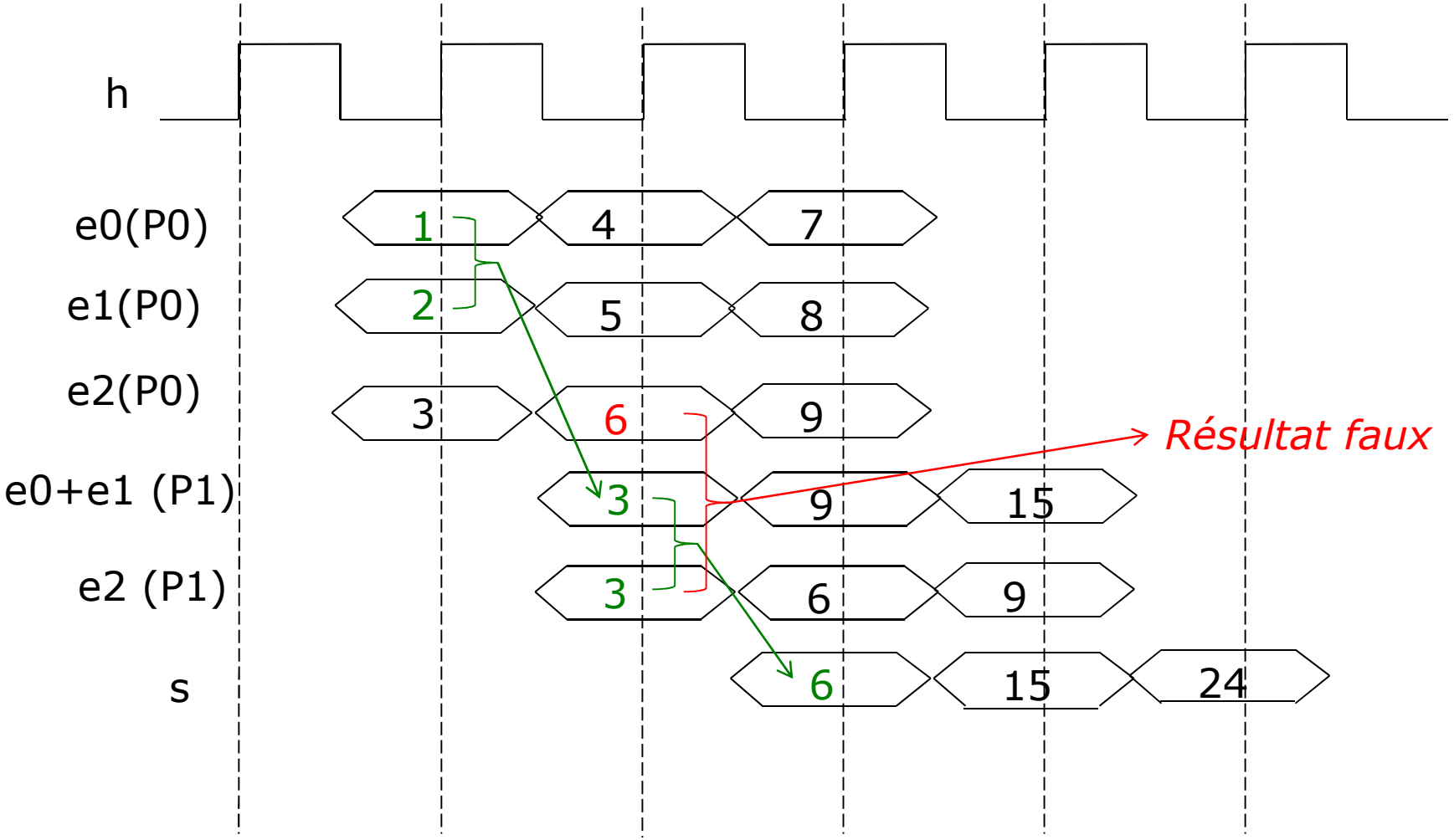


Conception B

herve.le-provost@cea.fr

Fréquence maximale de fonctionnement

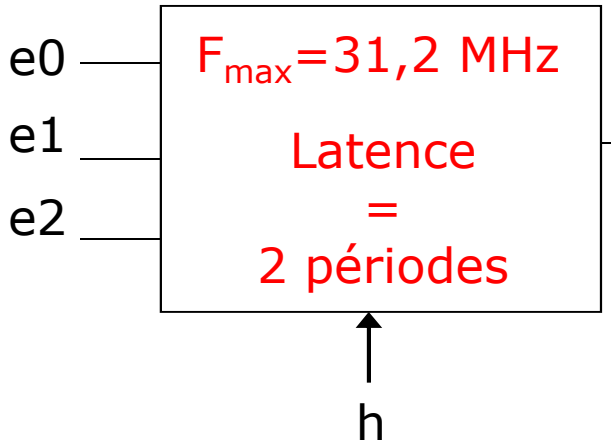
Conception B : de l'utilité de la bascule e2 (P1)



herve.le-provost@cea.fr

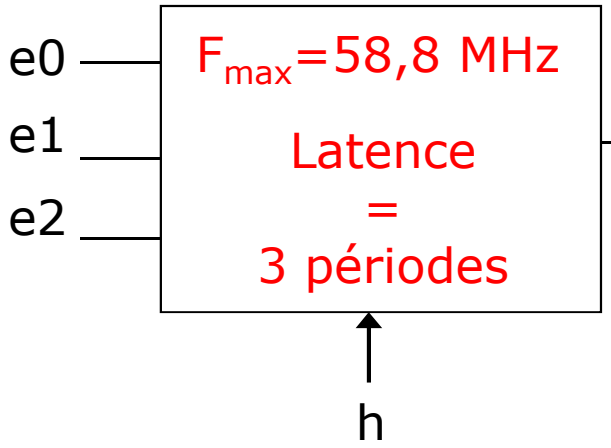
Fréquence maximale de fonctionnement

Conception A



1. Des opérandes toutes les 32 ns. Un résultat toutes les 32 ns
2. Le temps de calcul d'une opération est de 2x32ns=64ns

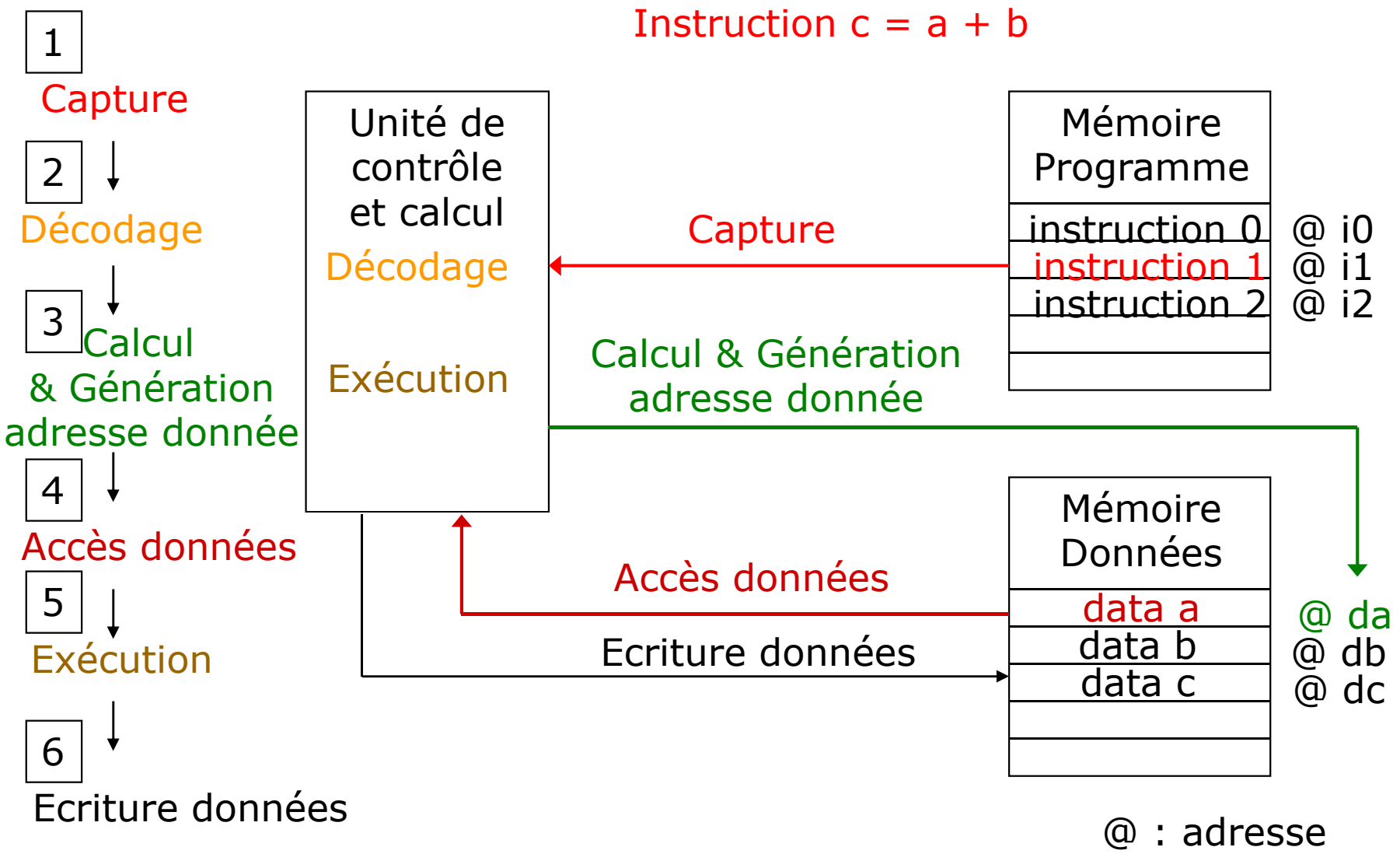
Conception B



1. Des opérandes toutes les 17 ns. Un résultat toutes les 17 ns
2. Le temps de calcul d'une opération est de 3x17ns=51ns

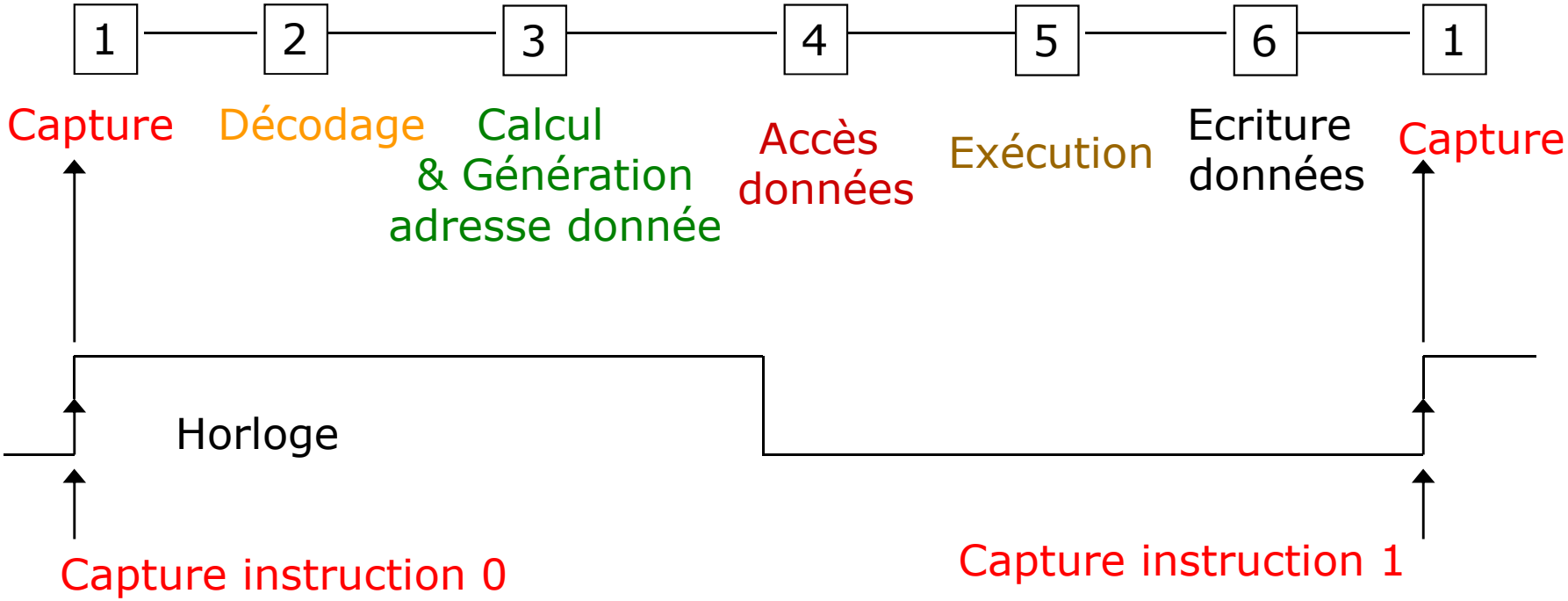
herve.le-provost@cea.fr

Exécution d'une instruction



herve.le-provost@cea.fr

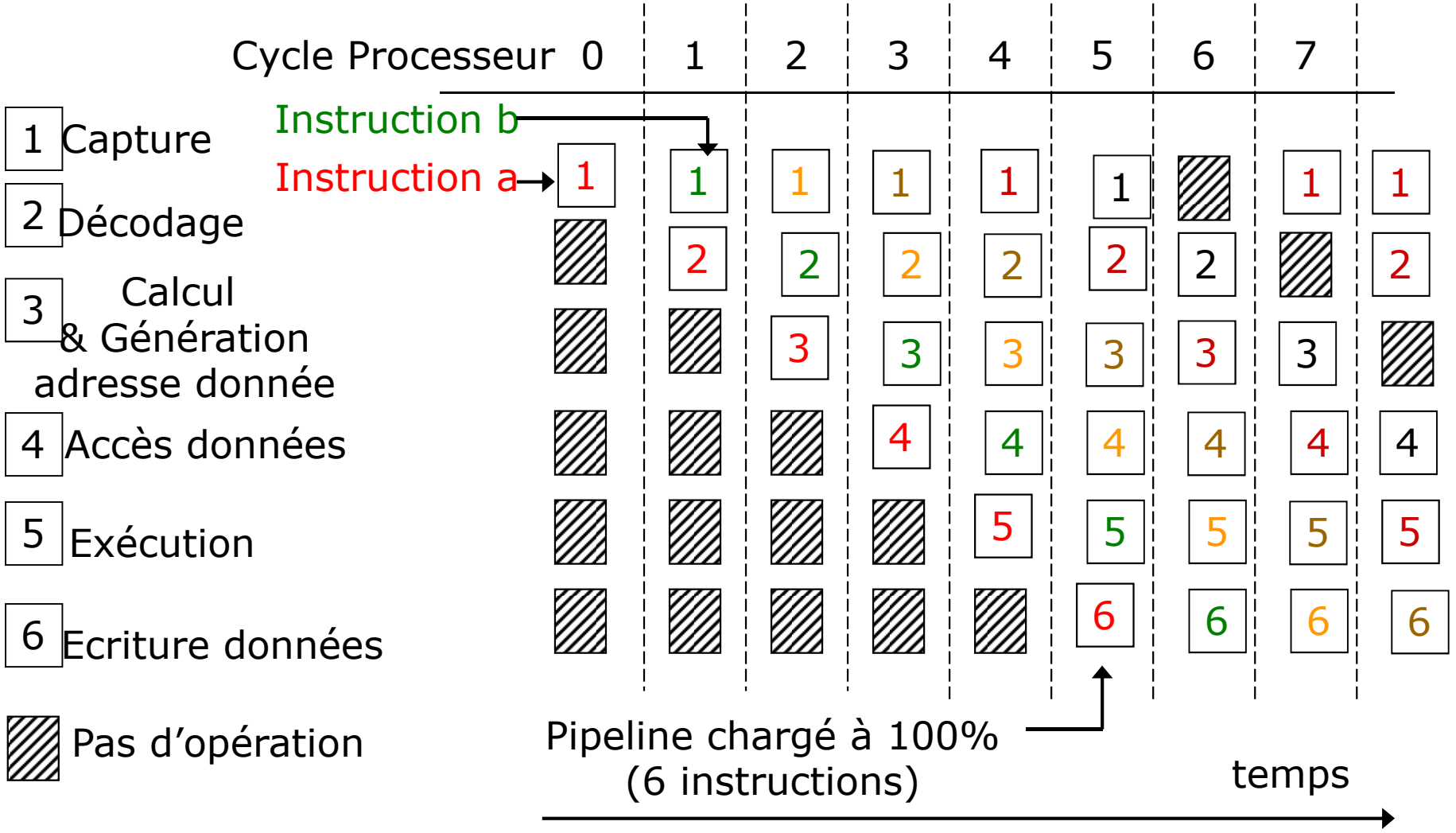
Exécution séquentielle des instructions



Comment augmenter la fréquence maximale de capture des instructions ?

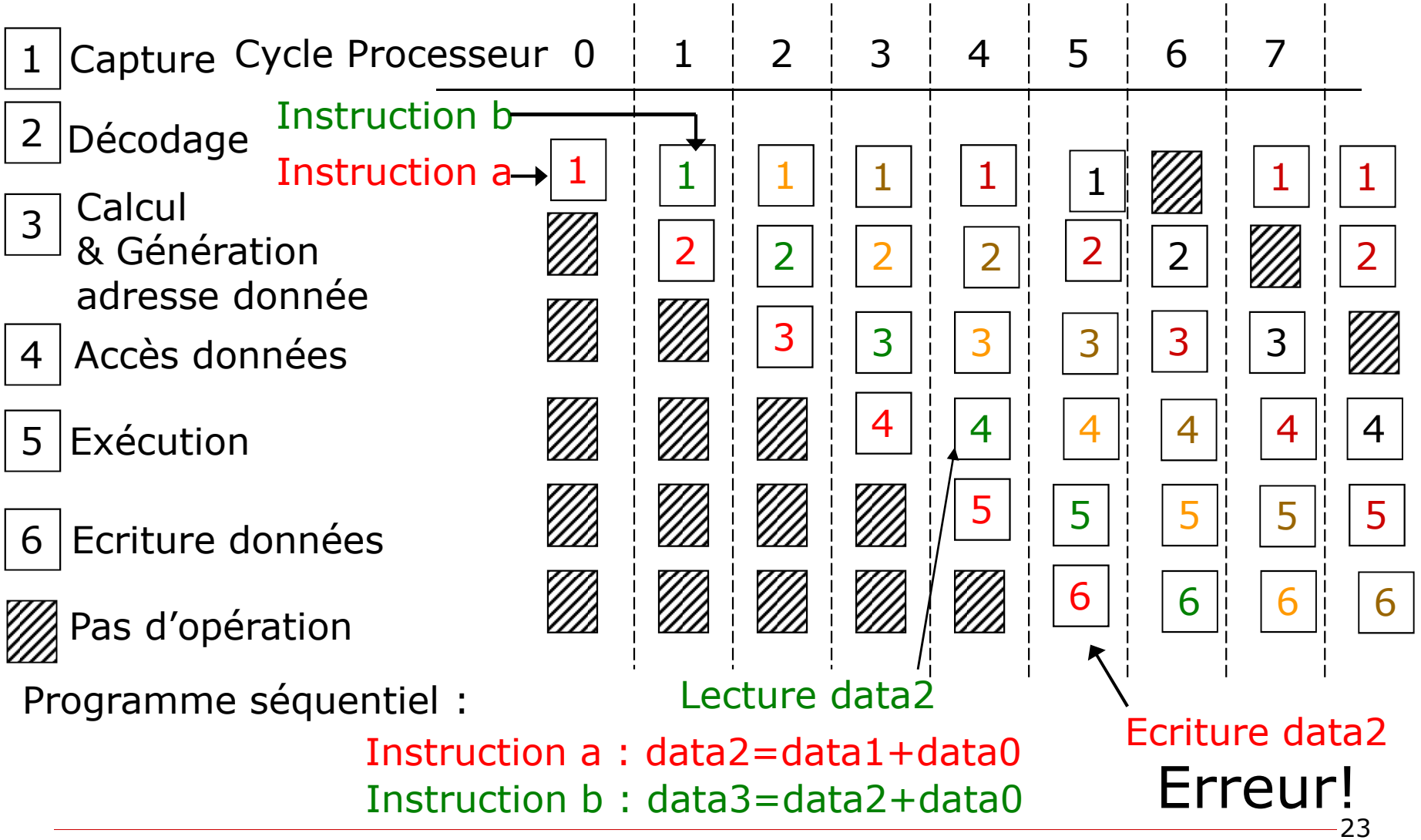
herve.le-provost@cea.fr

Exécution en pipeline des instructions - Principe



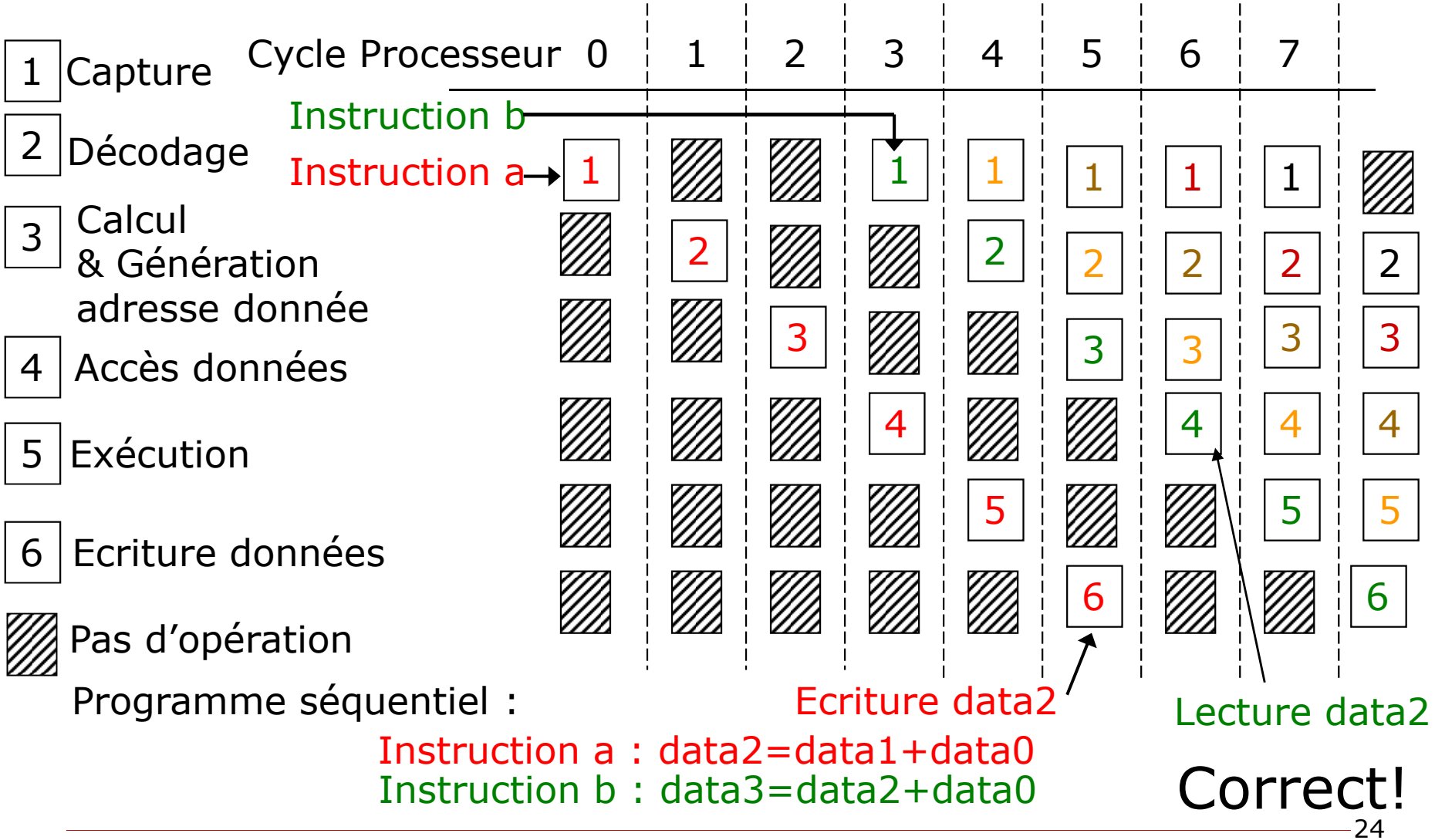
herve.le-provost@cea.fr

Exécution en pipeline des instructions



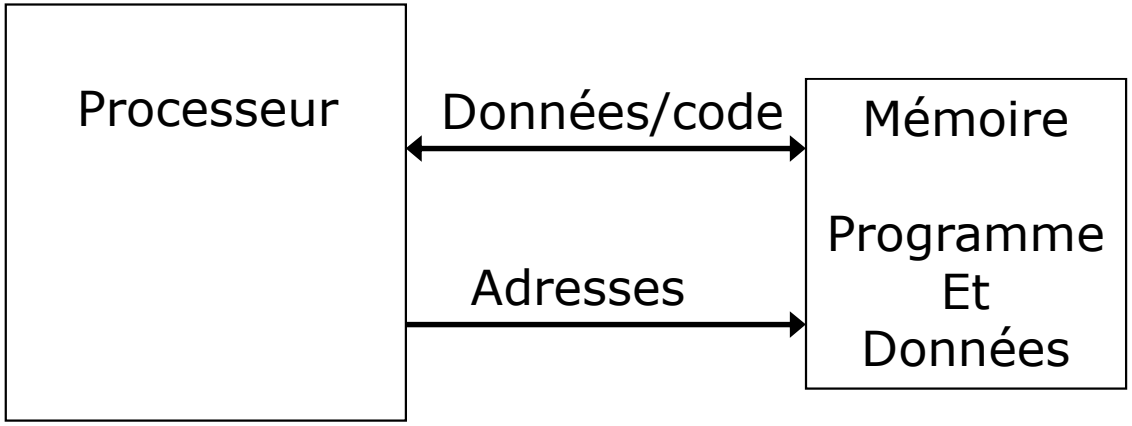
herve.le-provost@cea.fr

Exécution en pipeline des instructions



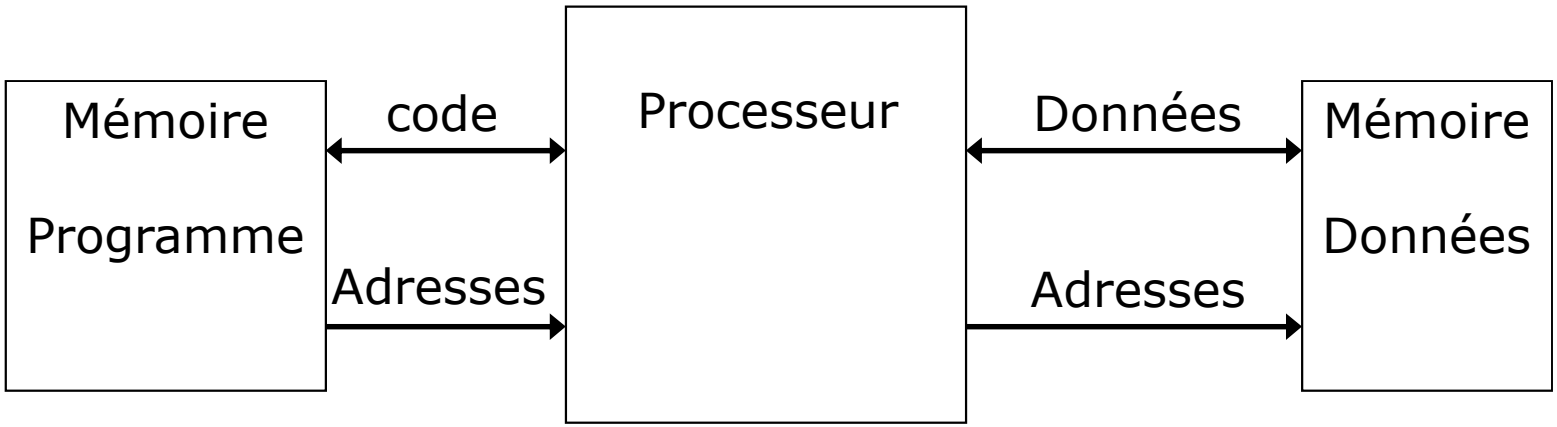
herve.le-provost@cea.fr

Processeur et Mémoires



Architecture von Neumann

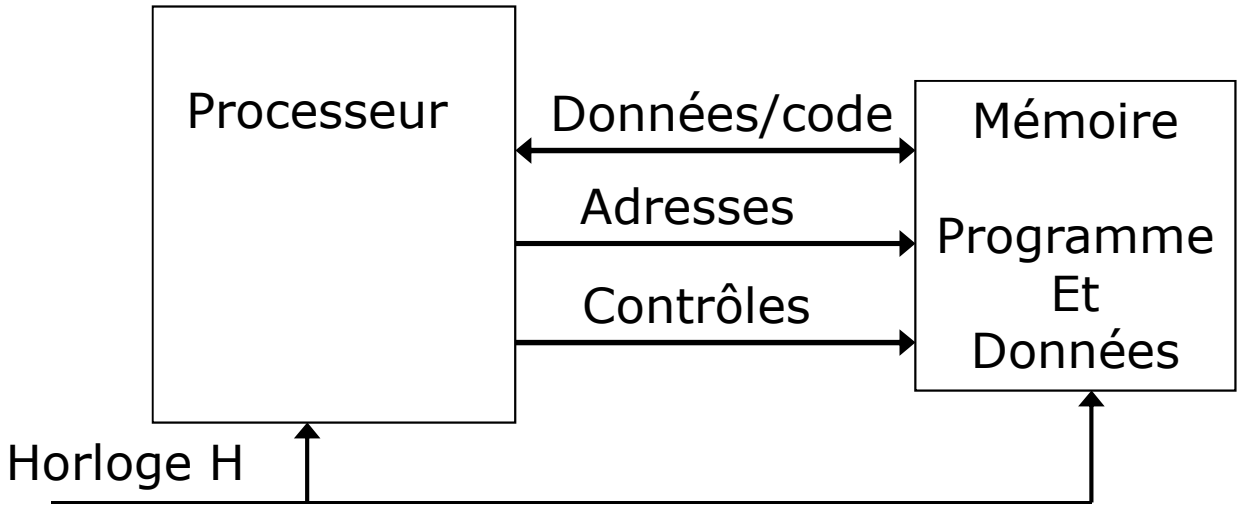
Quelle architecture est directement compatible avec une exécution en pipeline des instructions?



Architecture Harvard

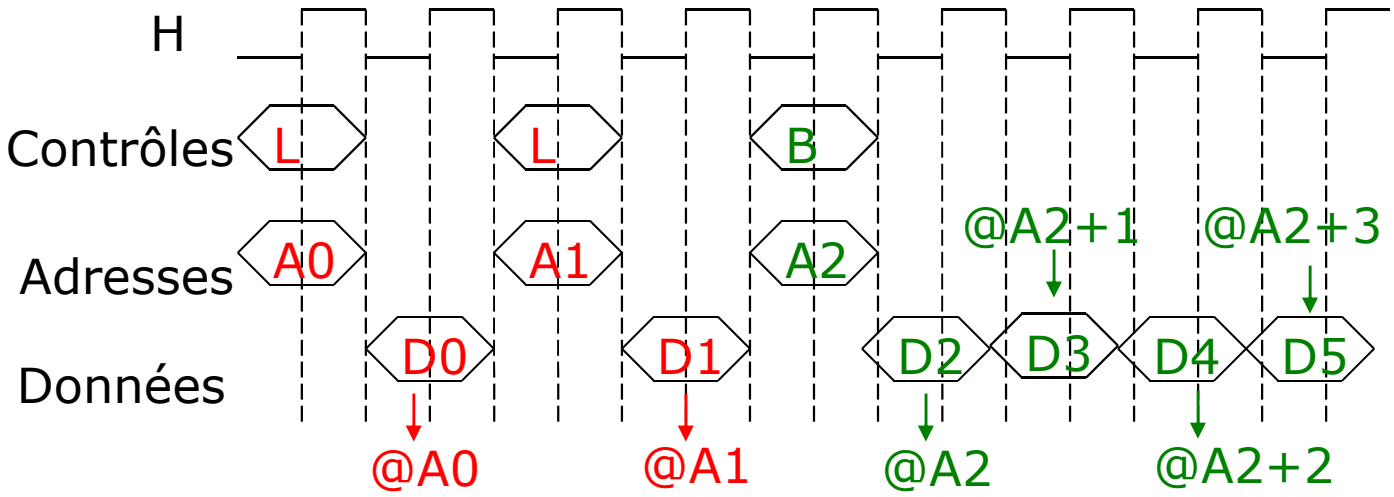
herve.le-provost@cea.fr

Bande passante mémoire



Bande Passante (BP):

Nombre maximum d'octets (données) transférable par seconde entre le processeur et la mémoire.



L : Lecture simple
F_H=100MHz
Donnée 32-bit
BP=200 Mo/s

B: Lecture multiple
FH=100MHz
Donnée 32-bit
4 données/Lecture
BP=320 Mo/s

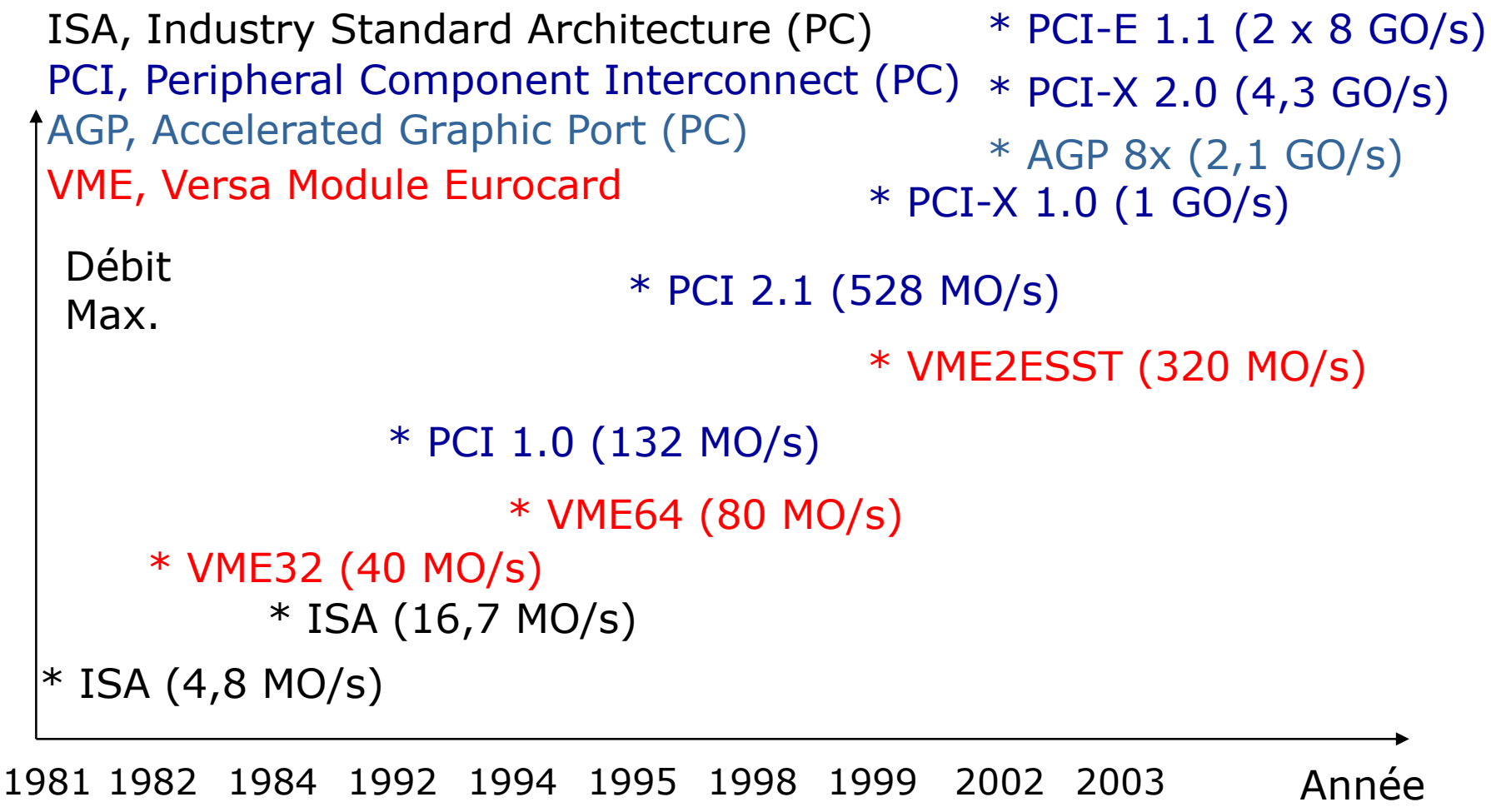
herve.le-provost@cea.fr

Calcul d'une bande passante

- Bus PCI (PC)
  - Version 1.0 32-bit/33 MHz
    - Bande passante?
  - Version 2.1 64-bit/66 MHz
    - Bande passante?
  - PCI-X 2.0 64-bit/533MHz
    - Bande passante?
- Bande passante vs Débit effectif
  - Multiplexage Adresse/Données (PCI)
  - Écriture vs lecture
  - Accès simple vs bouffée
  - Arbitrage sur le bus
  - Partage code/données (von Neumann)
  - Bande passante effective : données utiles transférées

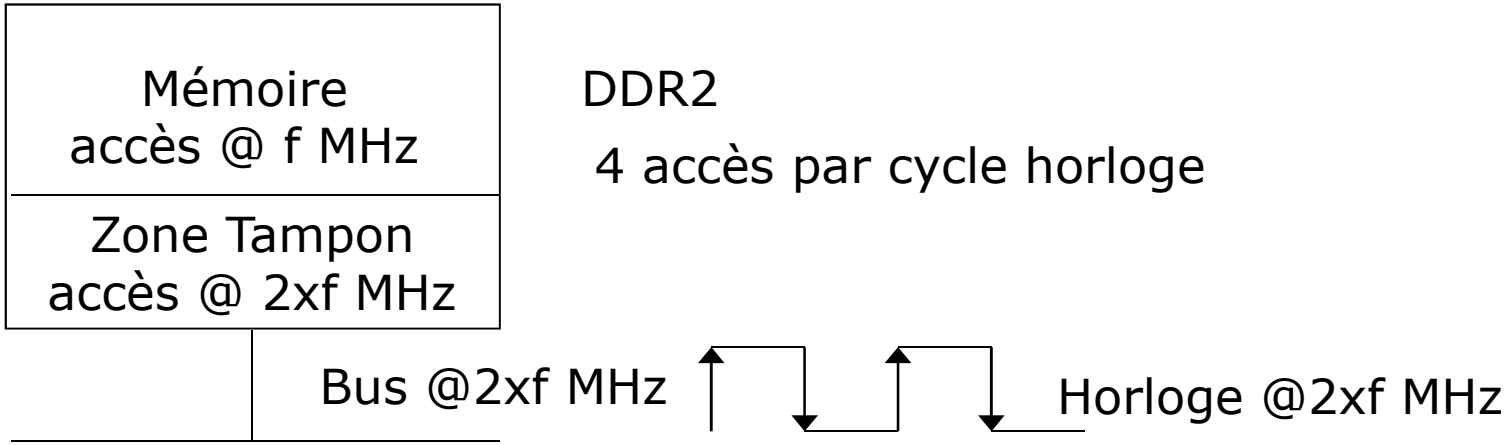
herve.le-provost@cea.fr

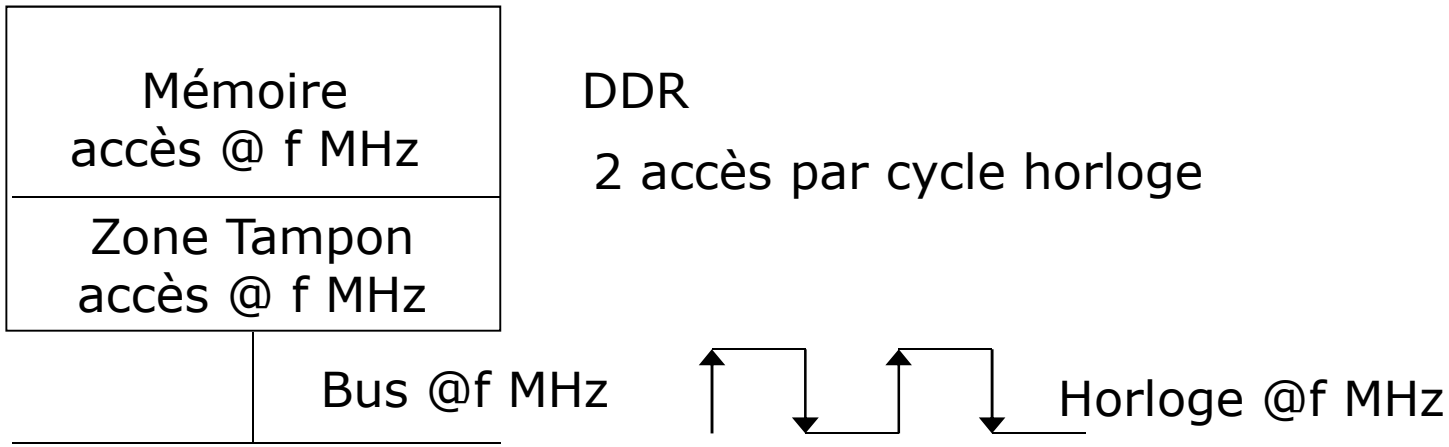
Évolution des Bus d'acquisition de données



herve.le-provost@cea.fr

Mémoire SDRAM DDR2 versus SDRAM DDR

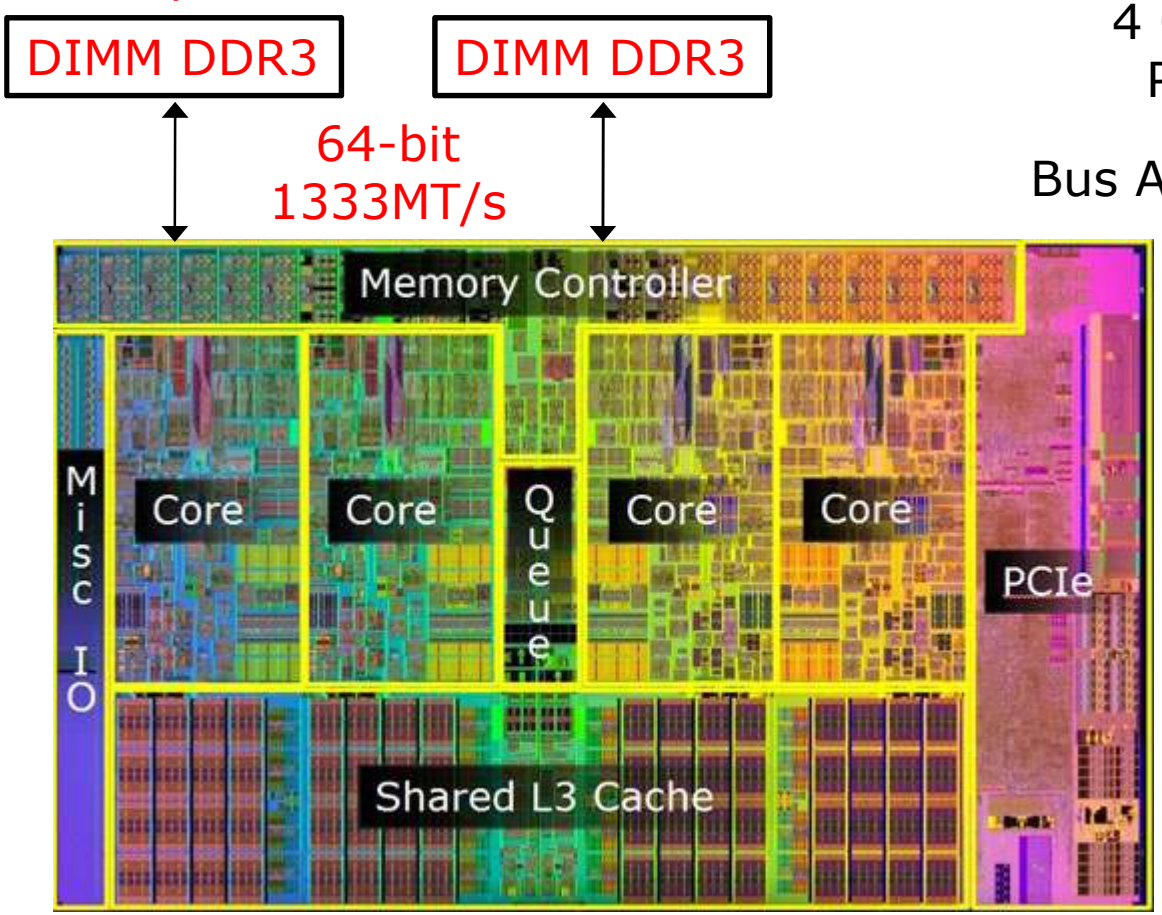




herve.le-provost@cea.fr

Exemple : le processeur Intel Core I5-700

Bande passante Mémoire 21 Go/s



Valeurs illustratives:
4 Cœurs @2,66GHz
Puissance 100W

Bus Adresse 36-bit (64 Go)

PCI Express

2 port x8
(liens full duplex)
ou
1 port x16
(liens full duplex)

Bande passante PCI-E
5 Gb/s par lien
(encodage 8-bit/10-bit)
Total : 16Go/s

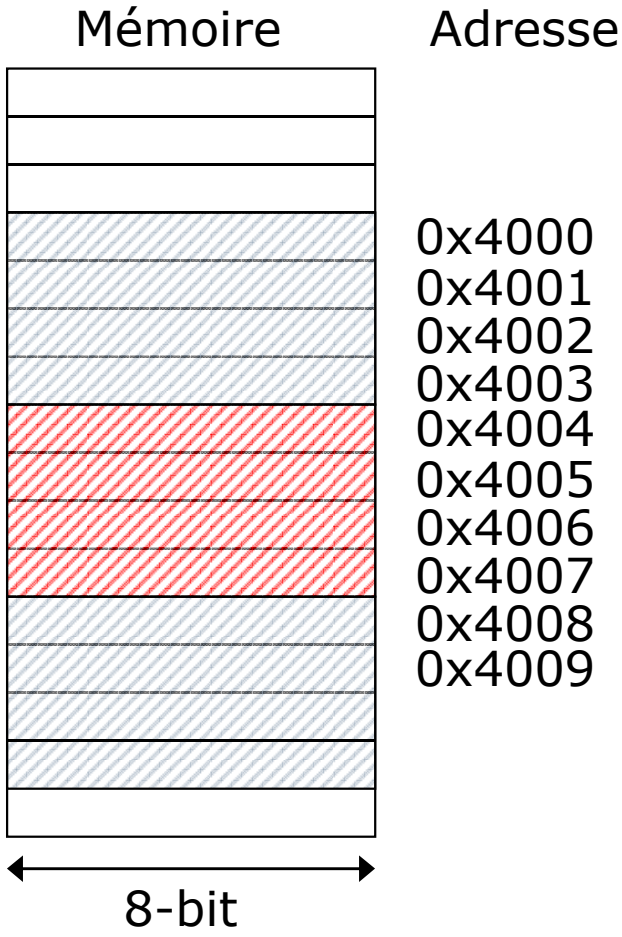
Sources: dessin www.hardware.fr Données techniques : Intel Core i7-800 and i5-700 Desktop processors series, datasheet-Volume 1, September 2009, Intel

herve.le-provost@cea.fr

Organisation mémoire

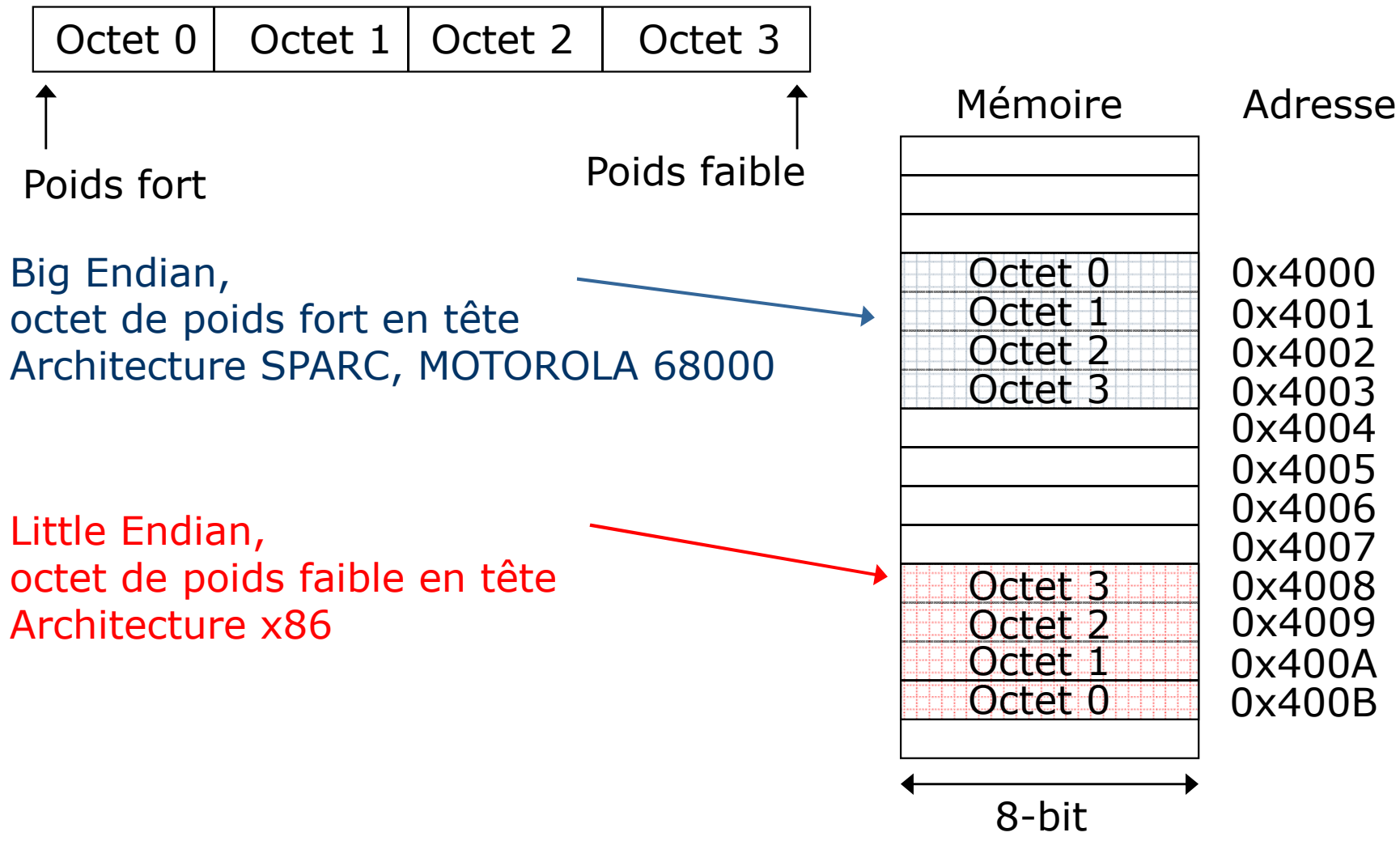
Hypothèses:
Le processeur adresse des octets
Un entier (integer) a pour taille 4 octets

```
void fonctionC()
{
int *ptr;

/*réservation 12 bytes*/
ptr=malloc(3*sizeof(int));

/* initialisation tampon */
for (i=0;i<3;i++)
    *ptr++=0;
…
}
```



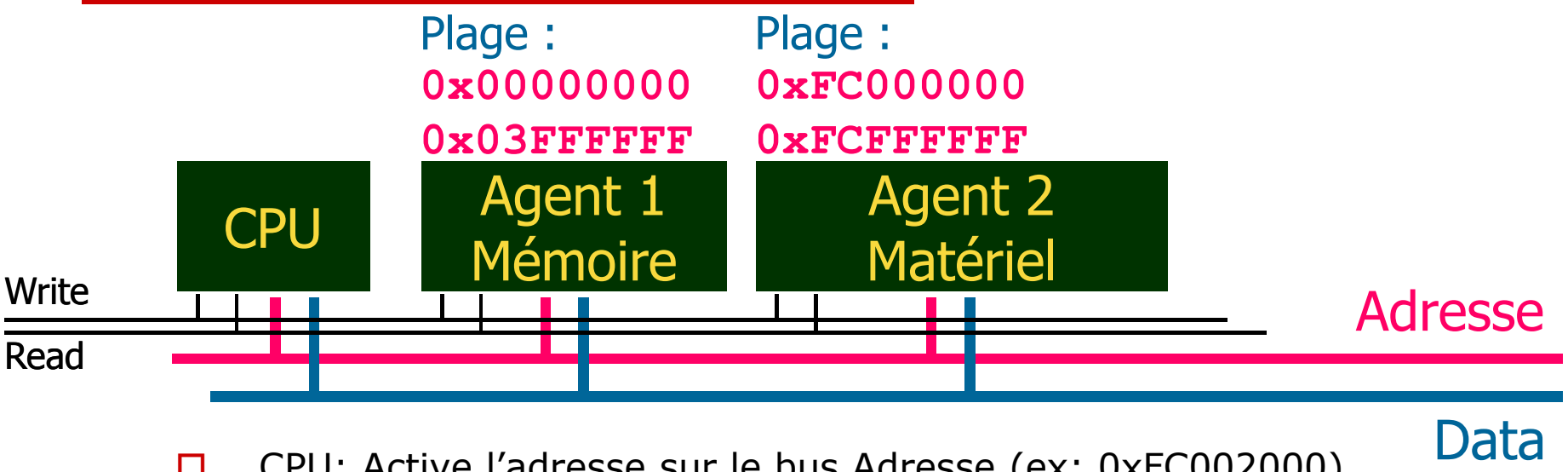
herve.le-provost@cea.fr

Organisation mémoire: Big Endian vs Little Endian



herve.le-provost@cea.fr

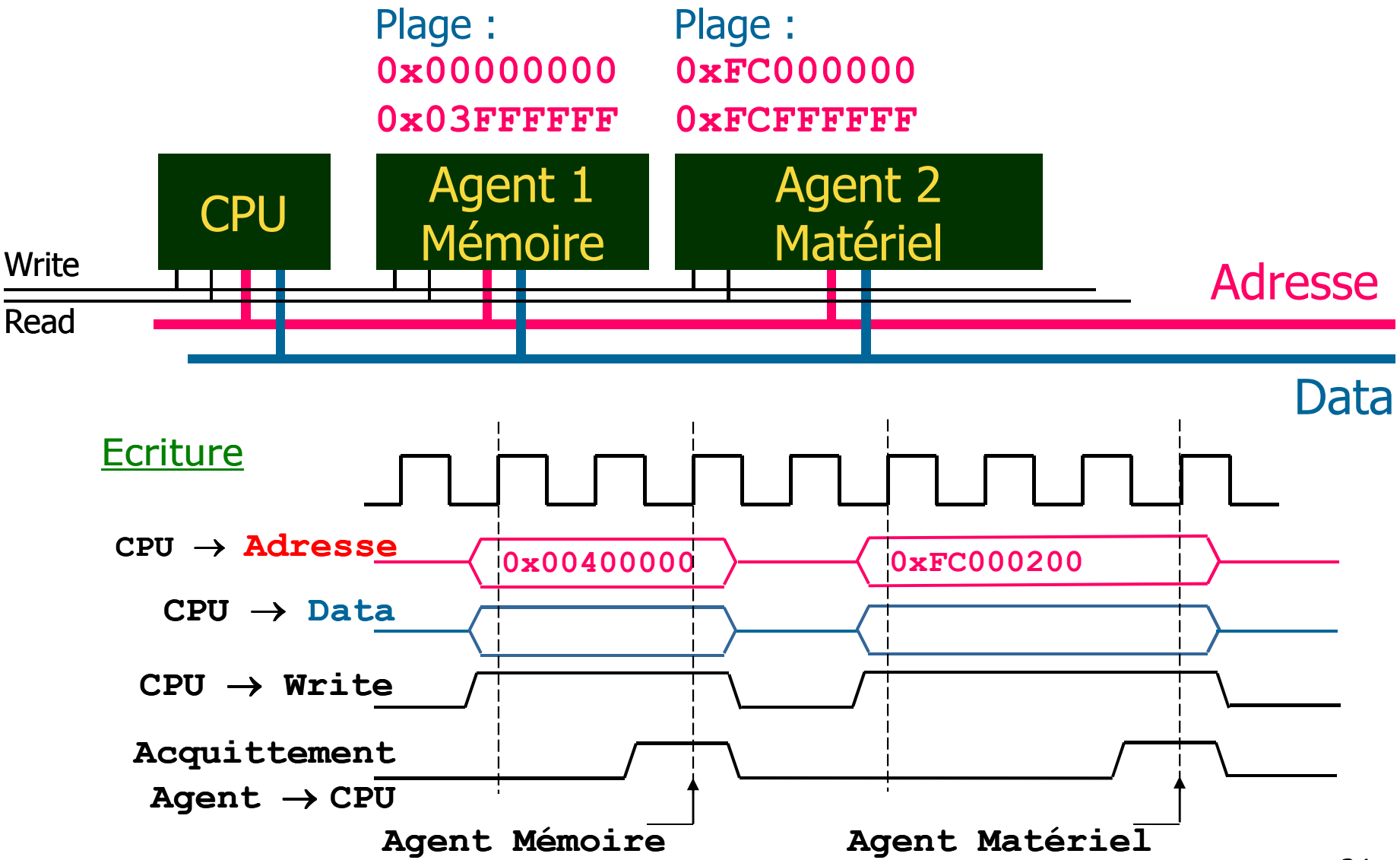
Principe de communication par bus



- CPU: Active l'adresse sur le bus Adresse (ex: 0xFC002000).
- Périphérique : un seul est sensible à cette adresse
- Si écriture :
  - CPU active la donnée sur le bus Data.
  - Périphérique : concerné traite la donnée sur le bus Data.
- Si lecture :
  - Périphérique : active la donnée sur le bus Data.
  - CPU : traite (lis) la donnée sur le bus Data.
- Modes de lecture/écriture spéciaux (rafale synchrone, etc.)
- Contrôleur mémoire : séparé ou intégré au CPU

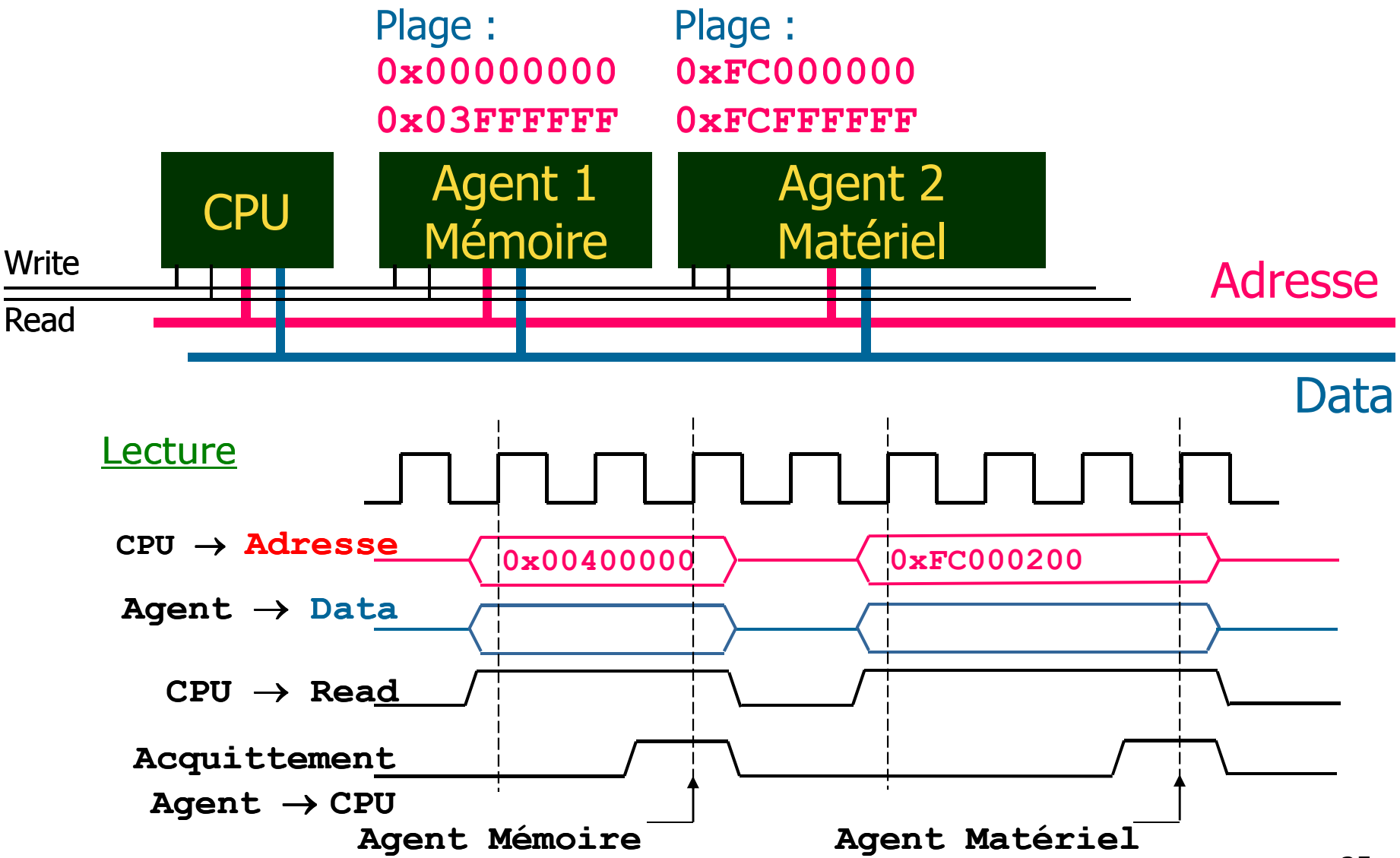
herve.le-provost@cea.fr

Principe de communication par bus



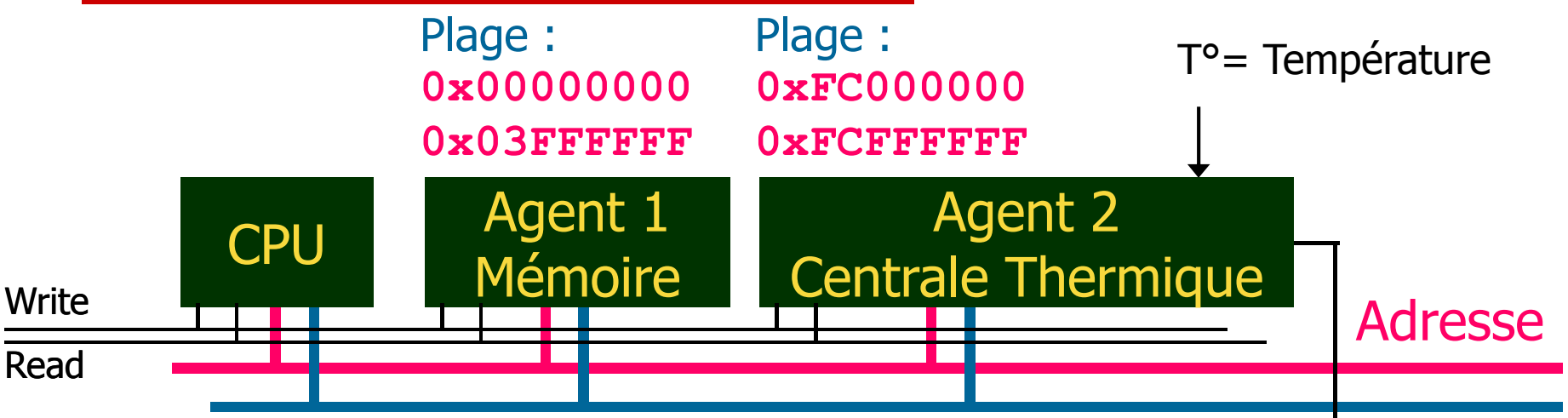
herve.le-provost@cea.fr

Principe de communication par bus



herve.le-provost@cea.fr

Principe de communication par bus



Adresse	Registre 32-bit	#bit	Mode Accès
0xFC000200	Alarme	0	Lecture/RAZ (Ecriture 1)
0xFC000204	T° courante(°C)	7 à 0	Lecture
0xFC000208	T° maximale (°C)	7 à 0	Lecture

Ecrire le code C qui sur alarme affiche
la température courante et maximale et efface l'alarme

herve.le-provost@cea.fr

Principe de communication par bus

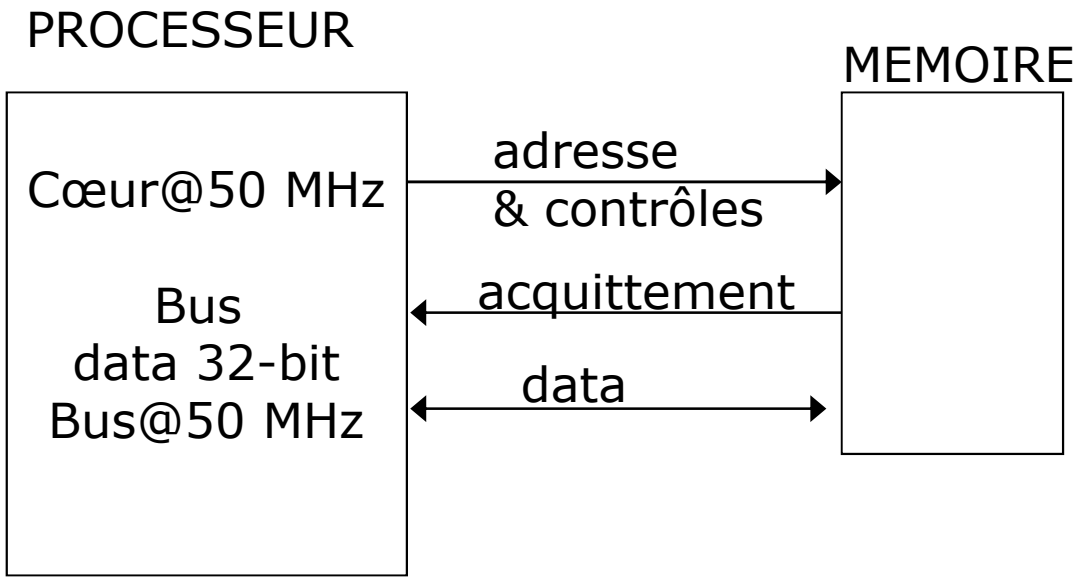
```
void alarm() {
volatile unsigned int *ptr1, *ptr2, *ptr3;
int alarme, T0, T1;
ptr1=(unsigned int*)0xFC000200;//alarme
ptr2=(unsigned int*)0xFC000204;//T° courante
ptr3=(unsigned int*)0xFC000208;//T° maximale

while (1) {
   alarme=*ptr1&1;
   if (alarme!=0) {
      T0=(*ptr2)&255;//lecture T° courante
      T1=(*ptr3)&255;//lecture T° maximale
      printf («Temp. courante %d maximale %d\n», T0, T1);
      *ptr1=1;//RAZ alarme
    }
}
}
```

Quel est l'impact de ce code sur les performances du CPU?

herve.le-provost@cea.fr

Exercices de Synthèse

Exercice 1



- 1 cycle d'adresse puis attente acquittement
- Mode rafale non supporté
- Acquittement sur le second cycle en écriture
- Acquittement sur le quatrième cycles en lecture

Calcul de la bande passante (Mo/s) effective en écriture

Calcul de la bande passante (Mo/s) effective en lecture

herve.le-provost@cea.fr

Exercices de Synthèse

Correction Exercice 1

Calcul de la bande passante effective en écriture

3 cycles@50MHz (60ns) pour transférer 4 octets
4/(60ns) = 66,7 Méga octets transférées en 1s

$BP_{Écriture}$ = 66,7 Mo/s

Calcul de la bande passante effective en lecture

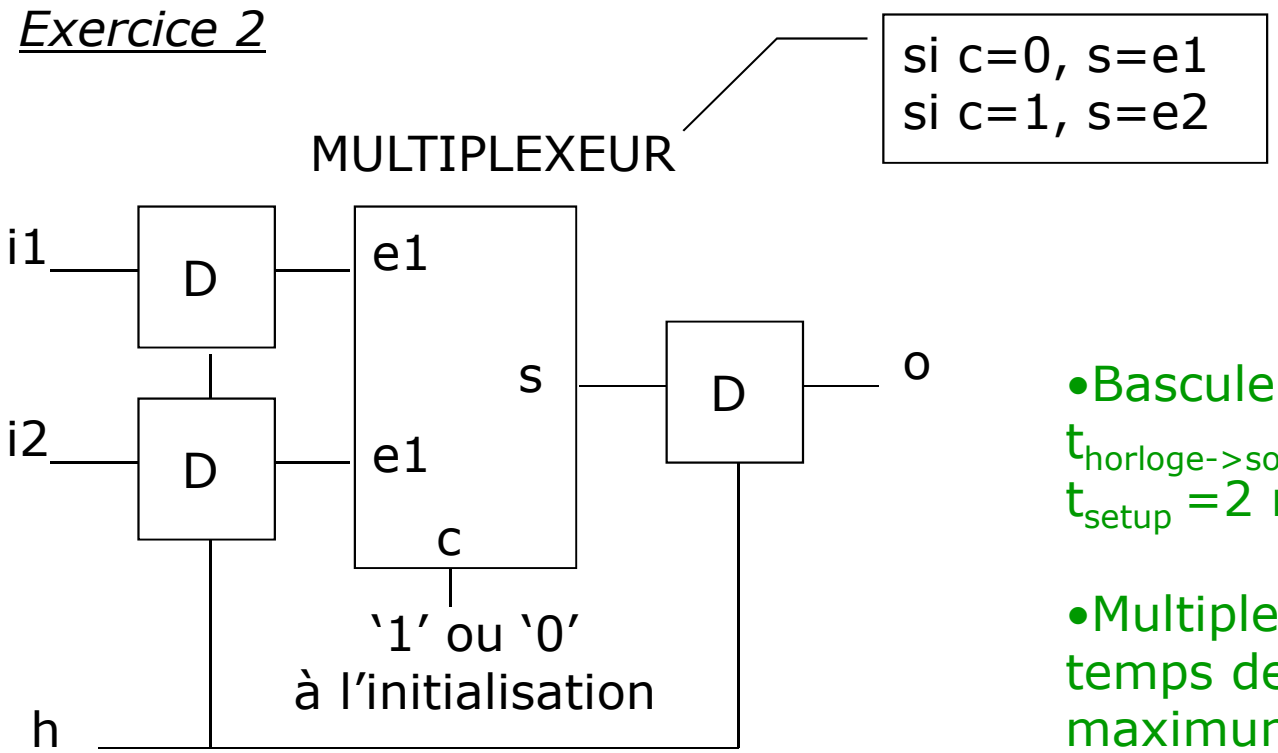
5 cycles@50MHz (100ns) pour transférer 4 octets
4/(100ns) = 40 Méga octets transférées en 1s

$BP_{Lecture}$ = 40 Mo/s

herve.le-provost@cea.fr

Exercices de Synthèse

Exercice 2



- Bascule D:
  $t_{horloge->sortie}$ =3 ns maximum
  t_{setup} =2 ns minimum

- Multiplexeur:
  temps de propagation 23 ns maximum

Calcul de la fréquence maximale de fonctionnement

herve.le-provost@cea.fr

Exercices de Synthèse

Correction Exercice 2

Calcul de la fréquence maximale de fonctionnement

$F_{max} = 1/(t_{horloge->sortie} + t_{setup} + t_{prop.\ Mux.})$

$F_{max} = 1/(3\ ns + 2\ ns + 23\ ns)$

$F_{max} = 35{,}7\ MHz$

herve.le-provost@cea.fr

Les interruptions

- L'autre moyen de communication entre un périphérique et le CPU est l'interruption matérielle. C'est la seule communication qui soit ***initiée à l'initiative du périphérique (esclave)***.

- Son fonctionnement exact dépend de l'architecture matérielle de la carte et des liaisons entre le périphérique et les broches d'interruption du CPU.

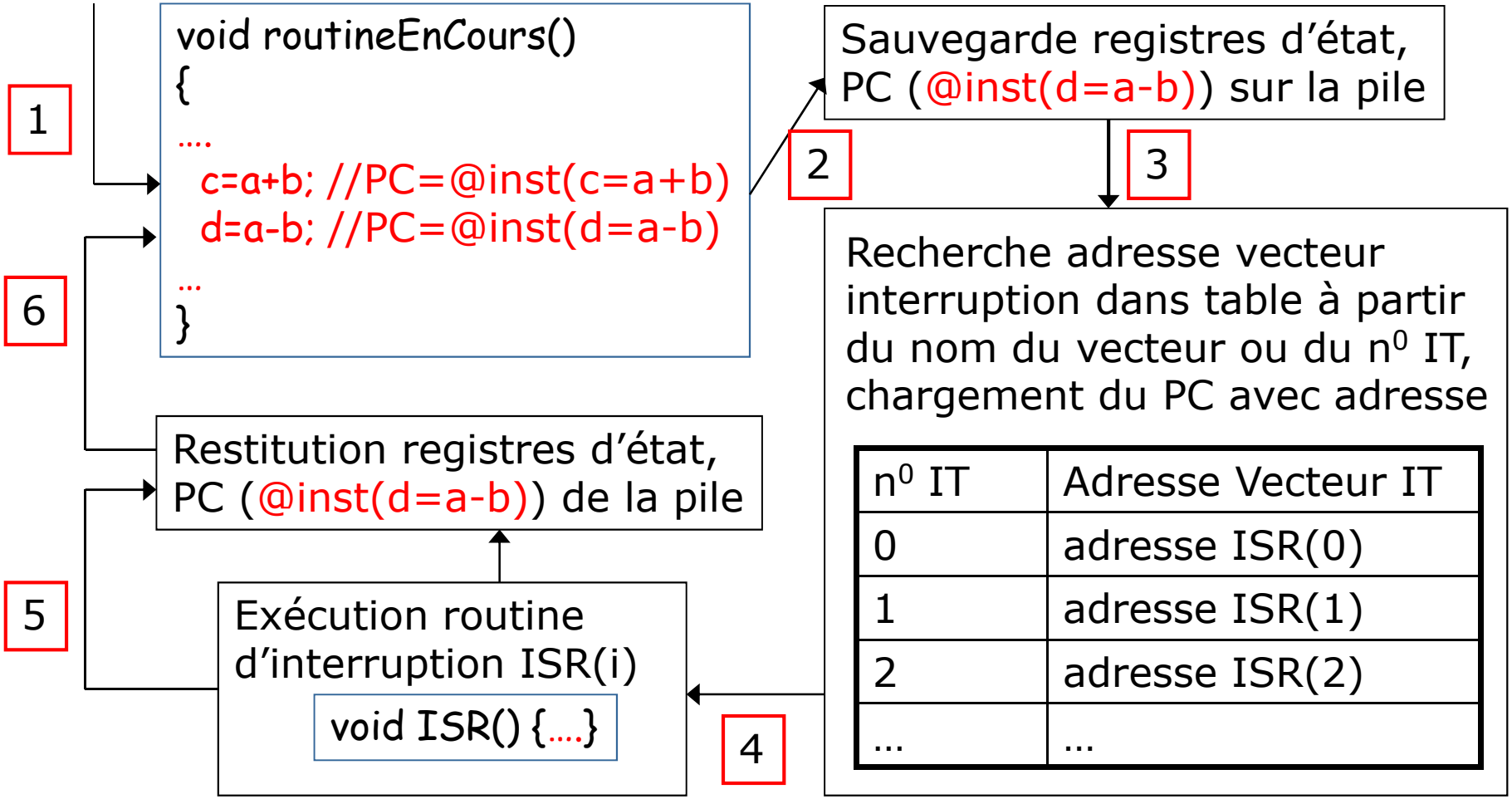
- Le CPU associe à une broche une fonction d'interruption. Dans cette fonction, l'interruption est traitée. Si le CPU est animé par un OS :
  - Tous les appels susceptibles d'être bloquants sont interdits au sein de la fonction d'interruption.
  - Le temps d'exécution de l'interruption doit être minimisé. Tout traitement lourd doit être effectué par une tâche standard en attente de libération d'un sémaphore. Ce sémaphore est libéré par la fonction d'interruption

herve.le-provost@cea.fr

Les interruptions

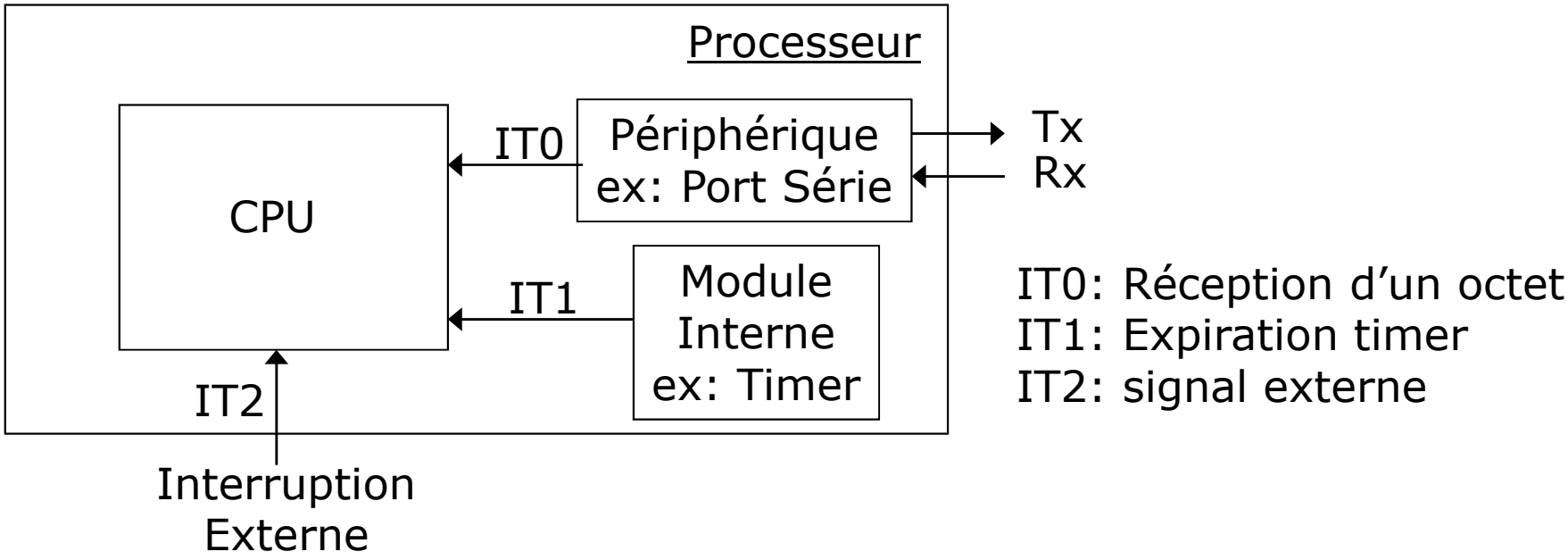
Interruption (IT) matérielle/logicielle n^0 i

ISR : Interrupt Service Routine
PC : Program Counter



herve.le-provost@cea.fr

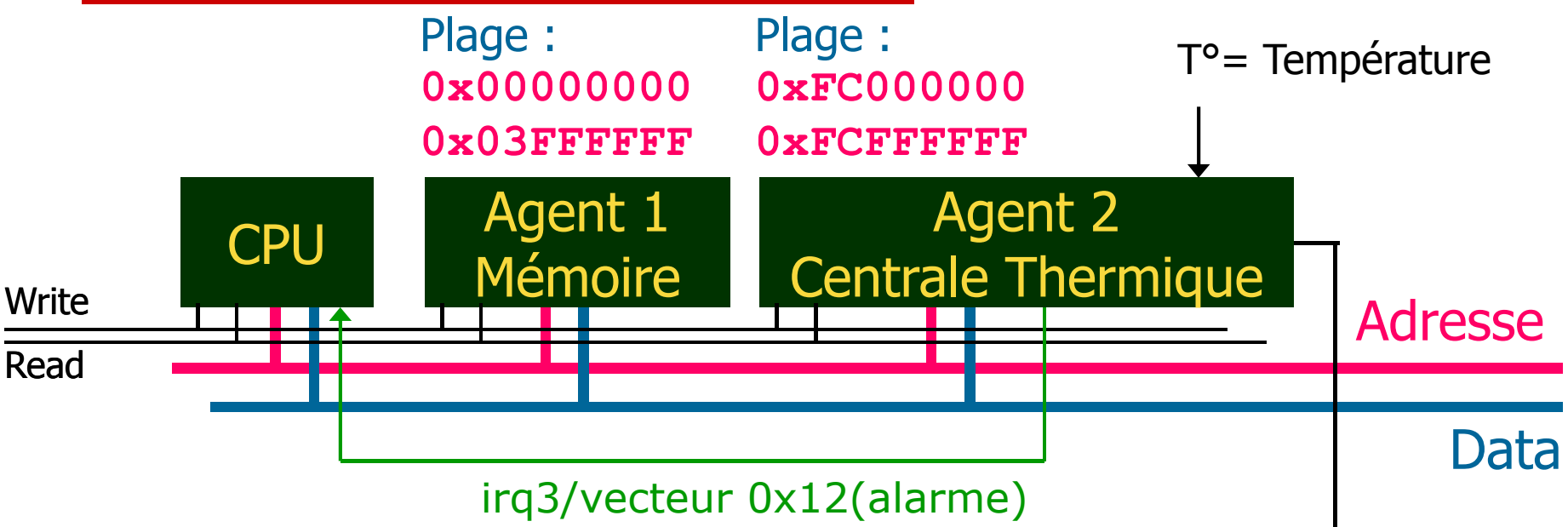
Les interruptions



- Interruption externe asynchrone, sélection front montant, état
- Trois principaux registres d'interruption:
  - Registre d'autorisation individuelle
  - Registre de présence individuelle
  - Registre de remise à zéro individuelle
- Priorités entre les interruptions définies par le processeur cible
- Sur interruption, sauvegarde automatique du PC et des registres d'état sur la pile

herve.le-provost@cea.fr

Les interruptions



Adresse	Registre 32-bit	#bit	Mode Accès
0xFC000204	T° courante(°C)	7 à 0	Lecture
0xFC000208	T° maximale (°C)	7 à 0	Lecture

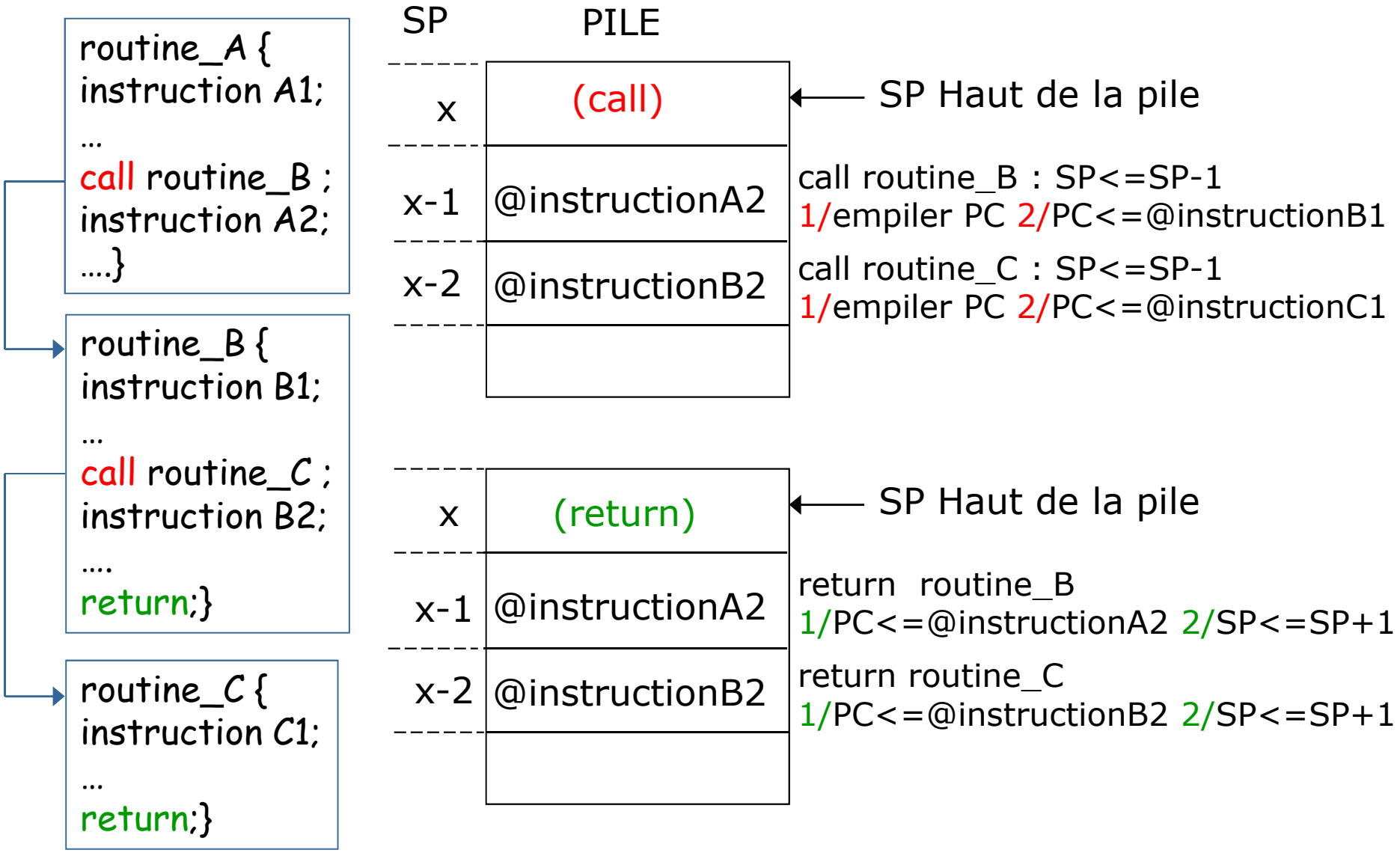
herve.le-provost@cea.fr

Les interruptions

```
//Exemple gestion interruptions-Système exploitation RTEMS
void initSystem() {
rtems_status_code status;
rtems_isr_entry old_handle;
//…
status=rtems_interrupt_catch(alarmCatch, 0x12,&old_handle);
//…
}
rtems_isr alarmCatch(rtems_vector_number vector) {
volatile unsigned int  *ptr2, ptr3;
int T0, T1;

ptr2=(unsigned int*)0xFC000204;//T° courante
ptr3=(unsigned int*)0xFC000208;//T° maximale
T0=(*ptr2)&255;//lecture T° courante
T1=(*ptr3)&255;//lecture T° maximale
 printk («Temp. courante %d maximale %d\n», T0, T1);
}
```


herve.le-provost@cea.fr

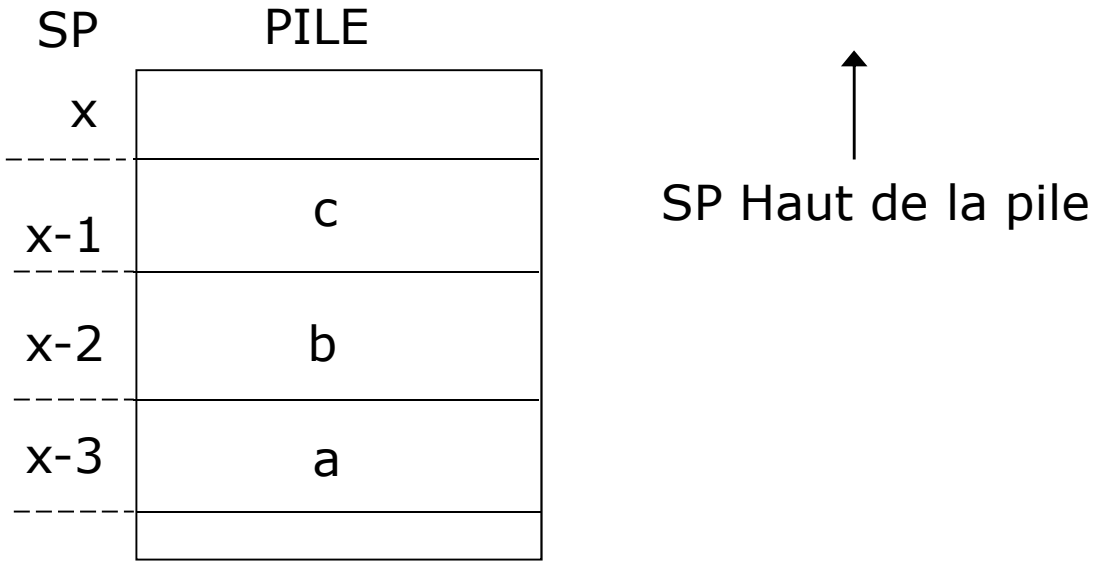
La Pile: Principe



herve.le-provost@cea.fr

La Pile : variables automatiques

```
int main()
{
int a=3,b=5,c;

c=a+b;

return 0;
}
```



1. Décrémentation SP de 3/Réservation mémoire pour ‘a’,’b’,’c’
2. Stockage valeurs ‘a’,’b’ sur la pile
3. Accès aux variables depuis la pile pour exécution du calcul
4. Stockage résultat ‘c’ sur la pile
5. Incrémentation SP de 3/Libération mémoire

herve.le-provost@cea.fr

La Pile : variables automatiques

```
;codage assembleur DSP TMS320C55
;int main()
;{
01418F                  main:
01418F 4efd             AADD #-3,SP
;int a=3,b=5,c;
014191 e60003           MOV #3,*SP(#00h)
014194 e60205           MOV #5,*SP(#01h)
;c=a+b;
014197 a902             MOV *SP(#01h),AR1
014199 d60099           ADD *SP(#00h),AR1,AR1
01419C c904             MOV AR1,*SP(#02h)
;return 0;
01419E 3c04             MOV #0,T0
;}
0141A0 4e03             AADD #3,SP
0141A2 4804             RET
```

Adresse instruction

Code machine

Décrémentation pointeur de pile

Stockage valeurs a et b sur la pile

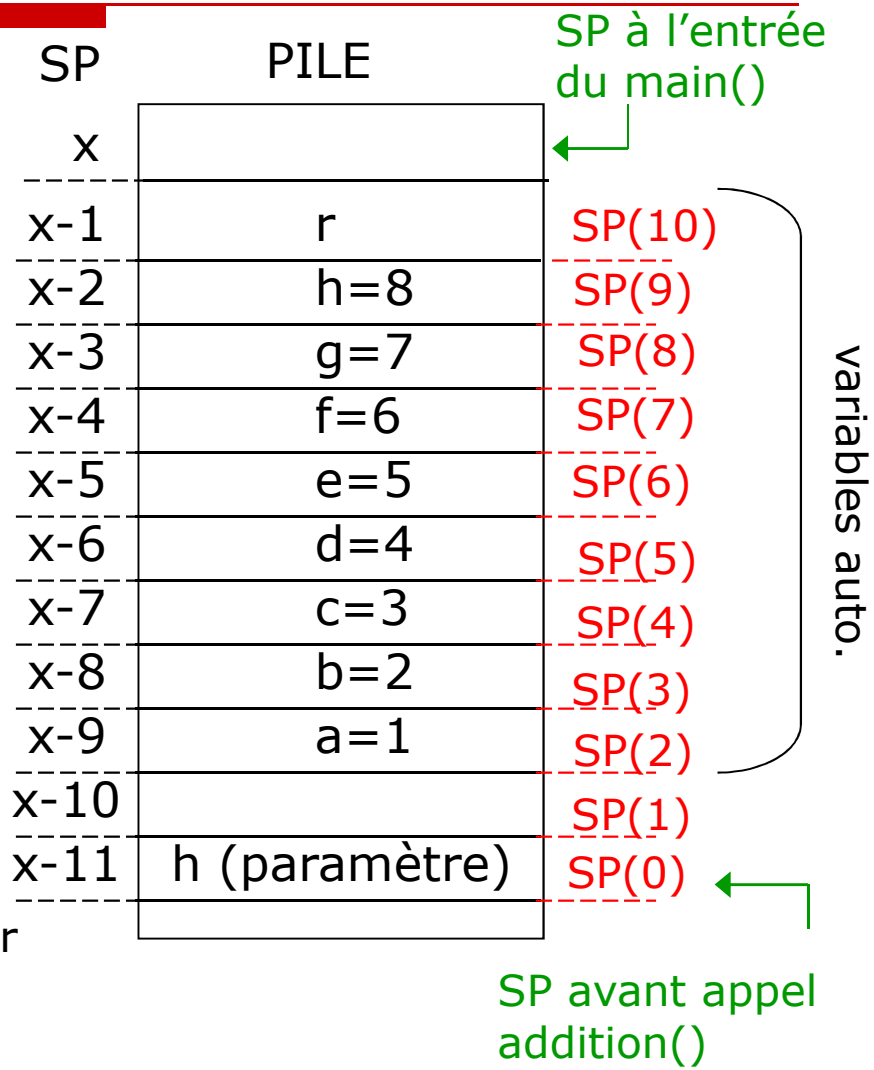
Calcul

Stockage résultat Sur la pile

herve.le-provost@cea.fr

La Pile : passage des paramètres

```
int main()
{
int a=1,b=2,c=3,d=4,e=5,f=6,g=7,h=8;
int r;
        r=addition(a,b,c,d,e,f,g,h);
        return 0;
}
int addition(int a,int b,int c,int d,int e,
int f,int g,int h)
{
int r;
        r=a+b+c+d+e+f+g+h;
        return r;
}
```



- Paramètres a, b, c, d, e, f, g passés par registre, paramètre h passé sur la pile
- Retour r passé par registre
- Dépendant du couple compilateur/cible, du niveau d'optimisation du compilateur

herve.le-provost@cea.fr

La Pile : passage des paramètres

```
01418F 4ef5          AADD #-11,SP
014191 e60401        MOV #1,*SP(#02h)
014194 e60602        MOV #2,*SP(#03h)
014197 e60803        MOV #3,*SP(#04h)
01419A e60a04        MOV #4,*SP(#05h)
01419D e60c05        MOV #5,*SP(#06h)
0141A0 e60e06        MOV #6,*SP(#07h)
0141A3 e61007        MOV #7,*SP(#08h)
0141A6 e61208        MOV #8,*SP(#09h)
0141A9 a912          MOV *SP(#09h),AR1
0141AB c900          MOV AR1,*SP(#00h)
0141AD a404          MOV *SP(#02h),T0
0141AF a506          MOV *SP(#03h),T1
0141B1 a808          MOV *SP(#04h),AR0
0141B3 aa0c          MOV *SP(#06h),AR2
0141B5 ab0e          MOV *SP(#07h),AR3
0141B7 ac10          MOV *SP(#08h),AR4
0141B9 a90a          MOV *SP(#05h),AR1
0141BB 080008        CALL addition
0141BE c414          MOV T0,*SP(#0ah)
0141C0 3c04          MOV #0,T0
0141C2 4e0b          AADD #11,SP
0141C4 4804          RET
```

Décrémentation pointeur de pile

Stockage valeurs de 8 variables sur la pile

Passage de 1 paramètre par la pile

Passage de 7 paramètres par registre

Appel addition

Stockage résultat sur la pile

herve.le-provost@cea.fr

La Pile : passage des paramètres

```
0141C6           addition:
0141C6 4ef7              AADD #-9,SP
0141C8 cc0c              MOV AR4,*SP(#06h)
0141CA cb0a              MOV AR3,*SP(#05h)
0141CC ca08              MOV AR2,*SP(#04h)
0141CE c906              MOV AR1,*SP(#03h)
0141D0 c804              MOV AR0,*SP(#02h)
0141D2 c502              MOV T1,*SP(#01h)
0141D4 c400              MOV T0,*SP(#00h)
0141D6 a902              MOV *SP(#01h),AR1
0141D8 d60099            ADD *SP(#00h),AR1,AR1
0141DB d60499            ADD *SP(#02h),AR1,AR1
0141DE d60699            ADD *SP(#03h),AR1,AR1
0141E1 d60899            ADD *SP(#04h),AR1,AR1
0141E4 d60a99            ADD *SP(#05h),AR1,AR1
0141E7 d60c99            ADD *SP(#06h),AR1,AR1
0141EA d61499            ADD *SP(#0ah),AR1,AR1
0141ED c90e              MOV AR1,*SP(#07h)
0141EF 2294              MOV AR1,T0
0141F1 4e09              AADD #9,SP
0141F3 4804              RET
```

Décrémentation pointeur de pile

Transfert paramètres des registres vers la pile

Addition paramètres passés par registre

Addition paramètres passé par la pile

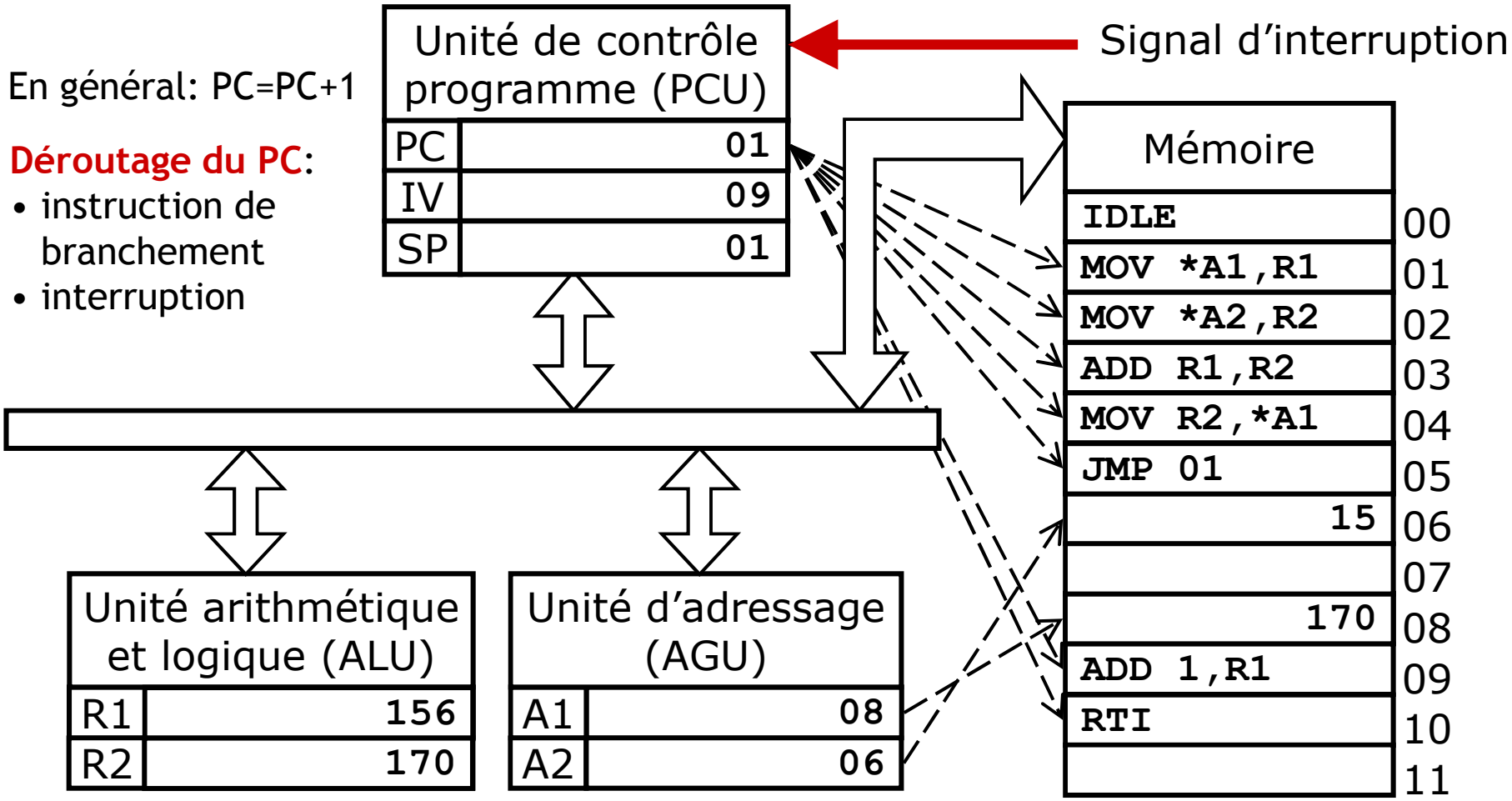
Stockage résultat sur la pile

Retour résultat par registre

herve.le-provost@cea.fr

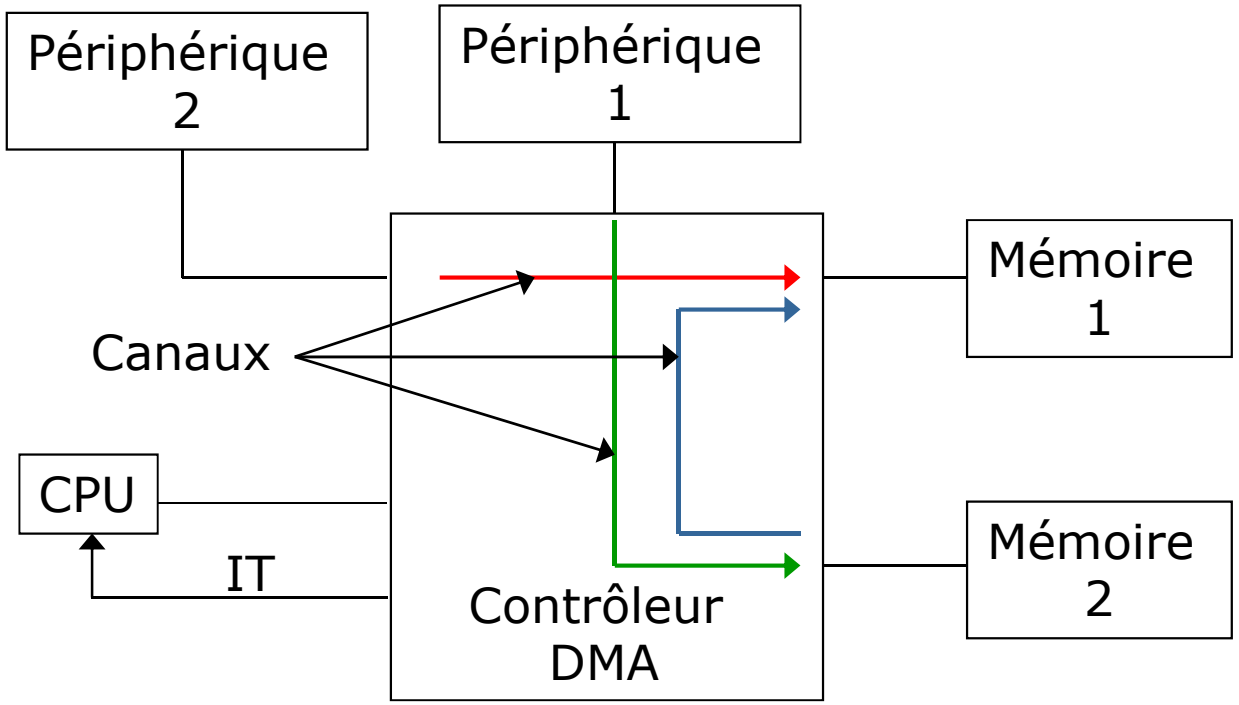
Fonctionnement du processeur

Animation PowerPoint



herve.le-provost@cea.fr

DMA (Direct Memory Access): principe



CPU → 1 Configuration :
- Port-adresse source
- Port-adresse destination
- Nombre d'éléments à transférer

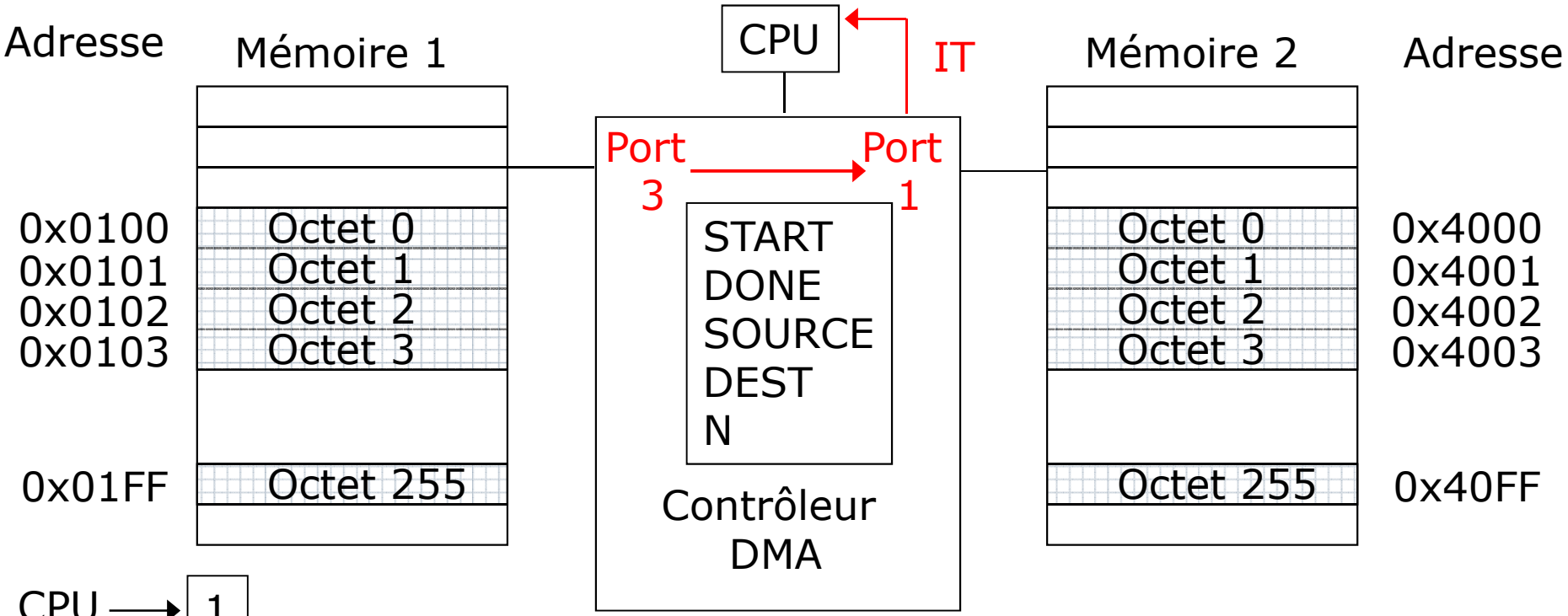
CPU → 2 Démarrage transfert

DMA → 3 Fin du transfert
Interruption CPU optionnelle

DMA → 4 Redémarrage automatique
Ou attente d'un nouveau démarrage ordonné par le CPU

herve.le-provost@cea.fr

DMA: exemple



CPU ⟶ 1

- Source Port=3 @=0x0100
- Dest. Port=1 @=0x4000
- N=256

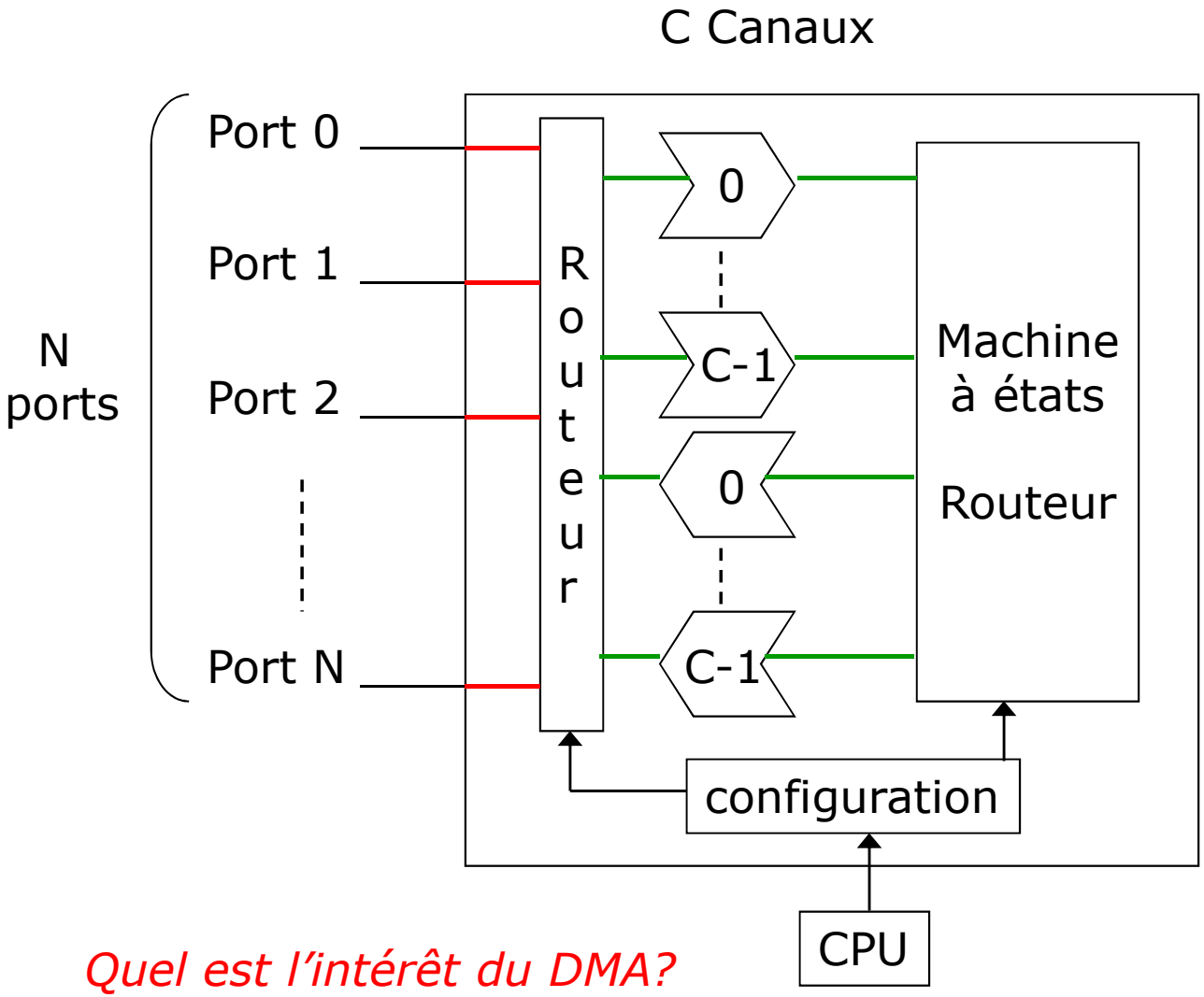
CPU ⟶ 2 START=1

DMA ⟶ 3 DONE=1; IT=1

DMA ⟶ 4 Attente Nouveau transfert

herve.le-provost@cea.fr

DMA: structure interne



- Limitation du nombre de canaux
- Distinction entre canaux et ports
- Port connectable sur lui-même

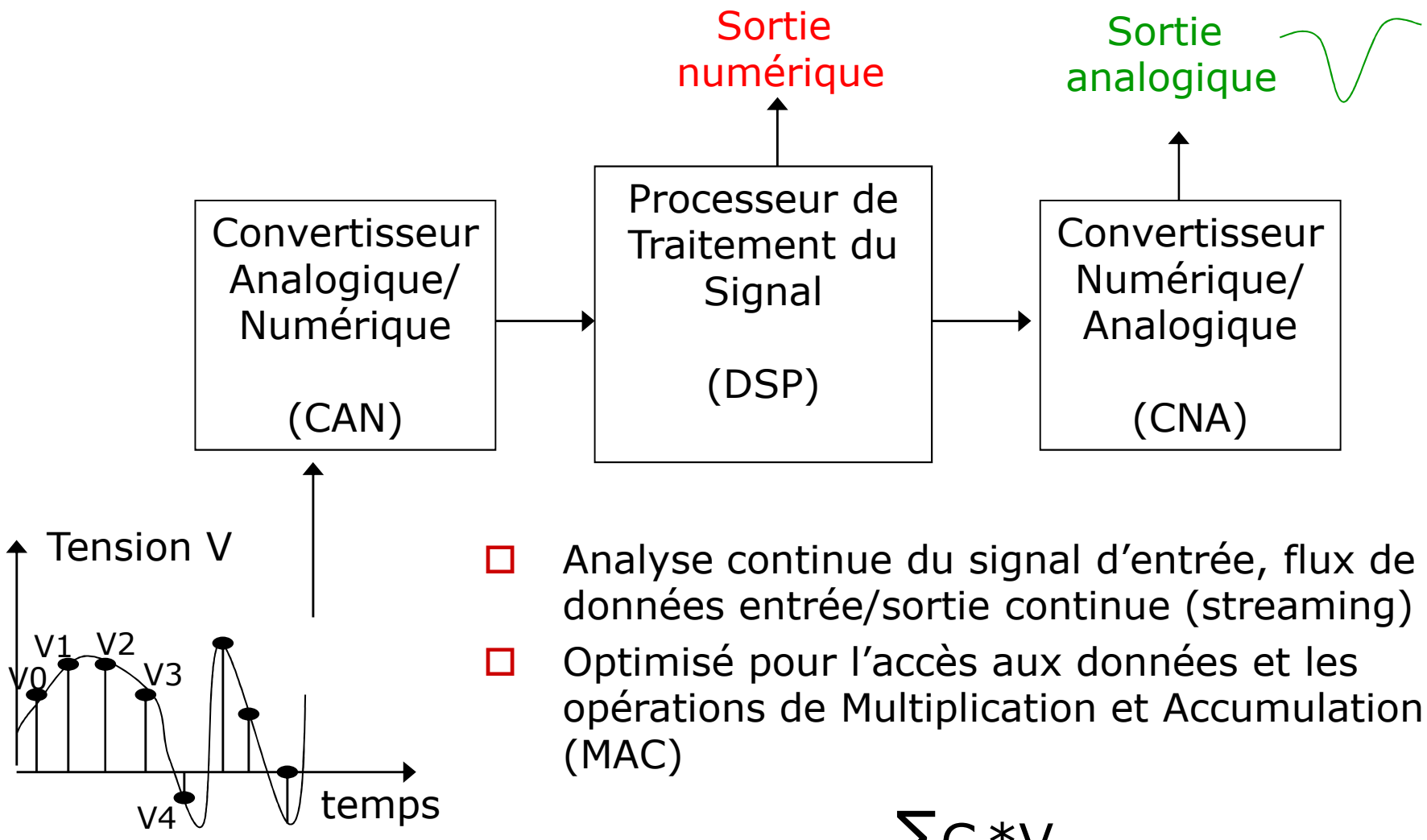
Quel est l'intérêt du DMA?

herve.le-provost@cea.fr

- Maîtrise fine du système:
  - Matériel
    - La bande passante et la gestion de la mémoire
    - La pile
    - Le pipeline des données et des instructions
    - La fréquence de fonctionnement
  - Logiciel
    - La chaine de compilation et ses niveaux d'optimisation et interactions avec le matériel.
    - Le langage assembleur pour les portions critiques
- Les éléments de performance
  - Les interruptions
  - Le DMA
  - *La mémoire cache* (seconde partie du cours)

herve.le-provost@cea.fr

Applications DSP

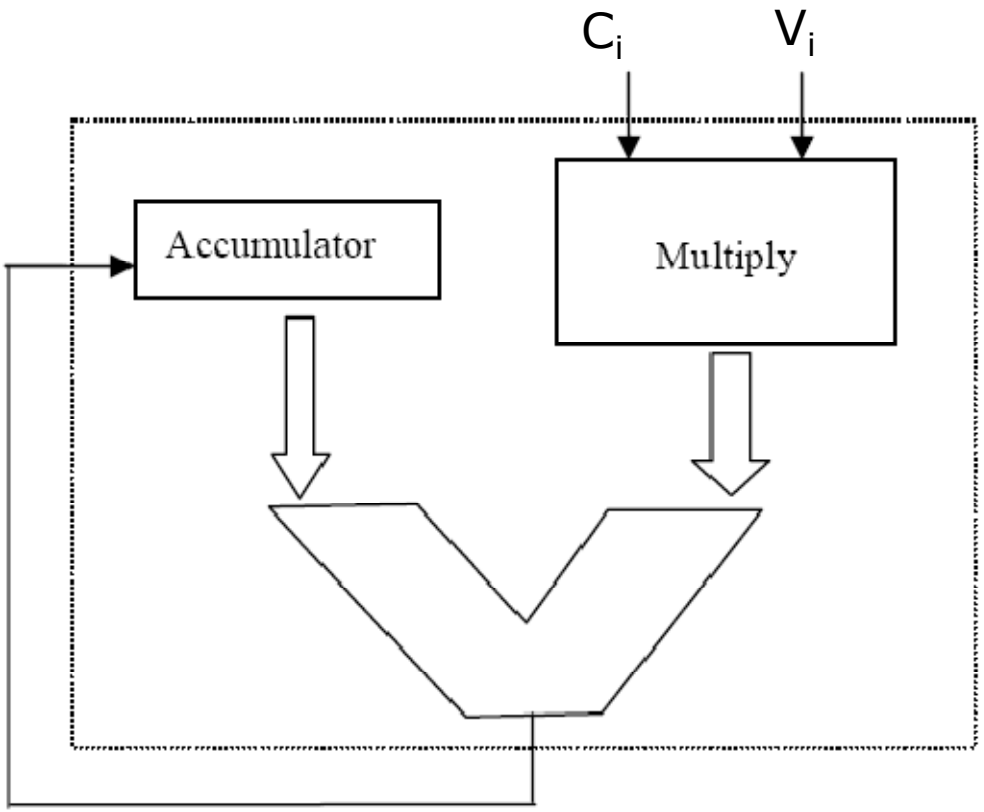


- Analyse continue du signal d'entrée, flux de données entrée/sortie continue (streaming)
- Optimisé pour l'accès aux données et les opérations de Multiplication et Accumulation (MAC)

$$\sum C_i * V_i$$

herve.le-provost@cea.fr

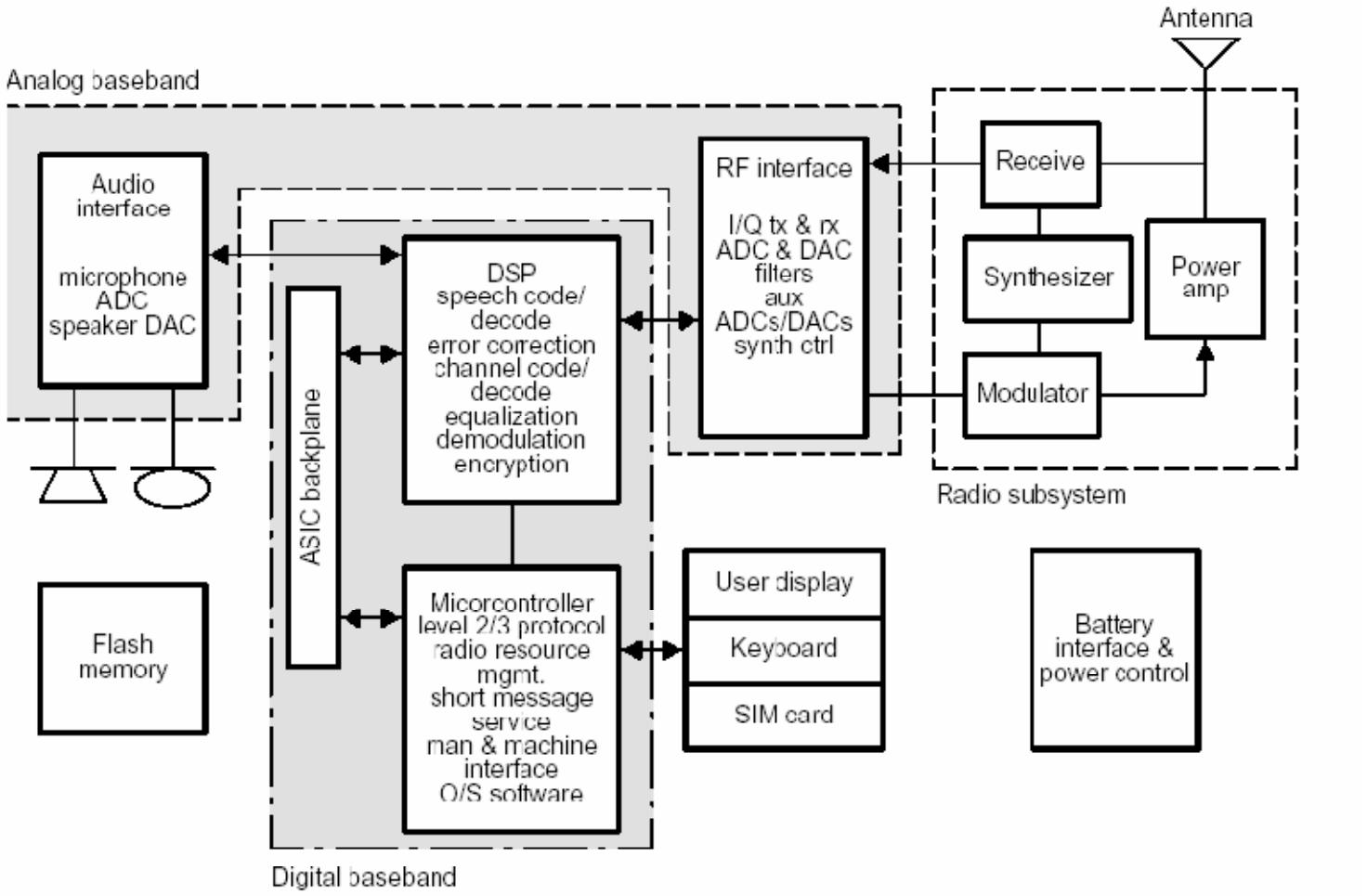
Applications DSP



MAC Câblé dans le silicium, 1 opération à chaque cycle processeur

herve.le-provost@cea.fr

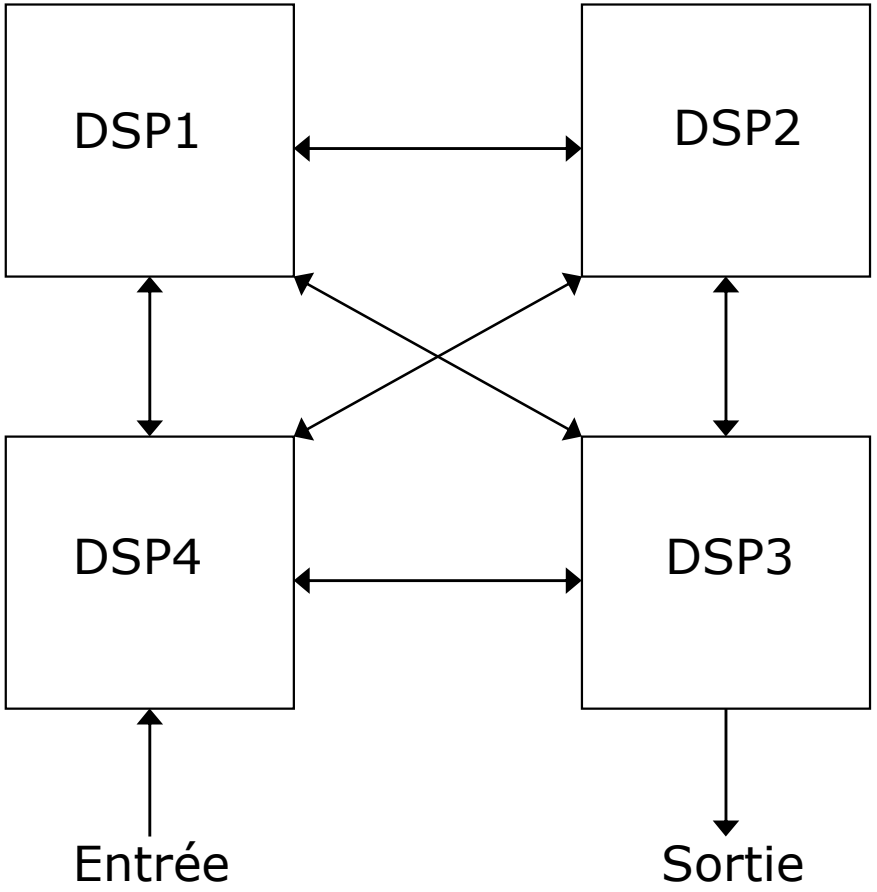
Architecture typique téléphone mobile



Source: Trends in Hardware Architecture for Mobile Devices
Dr. Margarita Esponda, Institut für Informatik Freie Universität Berlin Nov. 2004

herve.le-provost@cea.fr

Applications DSP

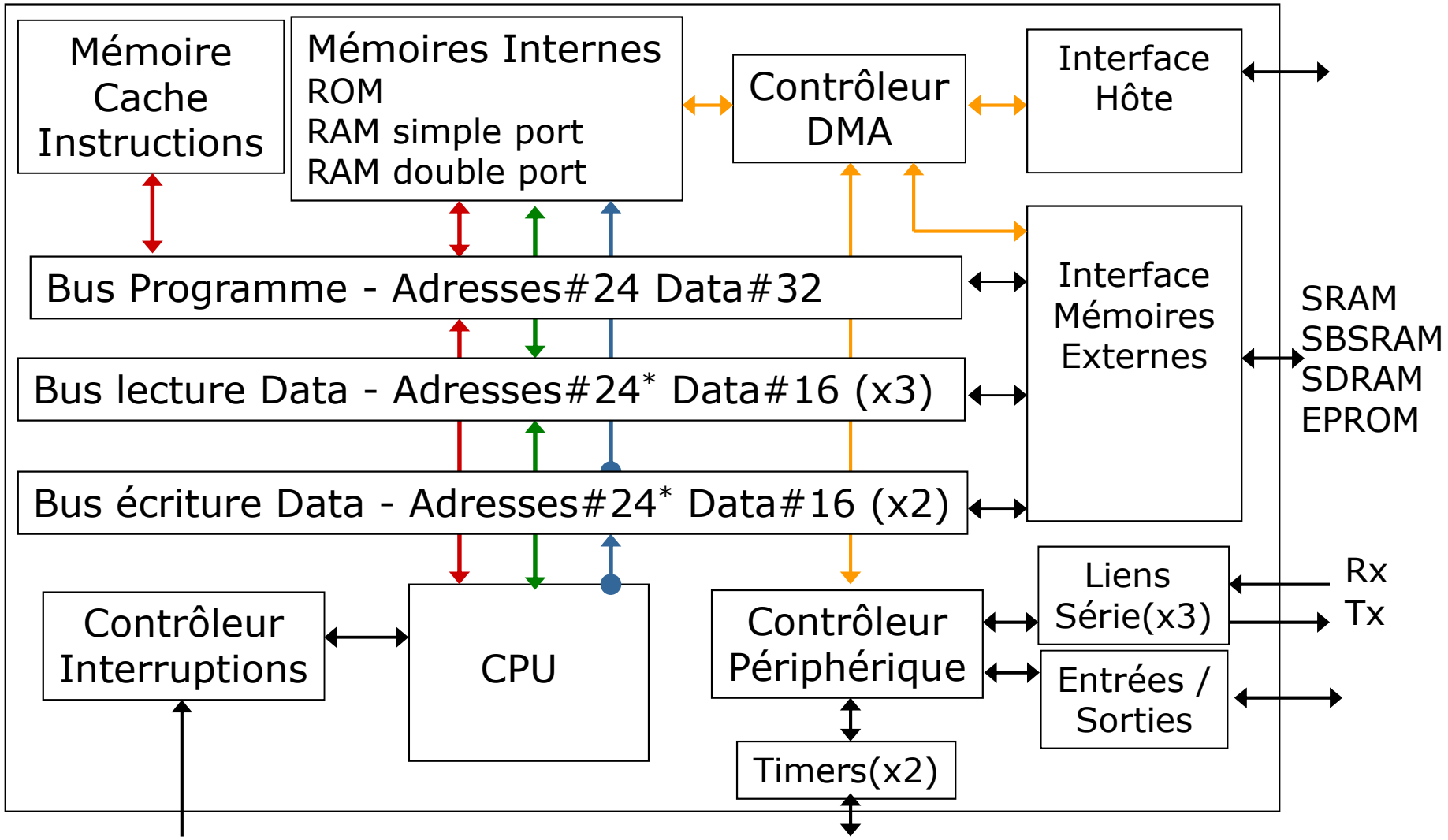


- Système Multiprocesseur
- Liens de communication rapides et dédiés
- Bande passante élevée

- Calcul intensif
- Mise en réseau des DSP
- Évaluation des performances vs un processeur généraliste
- Complexité du développement

herve.le-provost@cea.fr

TMS320C5510



(*)bit de poids faible forcé à 0

herve.le-provost@cea.fr

Performances DSP

- Vitesse:
  - Fréquence maximale 200MHz
  - Durée d'un cycle instruction 5ns@200MHz
- Flot de données (Bande Passante):
  - Jusqu'à 3 lectures et 2 écritures en un seul cycle
  - Jusqu'à 2 transferts DMA par cycle
- Calcul:
  - 2 Multiplieurs-Accumulateur (MAC), 1 multiplication 17-bit x 17-bit par cycle et par MAC
    - Jusqu'à 400 Million Multiplication/Accumulation par seconde
  - 2 Unités Arithmétique et Logique (ALU)
    - 1 ALU centrale 40-bit
    - 1 ALU auxiliaire 16-bit
- Instructions
  - Jeu d'instructions de taille variable, code compact
  - Exécution en pipeline avec prédiction des branchements
  - Cache instructions de 24K-octets
  - Jusqu'à 2 instructions exécutées par cycle

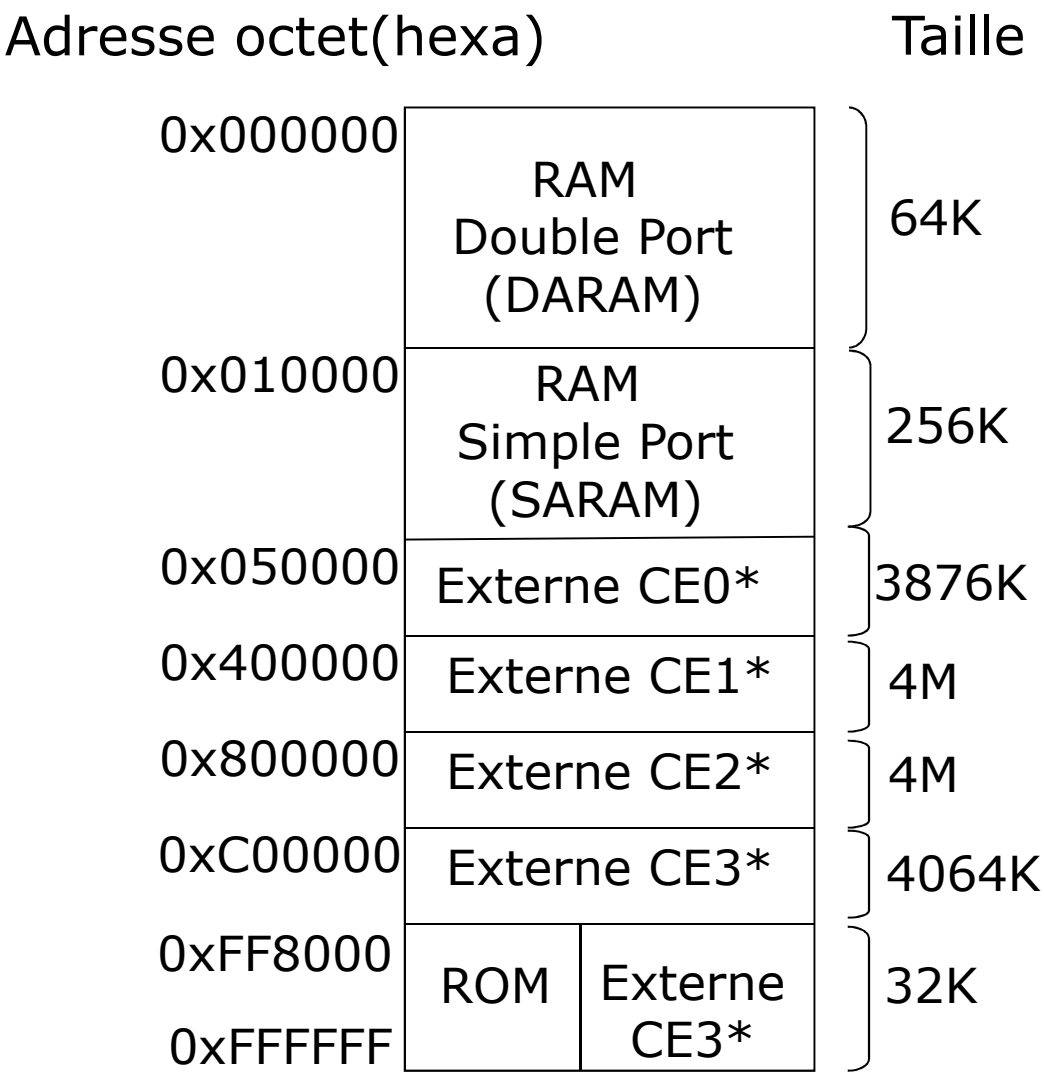
herve.le-provost@cea.fr

Performances DSP

- Interfaces
  - Mémoire externe (EMIF), interface directe aux mémoires asynchrones (EPROM, SRAM) et synchrones (SDRAM, SBSRAM)
  - 3 ports série full duplex (McBSPs) haute performance
  - Processeur Hôte (EHPI) pour accès aux mémoires internes
  - 8 broches d'entrées/sorties à usage général
  - port JTAG
- Gestion du temps
  - 2 timers et un générateur d'horloge
- Mémoires
  - Organisation x16-bit
  - Mémoires internes
    - ROM 32K octets
    - RAM simple port 256K octets
    - RAM double port 64K octets
  - Mémoire externe adressable 8M x 16-bit

herve.le-provost@cea.fr

La mémoire : les plages

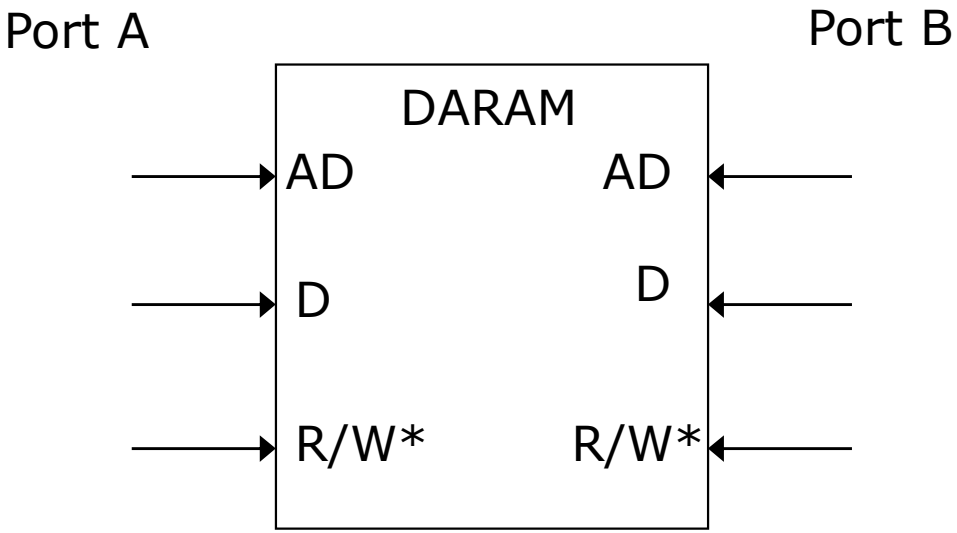


1K = 1024 octets
1M = 1048576 octets

herve.le-provost@cea.fr

La mémoire : Ram Double Port

Plage 0x000000-0x00FFFF / 8 x 8K-octets

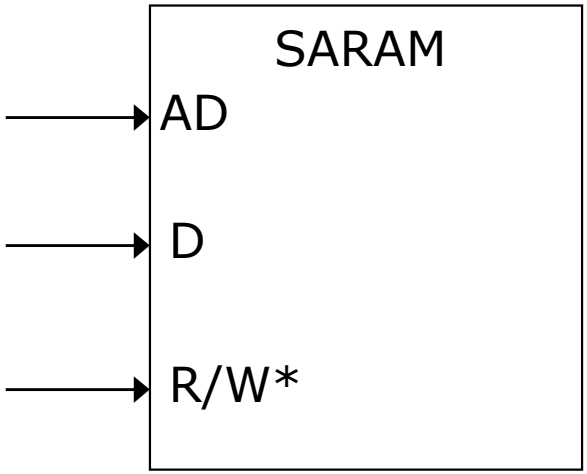


- Accessible par les bus de programme, données et DMA
- 2 accès par cycle:
  - 2 lectures
  - 2 écritures
  - 1 lecture et 1 écriture
- Memory-Mapped (CPU) Registers MMR (192 premier octets)

herve.le-provost@cea.fr

La mémoire : Ram Simple Port

Plage 0x010000-0x04FFFF / 32 x 8K-octets

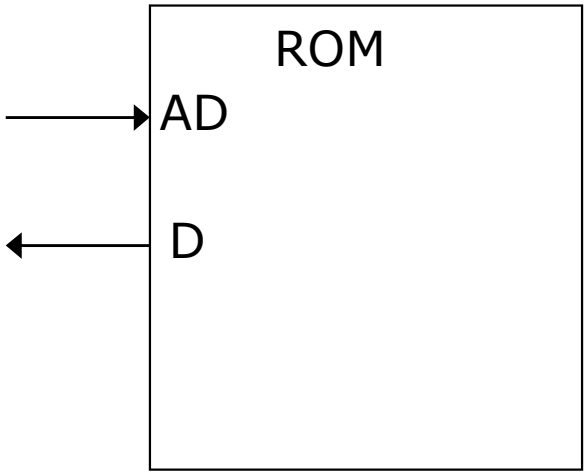


- Accessible par les bus de programme, données et DMA
- 1 accès par cycle:
  - 1 lecture
  - 1 écriture

herve.le-provost@cea.fr

La mémoire : ROM

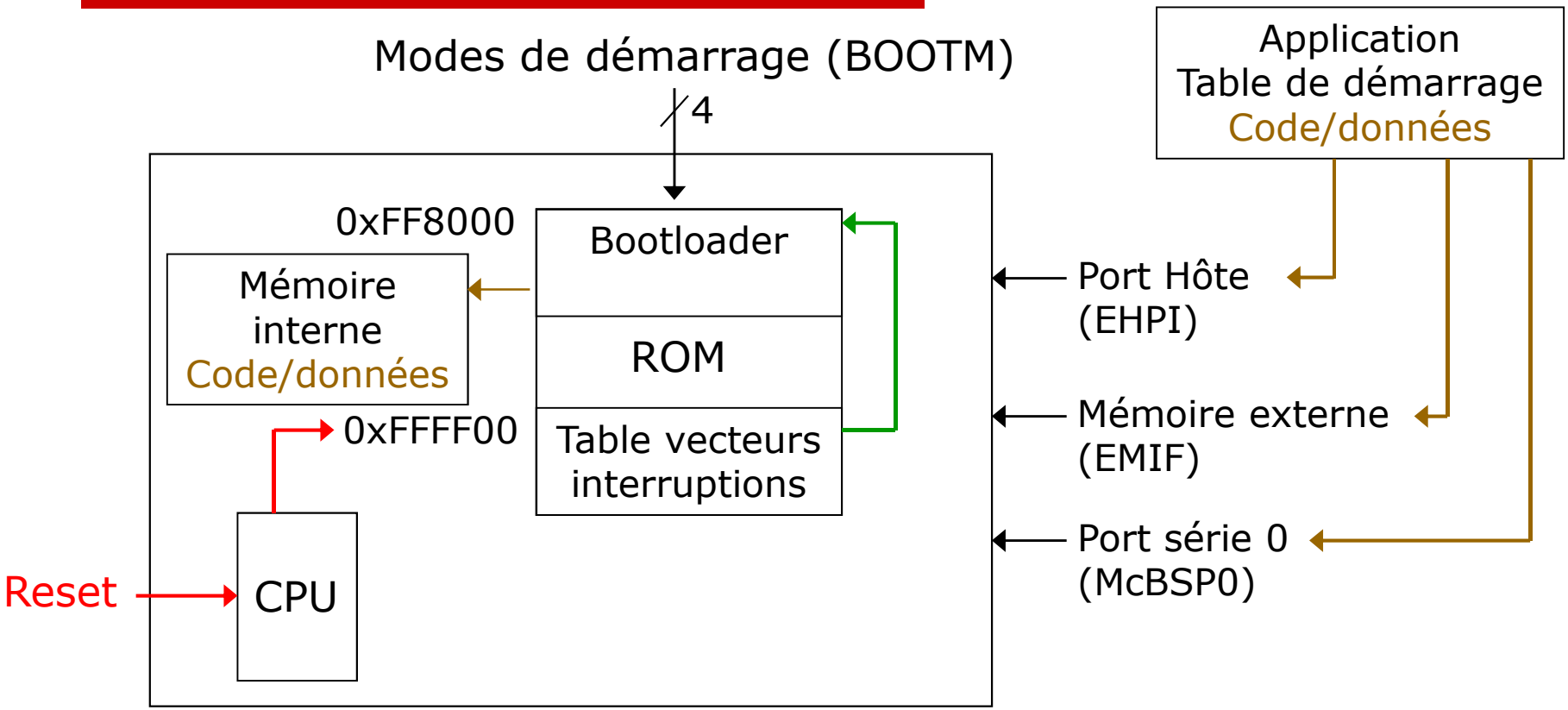
Plage 0xFF8000-0xFFFFFF (validée si MPNMC=0)



- Accessible par les bus de programme, données et DMA
- 1 accès tous les 2 cycles:
  - 1 lecture

herve.le-provost@cea.fr

La mémoire : ROM



- Bootloader:
  - Chargement de la table de démarrage (boot table) à partir du port sélectionné par les broches BOOTM
  - Démarrage de l'application chargée

Que se passe-t-il au reset si la ROM n'est pas présente dans l'espace mémoire (MPNMC=1)?

herve.le-provost@cea.fr

La Mémoire: accès en langage C

```
//TMS320C5510
#include <stdio.h>
int main()
{
printf ("sizeof(char): %d ", sizeof(char));
printf ("sizeof(short): %d ", sizeof(short));
printf ("sizeof(int): %d ", sizeof(int));
printf ("sizeof(long): %d \n", sizeof(long));
printf ("sizeof(long long): %d ", sizeof(long long));
printf ("sizeof(float): %d \n", sizeof(float));
return 0;
}
```

Type	#bit
char	16
short	16
int	16
long	32
long long	40
float	32

```
Sortie console:
sizeof(char): 1 sizeof(short): 1 sizeof(int): 1 sizeof(long): 2
sizeof(long long): 4 sizeof(float): 2
```

1 byte = 16-bit!

ISO C : 'sizeof' renvoie le nombre de bytes pour stocker un objet.

herve.le-provost@cea.fr

La Mémoire: accès en langage C

```
//TMS320C5510
#include <stdlib.h>
#include <stdio.h>
int main()
{
int *ptrI;long *ptrL;long long *ptrLL;
//allocation
ptrI=malloc(12*sizeof(int));
ptrL=malloc(12*sizeof(long));
ptrLL=malloc(12*sizeof(long long));
printf("base ptrI  x'%p  ptrL x'%p ptrLL x'%p\n", ptrI, ptrL, ptrLL);

ptrI++;ptrL++;ptrLL++; //incrementation
printf("  ++ ptrI  x'%p  ptrL x'%p ptrLL x'%p\n", ptrI, ptrL, ptrLL);
return 0;
}
```

```
Sortie console:
base ptrI  x'2002  ptrL x'2010 ptrLL x'202a
  ++   ptrI  x'2003  ptrL x'2012 ptrLL x'202e
```

Attention:
Tableau plage mémoire/adresses sur 24-bit (donnée pointée 8-bit)
Pointeurs C: adresse sur 23-bit (donnée pointée 16-bit)

herve.le-provost@cea.fr

La Mémoire: accès en langage C

```
//TMS320C5510
#include <stdlib.h>
#include <stdio.h>
const int digit[]={0,1,2,3,4,5,6,7};
int main()
{
int *ptr;
int tab[10]; //alloue sur la pile

//alloue dans section sysmem
ptr=malloc(12*sizeof(int));
//sections
printf("variable auto. tab x'%p constante digit x'%p\n", tab , digit);
printf("malloc ptr  x'%p \n", ptr);

return 0;
}
```

```
Sortie console:
variable auto. tab x'266 constante digit x'14102
malloc ptr  x'2002
```

Allocation de la mémoire à Partir d'un fichier descriptif des plages et sections

herve.le-provost@cea.fr

La Mémoire: fichier de commande (1)

```
/* LINKER command file for C5510 memory map.*/
MEMORY
{
   PAGE 0:
      MMR     : origin = 0000000h, length = 00000c0h
      SPRAM   : origin = 00000c0h, length = 0000040

      DARAM0  : origin = 0000100h, length = 0003F00h
      DARAM1  : origin = 0004000h, length = 0004000h…

      SARAM0  : origin = 0010000h, length = 0004000h
      SARAM1  : origin = 0014000h, length = 0004000h…
      SARAM6  : origin = 0028000h, length = 0004000h

      CE0     : origin = 0050000h, length = 03b0000h
      CE1     : origin = 0400000h, length = 0400000h
      CE2     : origin = 0800000h, length = 0400000h
      CE3     : origin = 0c00000h, length = 03f8000h

      PDROM   : origin = 0ff8000h, length = 07f00h
      VECS    : origin = 0ffff00h, length = 00100h  /* reset vector */
}
```


herve.le-provost@cea.fr

La Mémoire: fichier de commande (2)

```
SECTIONS
{
      vectors  : {} > VECS   PAGE 0    /* interrupt vector table */
      .cinit   : {} > SARAM0 PAGE 0
      .cio     : {} > SARAM0 PAGE 0
      .text    : {} > SARAM1 PAGE 0
      isrs     : {} > SARAM2 PAGE 0
      .stack   : {} > DARAM0 PAGE 0
      .sysstack: {} > DARAM0 PAGE 0
      .sysmem  : {} > DARAM1 PAGE 0
      .data    : {} > DARAM1 PAGE 0
      .bss     : {} > DARAM1 PAGE 0
      .const   : {} > SARAM6 PAGE 0
      .coeffs  : {} > DARAM2 PAGE 0
      .dbuffer : {} > DARAM3 PAGE 0
      files    : {} > DARAM2 PAGE 0    /* User-defined sections */
      statvar  : {} > DARAM2 PAGE 0
      statarry : {} > DARAM2 PAGE 0
      tempvar  : {} > DARAM2 PAGE 0
      temparry : {} > DARAM2 PAGE 0
}
```

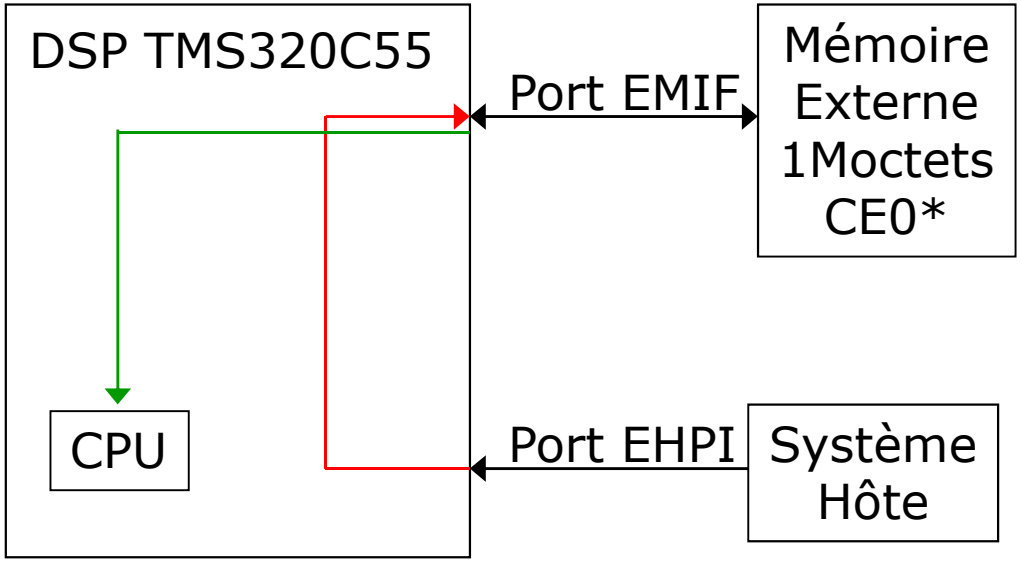
Recoupement avec le programme:

tab x'266*2=x'4CC
ptr x'2002*2=x'4004
digit x'14102*2=x'28204

Sortie console:
variable auto. tab x'266 constante digit x'14102
malloc ptr x'2002

herve.le-provost@cea.fr

La mémoire: exercice



Ecriture de 16 mots de 16-bit à l'adresse de base de l'espace externe 0

Ecrire le programme de lecture et d'affichage des 16 mots de 16-bit (la synchronisation des données est assurée par un mécanisme externe)

herve.le-provost@cea.fr

La mémoire: exercice

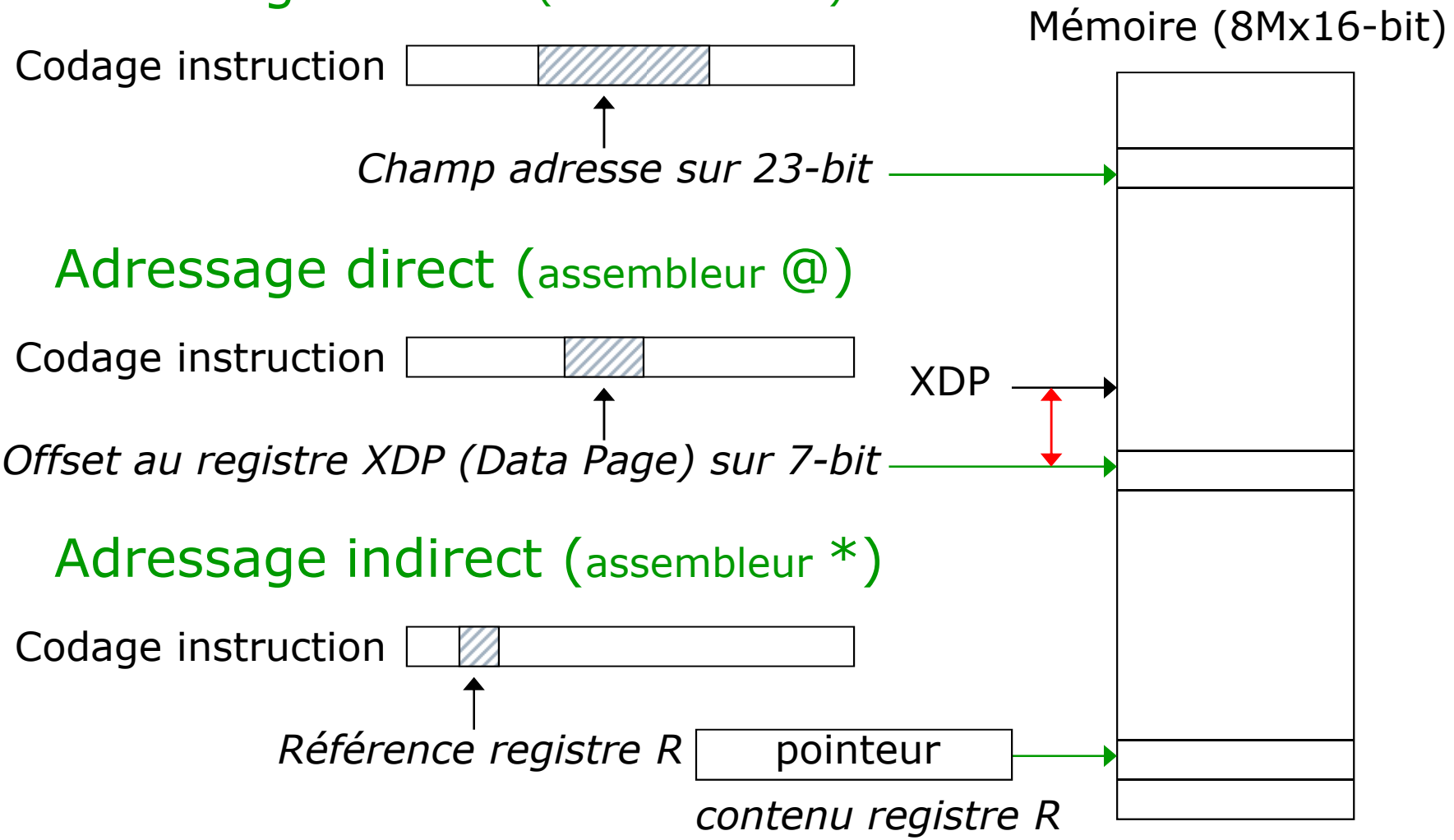
```
//TMS320C5510
#define ADR_CE0 0x28000 //0x50000 en octets
int main ()
{
unsigned short i;
//adresse de base espace 0 alignée sur un double mot
volatile long *pCe0;
volatile long *pSrc;
long tab[8];

pCe0=(long *)ADR_CE0;

pSrc=pCe0;
for (i=0;i<8;i++)
        tab[i]= *pSrc++;

for (i=0;i<8;i++)
        printf("tab[%d]=x'%lx\n", i, tab[i]);

return 0;
}
```

type long : augmentation de la bande Passante, opérande sur 32-bit

herve.le-provost@cea.fr

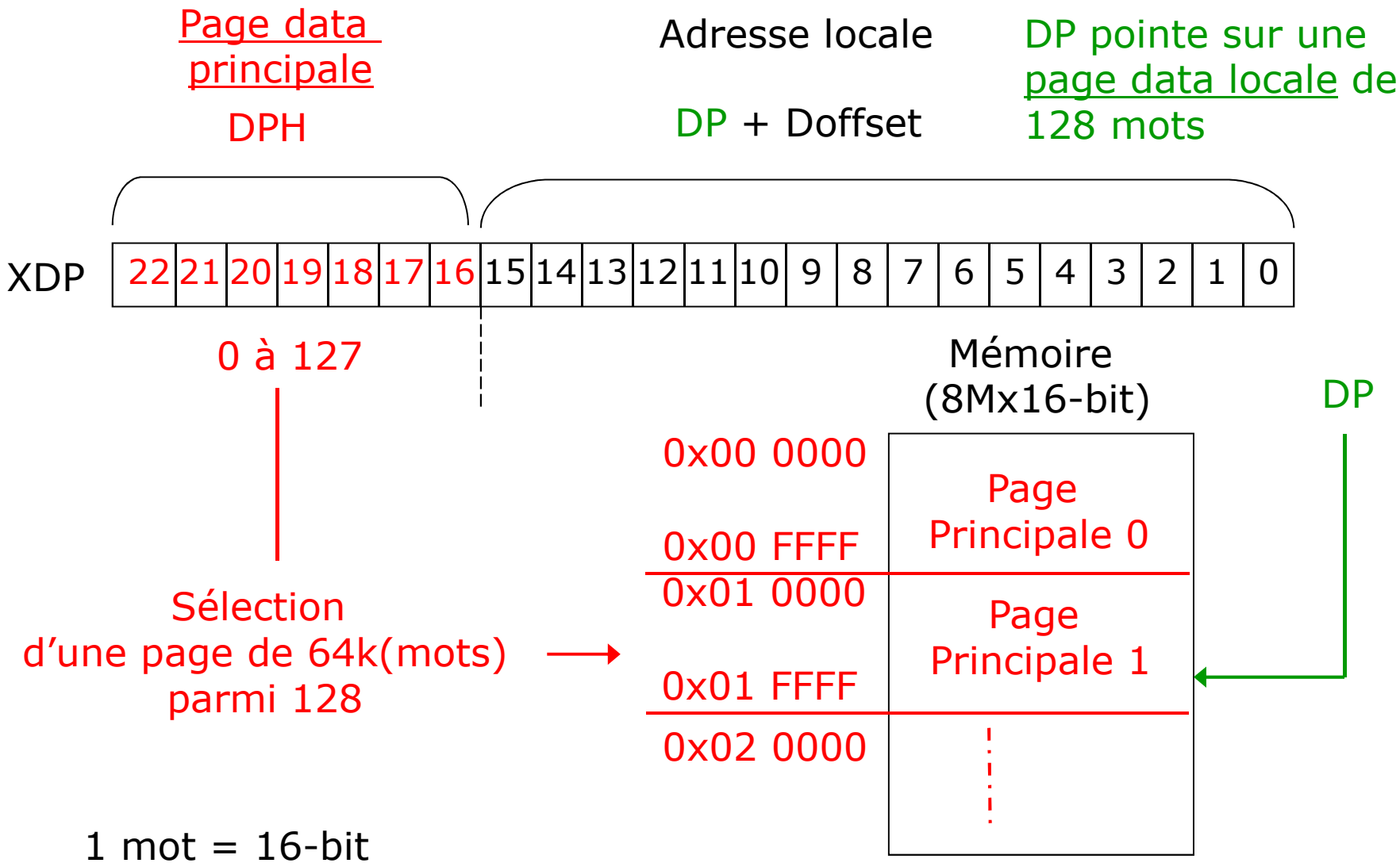
La mémoire: les modes d'adressage

Adressage absolu (assembleur #)



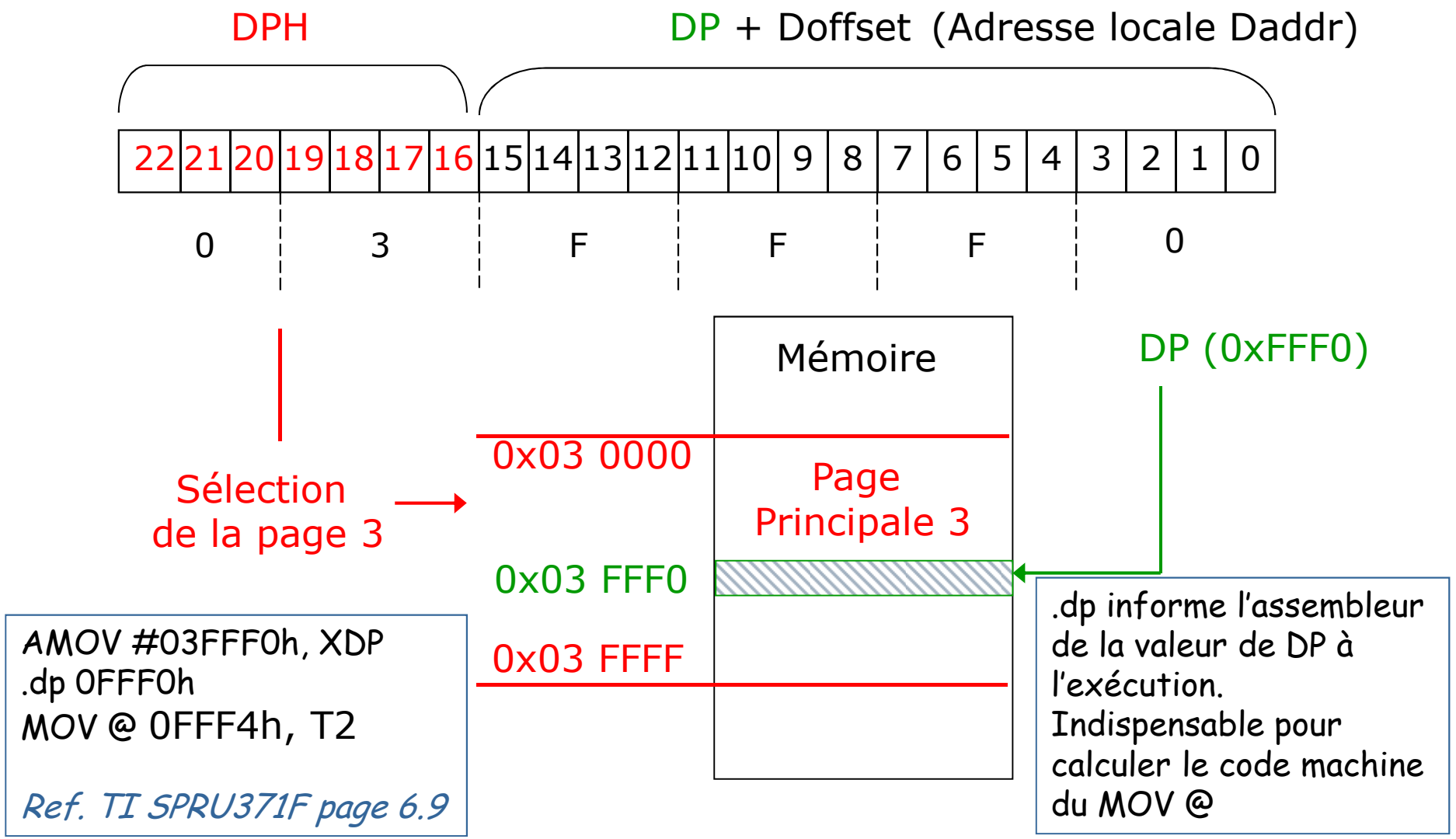
herve.le-provost@cea.fr

La mémoire: l'adressage direct



herve.le-provost@cea.fr

La mémoire: l'adressage direct



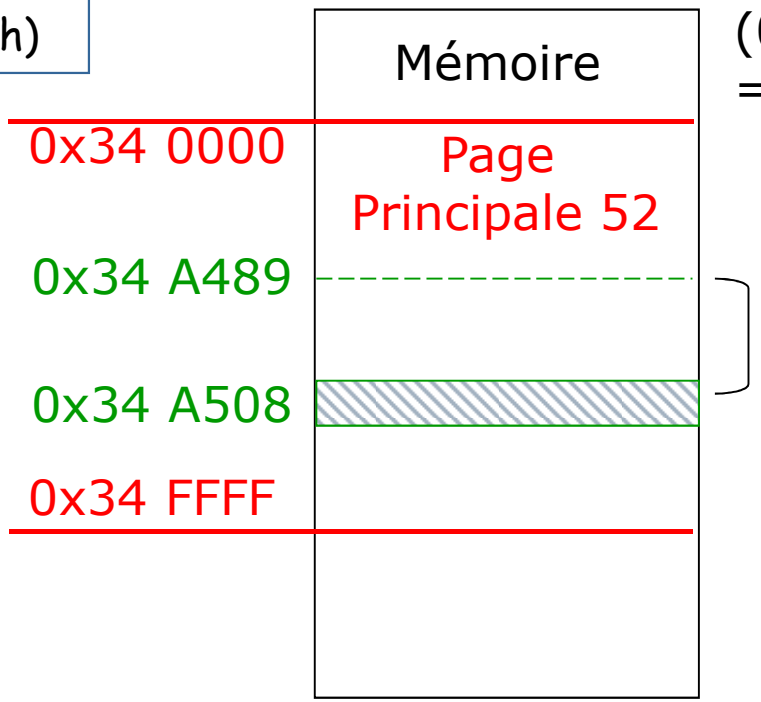
herve.le-provost@cea.fr

Adressage direct: exercice

Ecrire le code assembleur qui initialise l'adresse mémoire 0x34A508 à la valeur 0x1234

```
AMOV #34A4FFh, XDP
.dp 0A4FFh
MOV #1234h, @(0A508h)
```

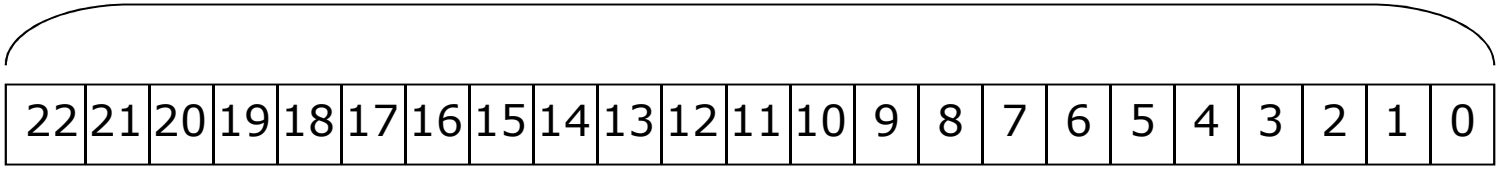
Initialiser DP avec une valeur comprise dans la plage 0xA508 et (0xA508 – 0x7F) =0xA489 (ex. 0xA4FF)

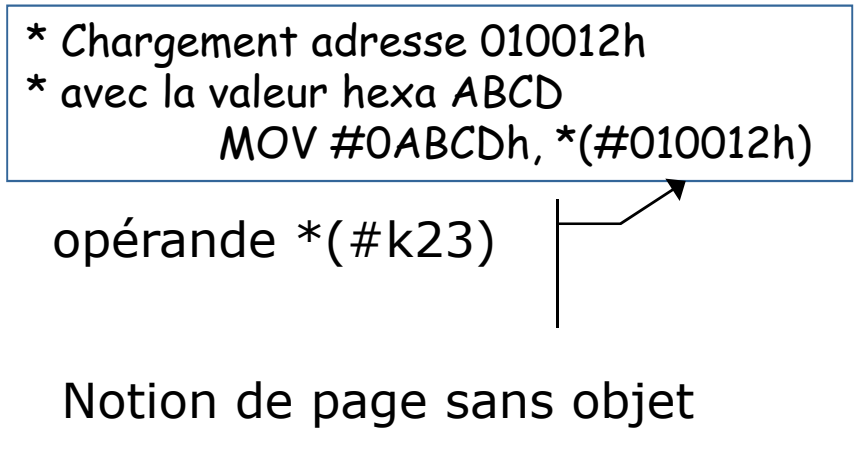


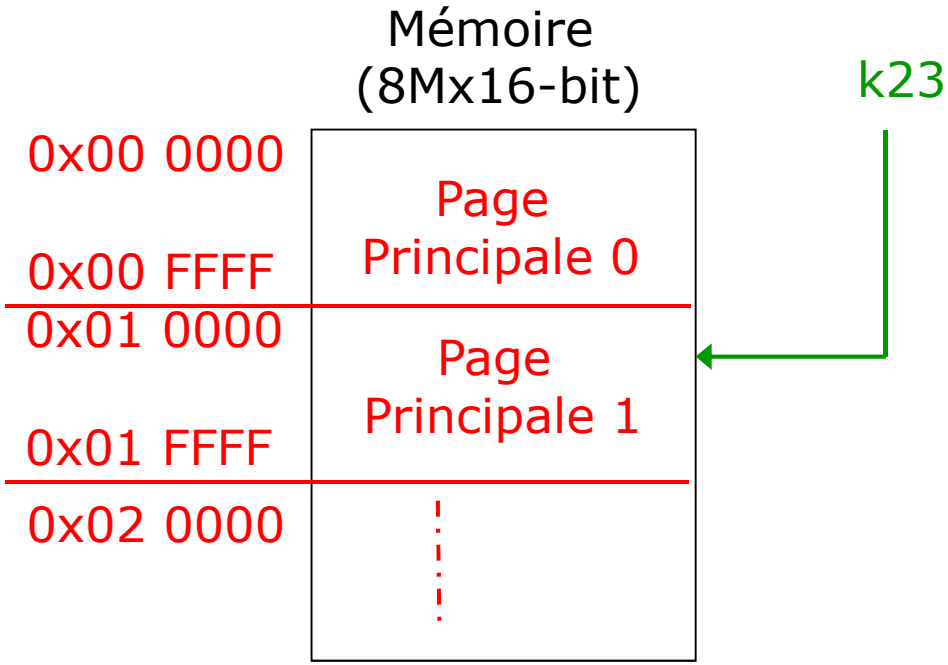
herve.le-provost@cea.fr

La mémoire: l'adressage absolu k23

k23: constante non signée 23-bit

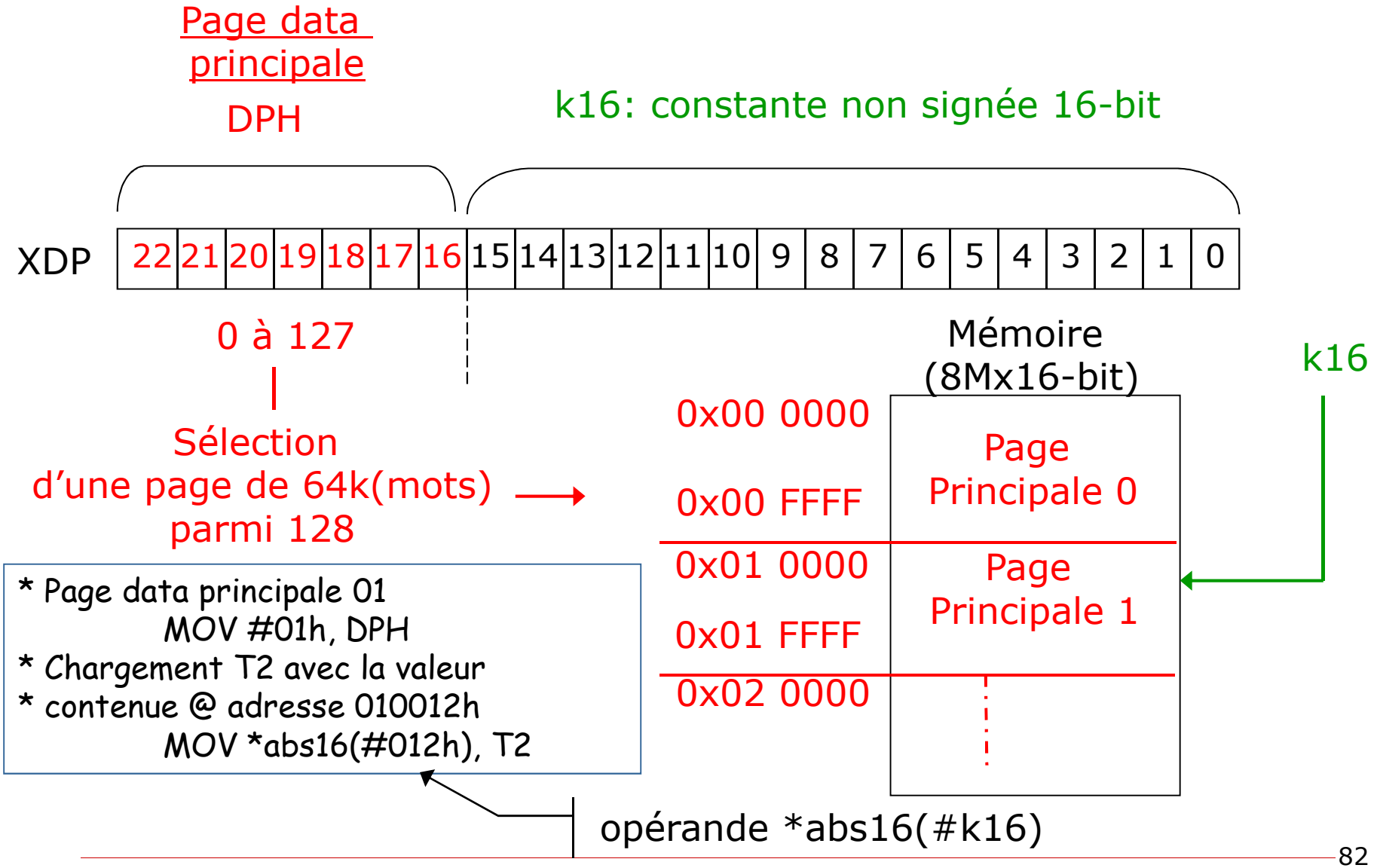






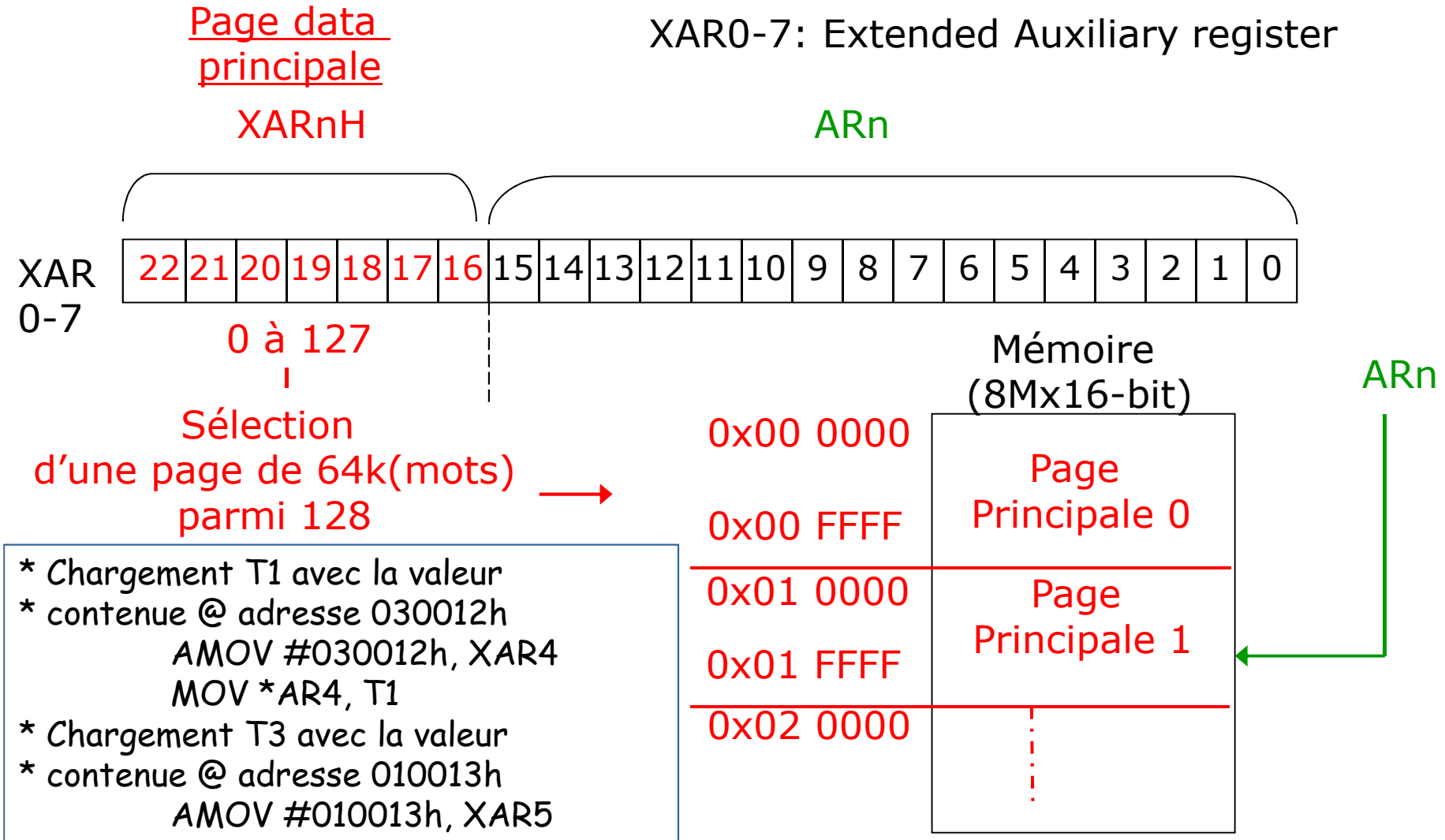
herve.le-provost@cea.fr

La mémoire: l'adressage absolu k16



herve.le-provost@cea.fr

La mémoire: l'adressage indirect



herve.le-provost@cea.fr

La mémoire: les modes d'adressage

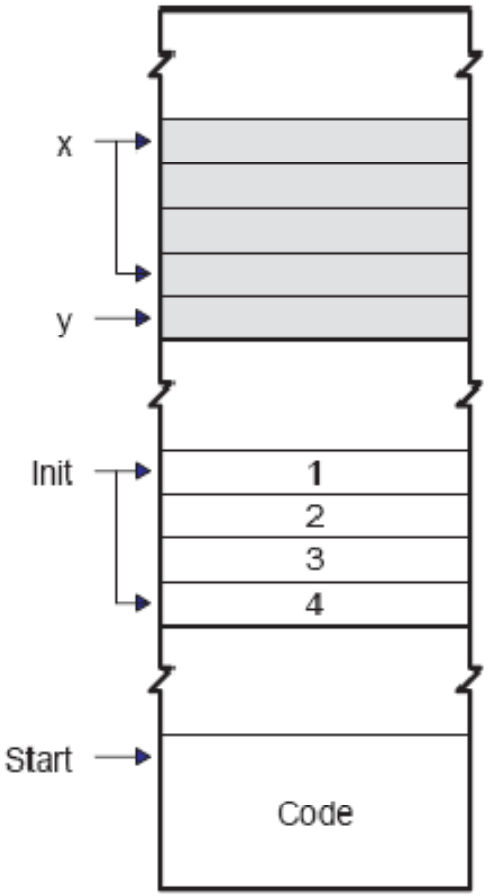
- Tutorial Texas Instruments
  - Programme assembleur y=x0+x1+x2+x3
  - Illustre les 3 modes d'adressage de la mémoire: absolu, direct, indirect
  - 3 étapes
    - Allocation des sections pour le code, les constantes, les variables
    - initialisation du mode de fonctionnement du processeur
    - Codage de l'opération y=x0+x1+x2+x3

herve.le-provost@cea.fr

La mémoire: les modes d'adressage

Allocation des sections pour le code, les constantes, les variables

```
* TMS320C5510
* Step 1: Section allocation
* ----------------------------------------
        .def x,y,init
* reserve 4 uninitalized 16-bit locations for x:
x       .usect vars,4
* reserve 1 uninitialized 16-bit location for y:
y       .usect vars,1
* create initialized section table to
* contain initialization values for x:
        .sect tablei
init .int 1,2,3,4
* create code section (default is .text):
        .text
* define label to the start of the code
        .def start
start
```



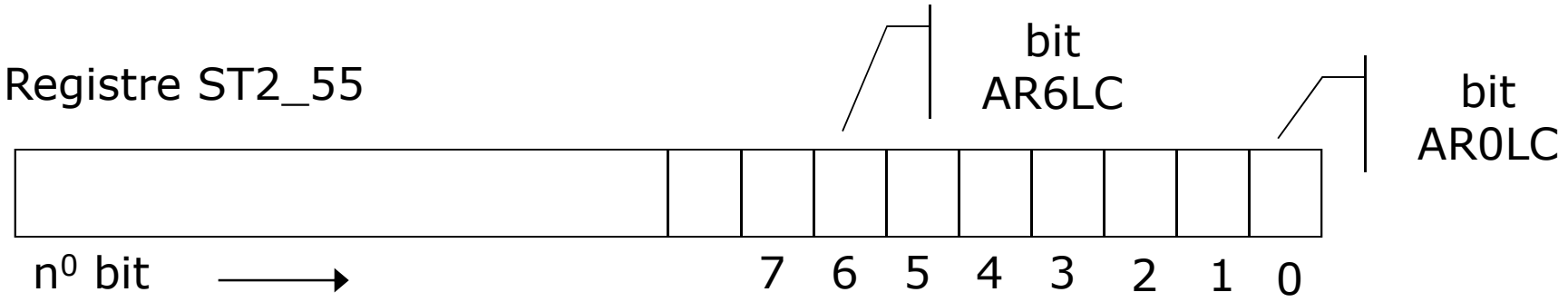
herve.le-provost@cea.fr

La mémoire: les modes d'adressage

initialisation du mode de fonctionnement du processeur

```
* TMS320C5510
* Step 2: Processor mode initialization
* ------
        BCLR C54CM      ; set processor to '55x native mode instead of
                        ; '54x compatibility mode (reset value)
        BCLR AR0LC ; set AR0 register in linear mode
        BCLR AR6LC ; set AR6 register in linear mode
```

Adressage bit dans un registre



herve.le-provost@cea.fr

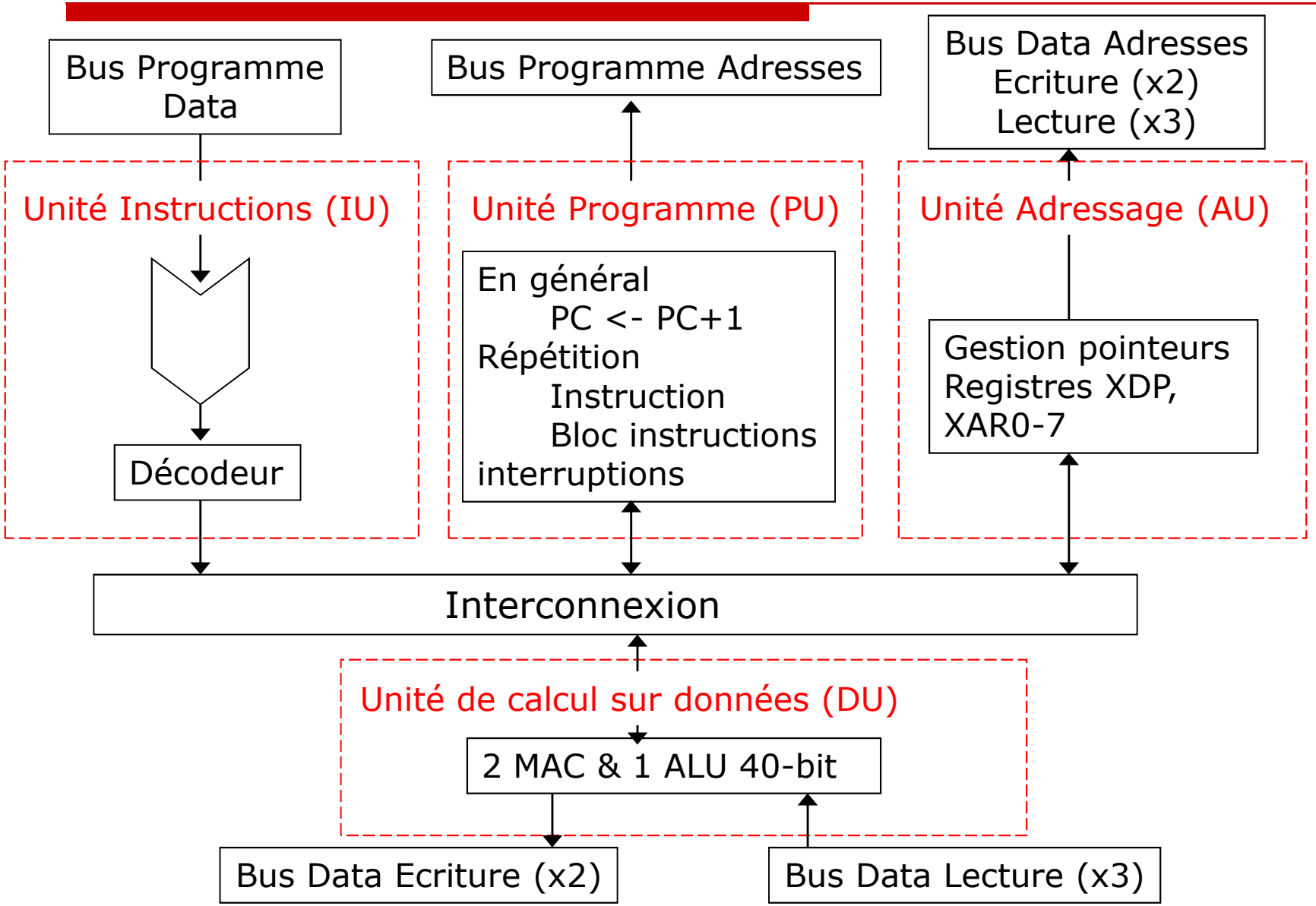
La mémoire: les modes d'adressage

```
* Step 3a: Copy initialization values to vector x using indirect addressing
copy
        AMOV #x, XAR0 ; XAR0 pointing to variable x
        AMOV #init, XAR6 ; XAR6 pointing to initialization table
        MOV *AR6+, *AR0+ ; copy starts from init to x
        MOV *AR6+, *AR0+
        MOV *AR6+, *AR0+
        MOV *AR6, *AR0
* Step 3b: Add values of vector x elements using direct addressing
add
        AMOV #x, XDP ; XDP pointing to variable x
        .dp x ; and the assembler is notified
        MOV @x, AC0
        ADD @(x+3), AC0
        ADD @(x+1), AC0
        ADD @(x+2), AC0
* Step 3c. Write the result to y using absolute addressing
        MOV AC0, *(#y)
end
        NOP
        B end
```

Codage de l'opération y=x0+x1+x2+x3

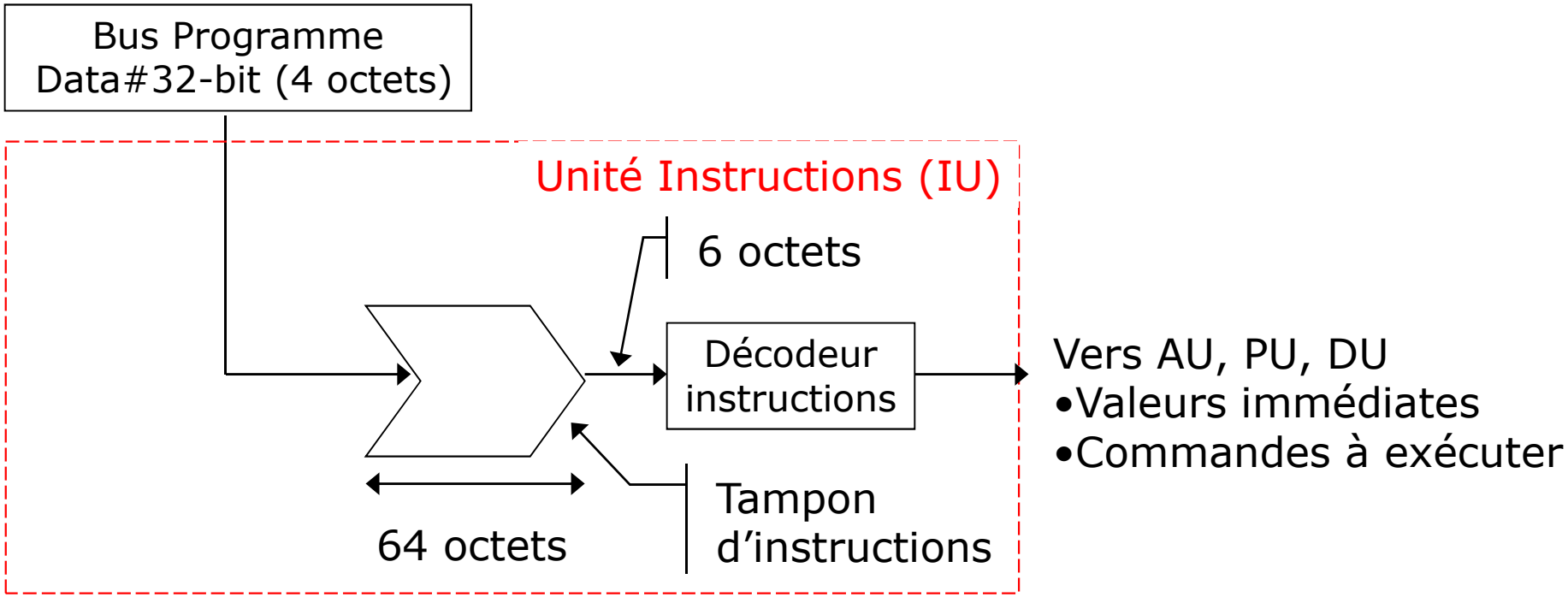
herve.le-provost@cea.fr

TMS320C5510 - Architecture CPU simplifiée



herve.le-provost@cea.fr

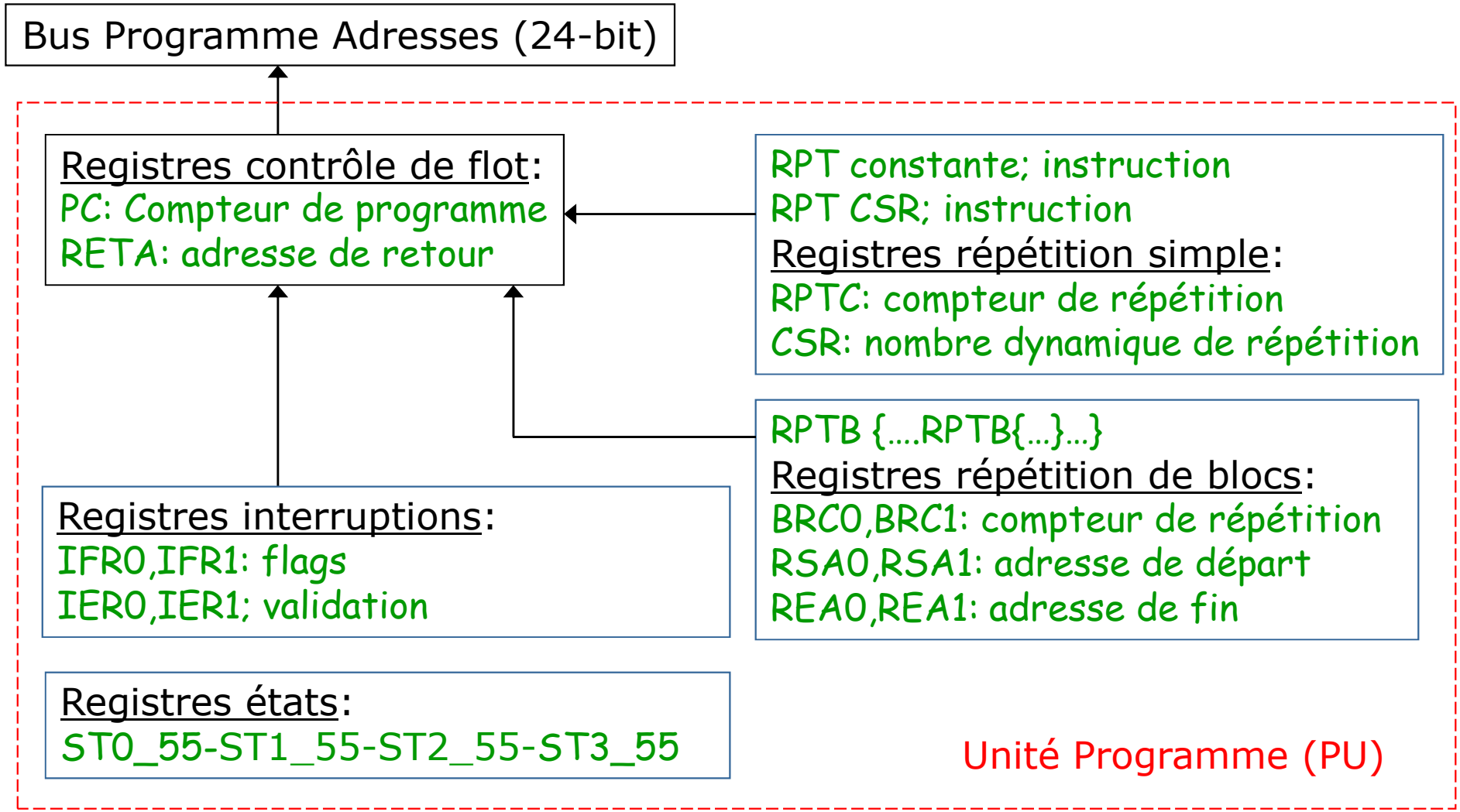
TMS320C5510 – Unité Instructions (IU)



- Répétition d'un bloc de code (instruction de répétition locale RPTBLOCAL), longueur du code inférieure à 64 octets
- Le décodeur extrait 6 octets du tampon d'instruction et identifie les frontières des instructions simples codées sur 8, 16 24, 32, 40 ou 48-bit

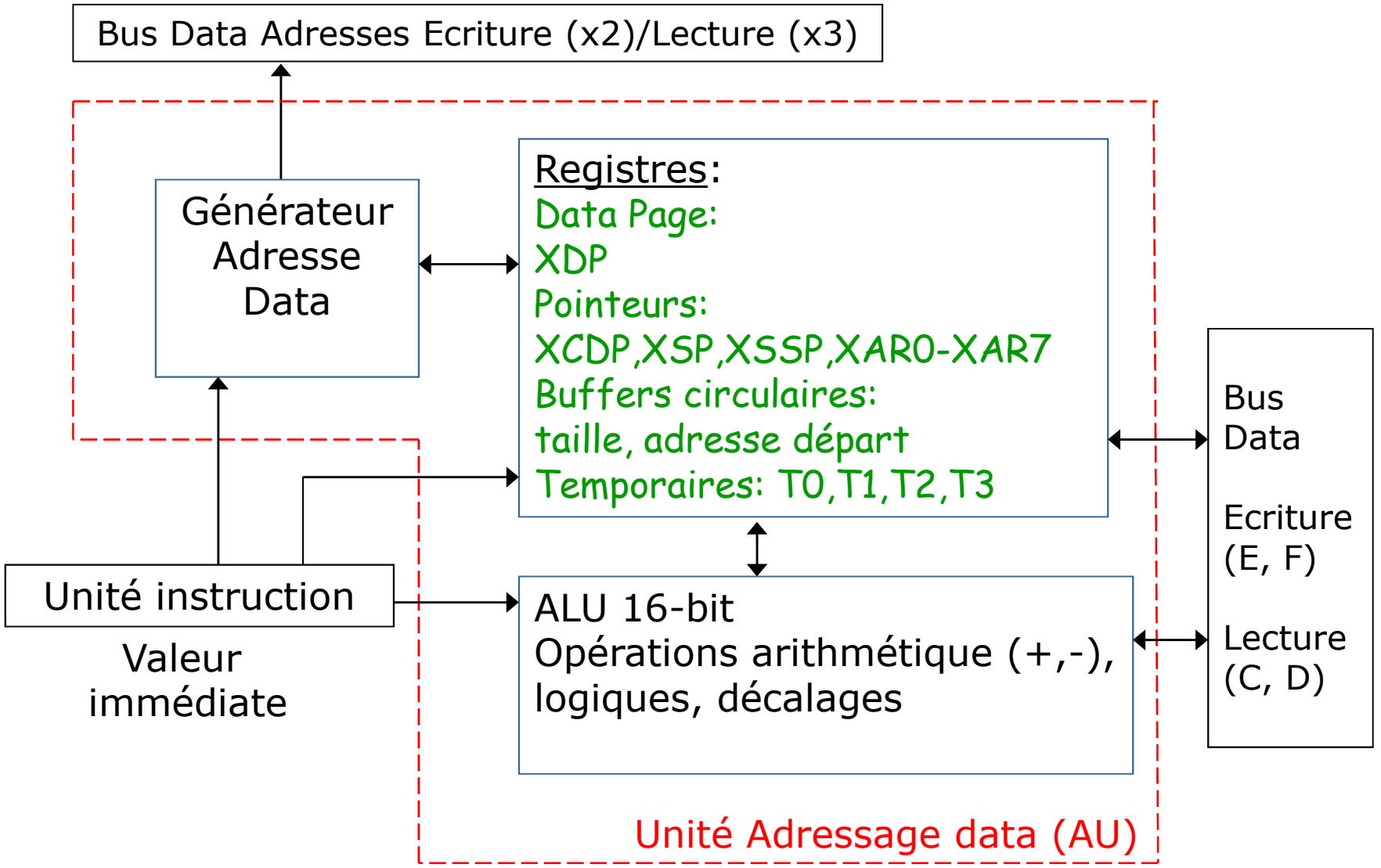
herve.le-provost@cea.fr

TMS320C5510 – Unité Programme (PU)



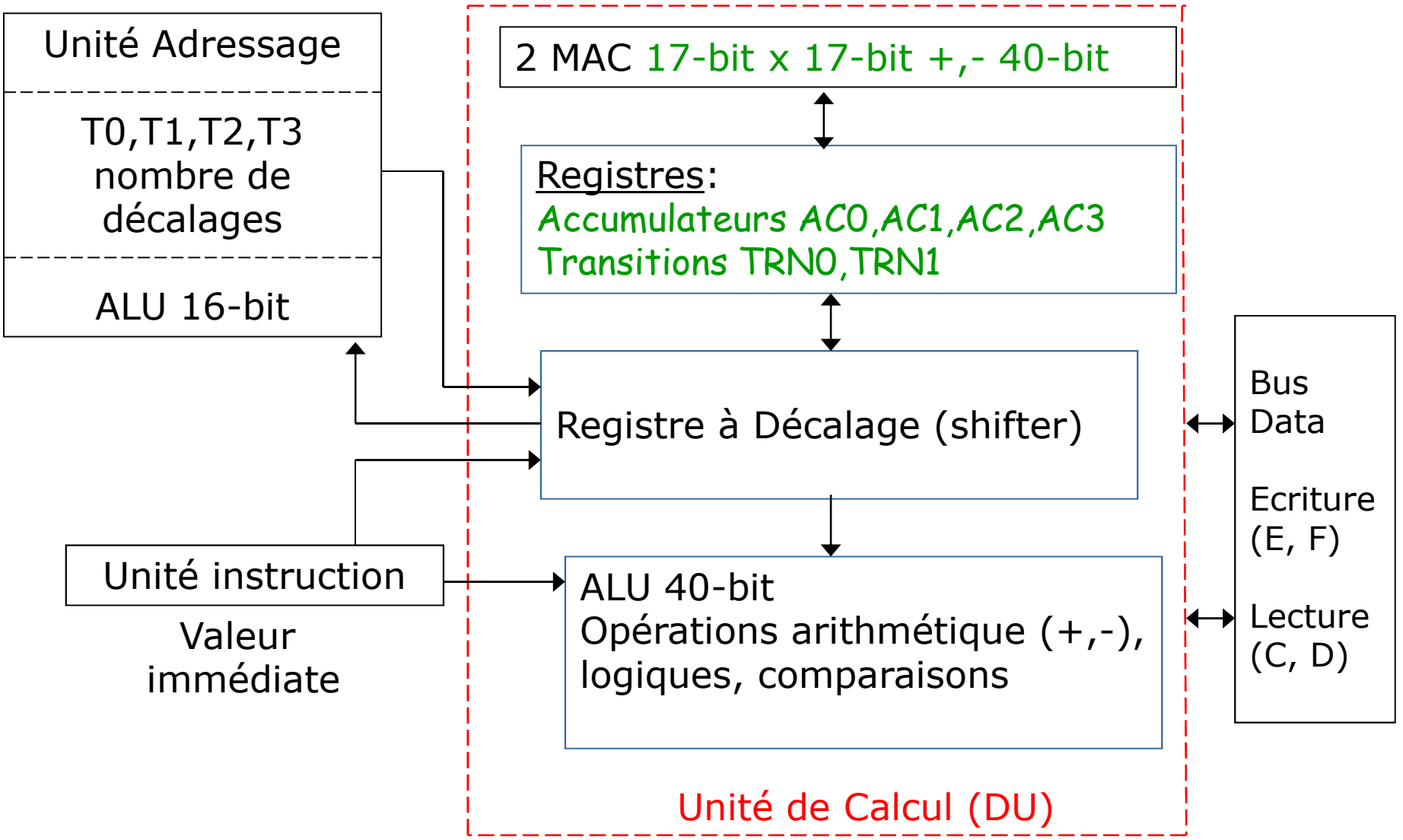
herve.le-provost@cea.fr

TMS320C5510 – Unité Adressage data (AU)



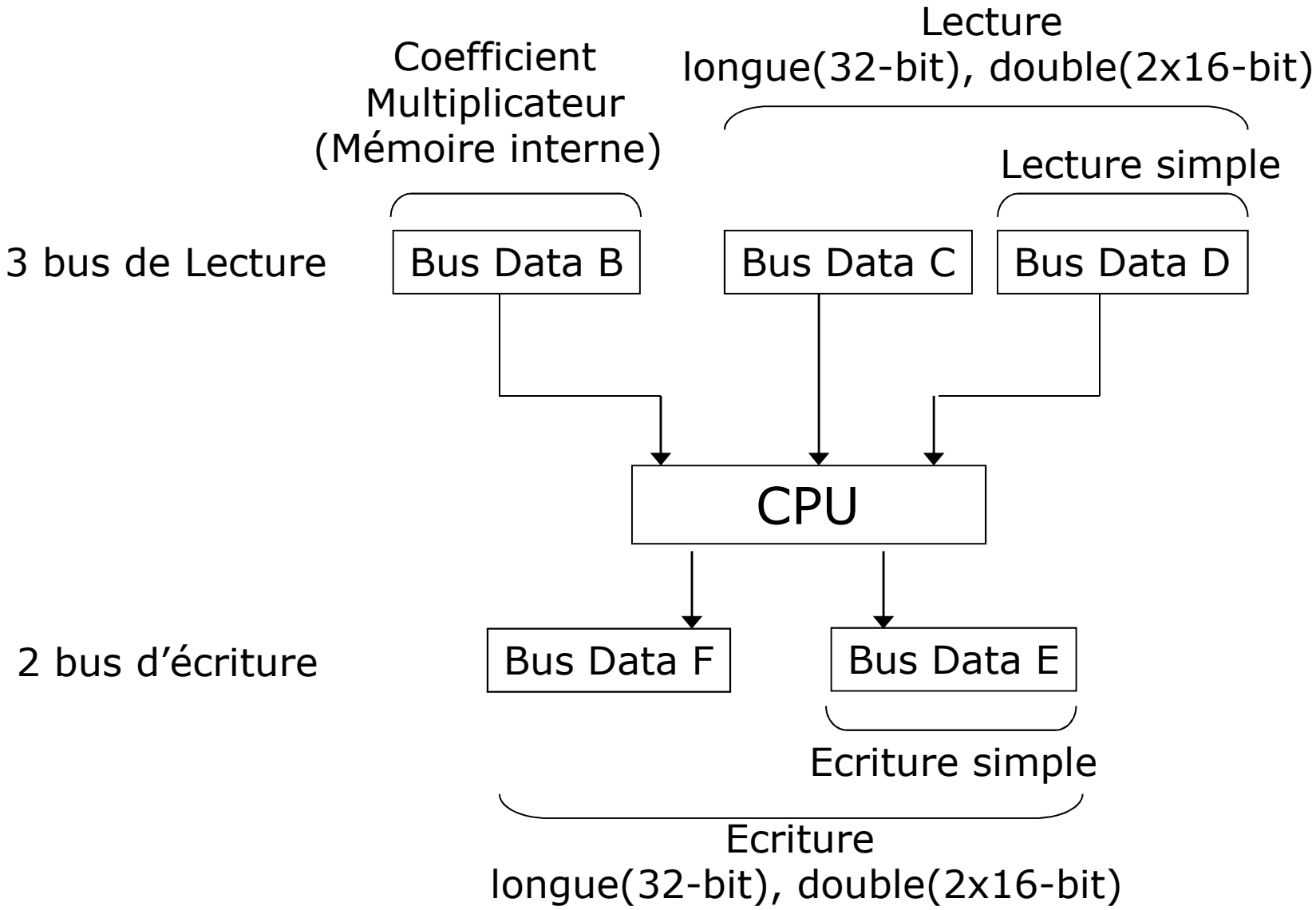
herve.le-provost@cea.fr

TMS320C5510 – Unité de calcul (DU)



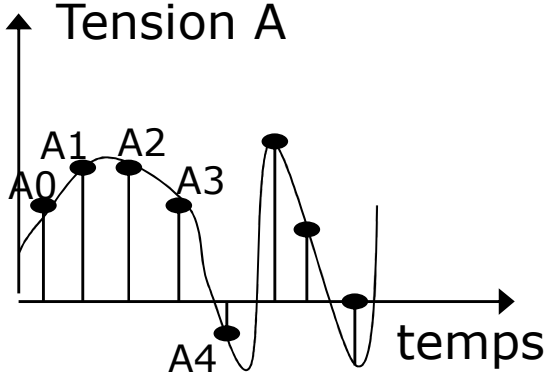
herve.le-provost@cea.fr

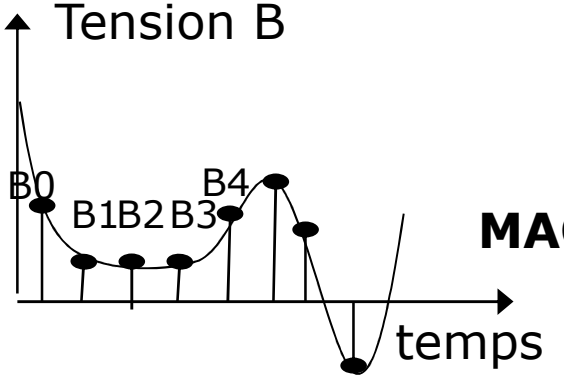
TMS320C5510 – Les bus de données



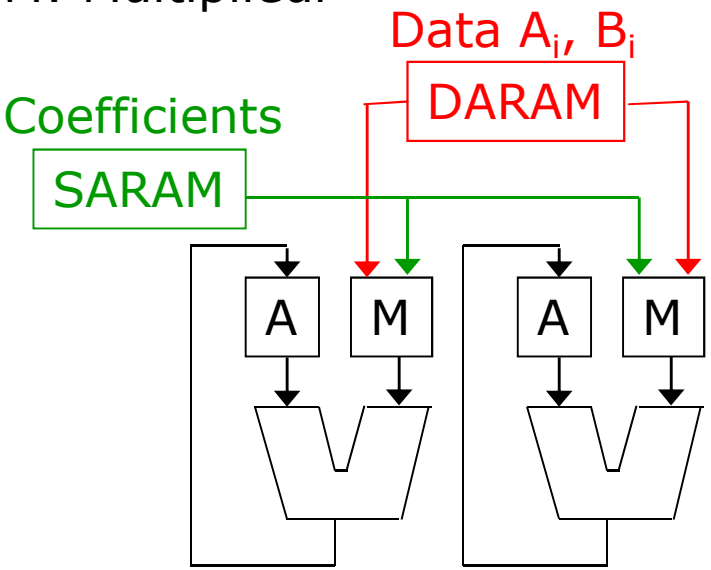
herve.le-provost@cea.fr

DUAL MAC





A: Accumulateur (AC0-1-2-3)
M: Multiplieur



MAC Xmem, Cmem, ACx**:: MAC** Ymem, Cmem, ACy

ACx = ACx + (Xmem * Cmem)
:: ACy = ACy + (Ymem * Cmem)

Placement des data est des coefficients en mémoire pour optimiser les performances

herve.le-provost@cea.fr

DUAL MAC Exercice

Coefficients C

1
2
3
4

Data A

5
6
7
8

Data B

9
10
11
12

Ecrire le code assembleur C5510 qui optimise le code C:

```
int main ()
{
int i;
onchip int coeff[4]={1,2,3,4};
int mac1[4]={5,6,7,8};
int mac2[4]={9,10,11,12};
int res1, res2;
for (i=0;i<4;i++)
{
        res1+=coeff[i]*mac1[i];
        res2+=coeff[i]*mac2[i];
}
return 0;
}
```

MACM Xmem, Ymem, ACx, ACy
ACy = ACx + **(**Xmem * Ymem**)**

Compilateur C (après optimisation): 2 instructions MACM

herve.le-provost@cea.fr

Exercice Dual MAC

```
* Step 1: Section allocation
* ----------------------------------------
        .def coeff, mac1, mac2, res1, res2
* reserve 1 uninitalized 16-bit locations for res1 and res2 :
res1    .usect vars,1
res2    .usect vars,1
* create initialized section table to
* contain initialization values for coeff, mac1 and mac2:
        .sect tab_coeff
coeff .int 1,2,3,4
        .sect tab_mac1
mac1 .int 5,6,7,8
        .sect tab_mac2
mac2 .int 9,10,11,12
* create code section (default is .text):
        .text
* define label to the start of the code
        .def start
```


herve.le-provost@cea.fr

Exercice Dual MAC

```
start
* Step 2: Clear accumulators
* -------
            AND #0, AC0, AC0
            AND #0, AC1, AC1
* Step 3: Calculate for i=0,1,2,3
* sigma(coeff(i).mac1(i))::sigma(coeff(i).mac2(i))
* -------
calcul
            AMOV #mac1, XAR0
            AMOV #mac2, XAR1
            AMOV #coeff, XCDP
            RPT #3
            MAC *AR0+,*CDP+,AC0::MAC *AR1+,*CDP+,AC1

* Step 4. Write MAC results using absolute addressing
* -------
            MOV AC0, *(#res1)
            MOV AC1, *(#res2)
end
            NOP
            B end
```


herve.le-provost@cea.fr

Exercice Dual MAC

```
/* extrait linker.cmd-fichier de commande-mapping mémoire C5510 */
SECTIONS
{
        .text           : {} > SARAM0 PAGE 0
        .data           : {} > DARAM1 PAGE 0
        .bss            : {} > DARAM0 PAGE 0
        vars            : {} > SARAM8 PAGE 0
        tab_coeff       : {} > SARAM7 PAGE 0
        tab_mac1        : {} > DARAM2 PAGE 0
        tab_mac2        : {} > DARAM2 PAGE 0
}
```


herve.le-provost@cea.fr

Exercice Dual MAC : les performances

*nombre maximum de " repeat " compatible avec le budget mémoire

- Utilisation de l'outil de profilage de "Code Composer Studio"
  - Profile->Clock->View
  - Profile->Clock->Setup, count=Cycle.Total

On fait varier le nombre de " repeat " et on mesure le nombre total de cycles* pour exécuter les 2 instructions

```
RPT #4
MAC *AR0+,*CDP+,AC0::MAC *AR1+,*CDP+,AC1
```

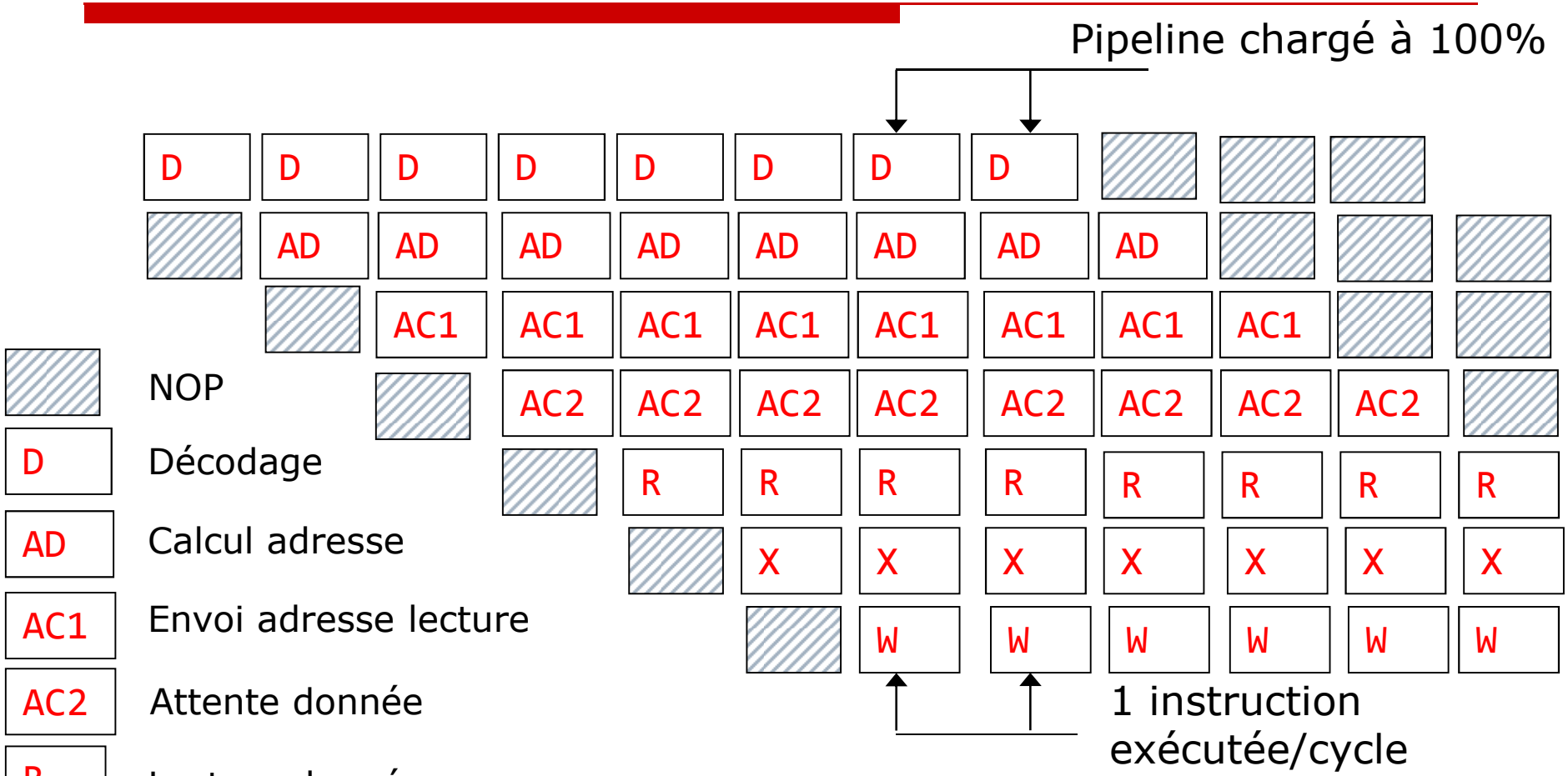
Nombre moyen de cycles par instruction

#RPT	#CYCLE	RATIO (#cycle/(#RPT+1))
1	10	5
4	13	2,6
32	41	1,24
256	265	1,03

Pourquoi le nombre moyen de cycles par instruction ↘ quand #RPT ↗ ?

herve.le-provost@cea.fr

TMS320C5510 : PIPELINE des INSTRUCTIONS



Data manual :Dual Multipliers (Up to 400 Million Multiply-Accumulates Per Second (MMACS))

Comment est calculée cette performance ?

herve.le-provost@cea.fr

TMS320C5510 : performances MMACS

- Conditions :
  - Horloge@200MHz, 200 Méga instructions par seconde
  - Pas de goulot d'étranglement lié à la bande passante mémoire si les opérandes et coefficients sont en mémoire interne
  - Instruction dual MAC (MAC::MAC)
- Calcul
  - 200 Méga*2=400 MMACS

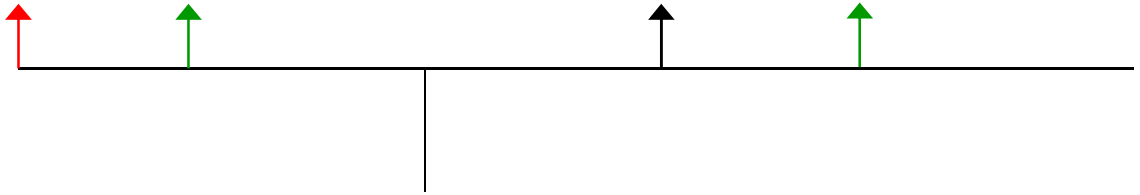
Calculer la bande passante mémoire (cumulée) correspondant au calcul de la performance de 400 MMACS

herve.le-provost@cea.fr

TMS320C5510 : performances MMACS

Bande passante mémoire (cumulée) correspondant au calcul de la performance de 400 MMACS

MAC Xmem, Cmem, ACx**:: MAC** Ymem, Cmem, ACy



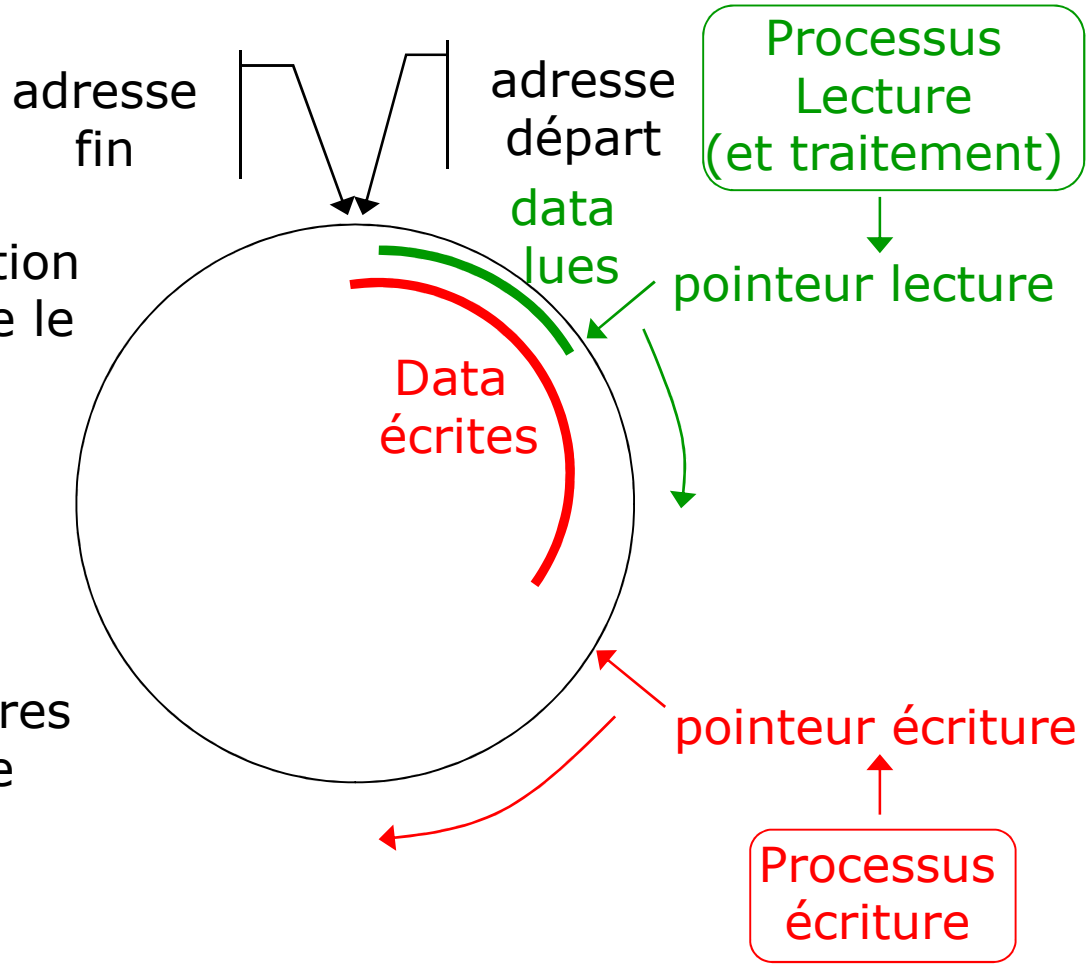
3 accès mémoire par cycle

@200MHz, 3 mots de 16-bit transférées en 5 ns
soit 6 octets en 5 ns ou encore 1,2 Go/s

herve.le-provost@cea.fr

Tampons circulaires

adresse fin
=
adresse départ-1



Mécanismes logiciels de protection si le pointeur d'écriture rattrape le pointeur de lecture

TMS320C5510

- Utilisé en adressage indirect (AR0-AR7, CDP)
- Bit de configuration des registres AR0-AR7, CDP en mode linéaire ou circulaire.
- Buffer Start Address Register (BSA)
- Buffer Size Register (BK)

herve.le-provost@cea.fr

Tampons circulaires : TMS320C5510

Pointer	Linear/Circular Con-figuration Bit	Supplier of Main Data Page	Buffer Start Address Register	Buffer Size Register
AR0	ST2_55(0) = AR0LC	AR0H	BSA01	BK03
AR1	ST2_55(1) = AR1LC	AR1H	BSA01	BK03
AR2	ST2_55(2) = AR2LC	AR2H	BSA23	BK03
AR3	ST2_55(3) = AR3LC	AR3H	BSA23	BK03
AR4	ST2_55(4) = AR4LC	AR4H	BSA45	BK47
AR5	ST2_55(5) = AR5LC	AR5H	BSA45	BK47
AR6	ST2_55(6) = AR6LC	AR6H	BSA67	BK47
AR7	ST2_55(7) = AR7LC	AR7H	BSA67	BK47
CDP	ST2_55(8) = CDPLC	CDPH	BSAC	BKC

Source: Texas Instruments, spru374g, page 3-20

herve.le-provost@cea.fr

Tampons circulaires : exemple TMS320C5510

```
* Circular buffer-source : Texas Instruments spru371f page 6.111
* ----------------------------------------------
* create code section (default is .text):
        .text
* define label to the start of the code
        .def start
start
        MOV #3, BK03            ; Circular buffer size is 3 words
        BSET AR1LC              ; AR1 is configured to be modified circularly
        AMOV #010000h, XAR1     ; Circular buffer is in main data page 01
        MOV #0A02h, BSA01       ; Circular buffer start address is 010A02h
        MOV #0000h, AR1         ; Index (in AR1) is 0000h
        MOV *AR1+, AC0          ; AC0 loaded from 010A02h + (AR1) = 010A02h,
                                ; and then AR1 = 0001h
        MOV *AR1+, AC0          ; AC0 loaded from 010A02h + (AR1) = 010A03h,
                                ; and then AR1 = 0002h
        MOV *AR1+, AC0          ; AC0 loaded from 010A02h + (AR1) = 010A04h,
                                ; and then AR1 = 0000h
        MOV *AR1+, AC0          ; AC0 loaded from 010A02h + (AR1) = 010A02h,
                                ; and then AR1 = 0001h
end
        B end
```


herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)

ABS	Absolute value
ADD	Addition
AND	Bitwise AND
B	Branch
CALL	Function call
CLR	Assign the value to 0
CMP	Compare
CNT	Count
EXP	Exponent
MAC	Multiply and accumulate
MAR	Modify auxiliary register content
MAS	Multiply and subtract
MAX	Maximum
MIN	Minimum
MOV	Move data
MPY	Multiply
NEG	Negate (2s complement)
NOT	Bitwise complement (1s complement)
OR	Bitwise OR
POP	Pop from top of the stack
PSH	Push to top of the stack
RET	Return
ROL	Rotate left
ROR	Rotate right
RPT	Repeat
SAT	Saturate
SET	Assign the value to 1
SFT	Shift (left or right depending on sign of shift count)
SQA	Square and add
SQR	Square
SQS	Square and subtract
SUB	Subtraction
SWAP	Swap register contents
TST	Test bit
XOR	Bitwise exclusive-OR (XOR)
XPA	Expand
XTR	Extract

herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)

Préfixes mnémoniques

Préfixe A: l'instruction est effective dans la phase de calcul de l'adresse et est exécutée dans l'unité d'adressage des data (AU).
exemple :
instruction AMOV TAx, TAy

TAx, TAy	Auxiliary register (ARx) or temporary register (Tx): AR0, AR1, AR2, AR3, AR4, AR5, AR6, AR7 T0, T1, T2, T3

Préfixe B: instruction BIT
exemple :
instruction BTST src, Smem, TCx

Numéro du bit à tester

Copie du bit (TC0 ou TC1)

Smem	Word single data memory access (16-bit data access)
src	AC0, AC1, AC2, AC3 AR0, AR1, AR2, AR3, AR4, AR5, AR6, AR7 T0, T1, T2, T3

BTST AC0,*AR0,TC1

herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)

Suffixes mnémoniques (combinaisons possibles MKR{40,A,,Z ou U}

40	Enables the M40 mode (all 40 bits of the accumulator count)
B	Borrow
C	Carry
CC	Conditional
I	Enable interrupts
K	Multiply has a constant operand
L	Logical shift (left or right depending on sign of shift count)
M	This instruction has the option of assigning a memory operand to T3; regardless of whether that assignment actually occurs.
R	Round
S	Signed shift (left or right depending on sign of shift count)
U	Unsigned
V	Absolute value
Z	Delay on the memory operand

Nombre de décalages
décalage à droite <0
décalage à gauche >0

exemple:

SFTL ACx, Tx[,Acy]

Registre à décaler

Registre de stockage du résultat

herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)

Modificateurs d'opérande

dbl	Access a true 32-bit memory operand
dual	Access a 32-bit memory operand for use as two independent 16-bit halves of the given operation
HI	Access upper 16 bits of the accumulator
high_byte	Access the high byte of the memory location
LO	Access lower 16 bits of the accumulator
low_byte	Access the low byte of the memory location
pair	Dual register access
rnd	Round
saturate	Saturate
uns	Unsigned operand (not used in MOV instructions)

exemple:

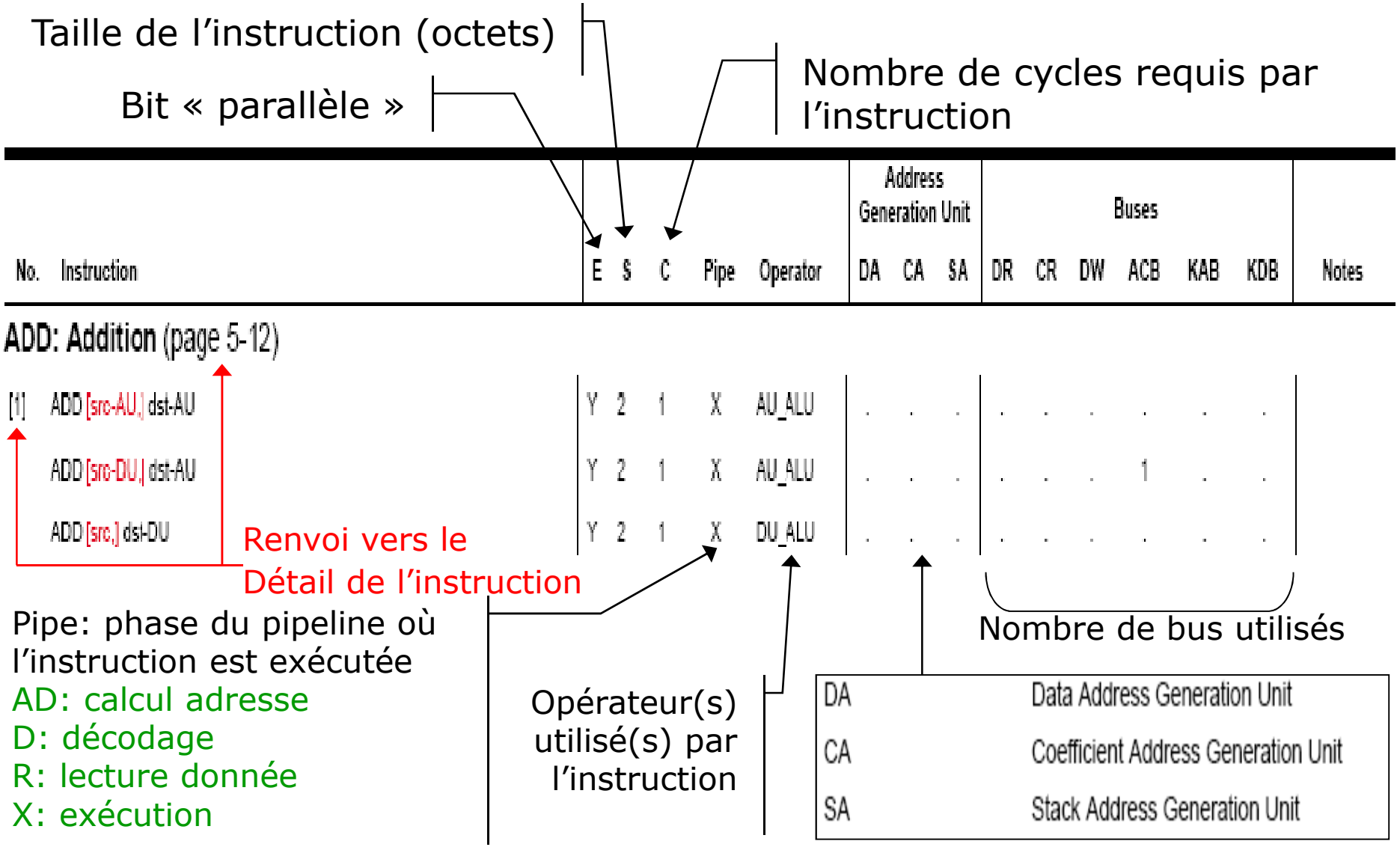
MOV[40] dbl(Lmem), ACx MOV40 dbl(*AR3-), AC0

AR3 est décrémenté de 2 après exécution

Lmem	Long-word single data memory access (32-bit data access).

herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)



herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)

Syntax Characteristics

No.	Syntax	Parallel Enable Bit	Size	Cycles	Pipeline
[1]	ADD [src,] dst	Yes	2	1	X

Opcode

0010 010E	FSSS FDDD

Operands dst, src

Description This instruction performs an addition operation between two registers:

```
dst = dst + src
```

- When the destination (dst) operand is an accumulator:
  - The operation is performed on 40 bits in the D-unit ALU.

Repeat This instruction can be repeated.

Example

Syntax	Description
ADD AC1, AC0	The content of AC1 is added to the content of AC0 and the result is stored in AC0.

herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)

Syntax Characteristics

No.	Syntax	Parallel Enable Bit	Size	Cycles	Pipeline
[1]	ADD [src,] dst	Yes	2	1	X

Table 1–1. Instruction Set Terms, Symbols, and Abbreviations

Symbol	Meaning
. . .	
dst	Destination accumulator (ACx), lower 16 bits of auxiliary register (ARx), or temporary register (Tx): AC0, AC1, AC2, AC3 AR0, AR1, AR2, AR3, AR4, AR5, AR6, AR7 T0, T1, T2, T3
. . .	
src	Source accumulator (ACx), lower 16 bits of auxiliary register (ARx), or temporary register (Tx): AC0, AC1, AC2, AC3 AR0, AR1, AR2, AR3, AR4, AR5, AR6, AR7 T0, T1, T2, T3
. . .	

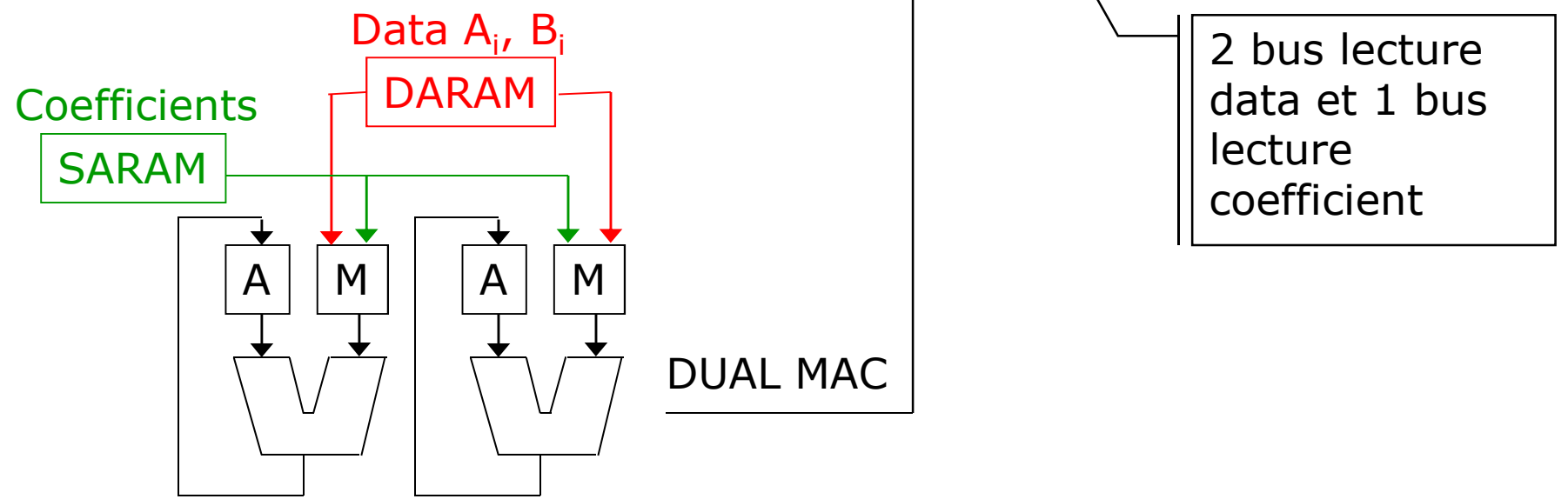
herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)

No.	Instructicn	E	S	C	Pipe	Operator	Address Generation Unit DA	CA	SA	Busses DR	CR	DW	ACB	KAB	KDB	Notes
MAC::MAC: Parallel Multiply and Accumulates (page 5-179)																
[1]	MAC[R][40] [uns()]Xmem[]), [uns()]Cmem[]), ACx :: MAC[R][40] [uns()]Ymem[]), [uns()]Cmem[]), ACy	N	4	1	X	DU_ALU	2	1	.	2	1	.	.	.	.	

1 cycle

2 bus lecture data et 1 bus lecture coefficient



herve.le-provost@cea.fr

TMS320C5510: mnémoniques (Texas Instruments SPRU374G)

Call Unconditionally

Syntax Characteristics

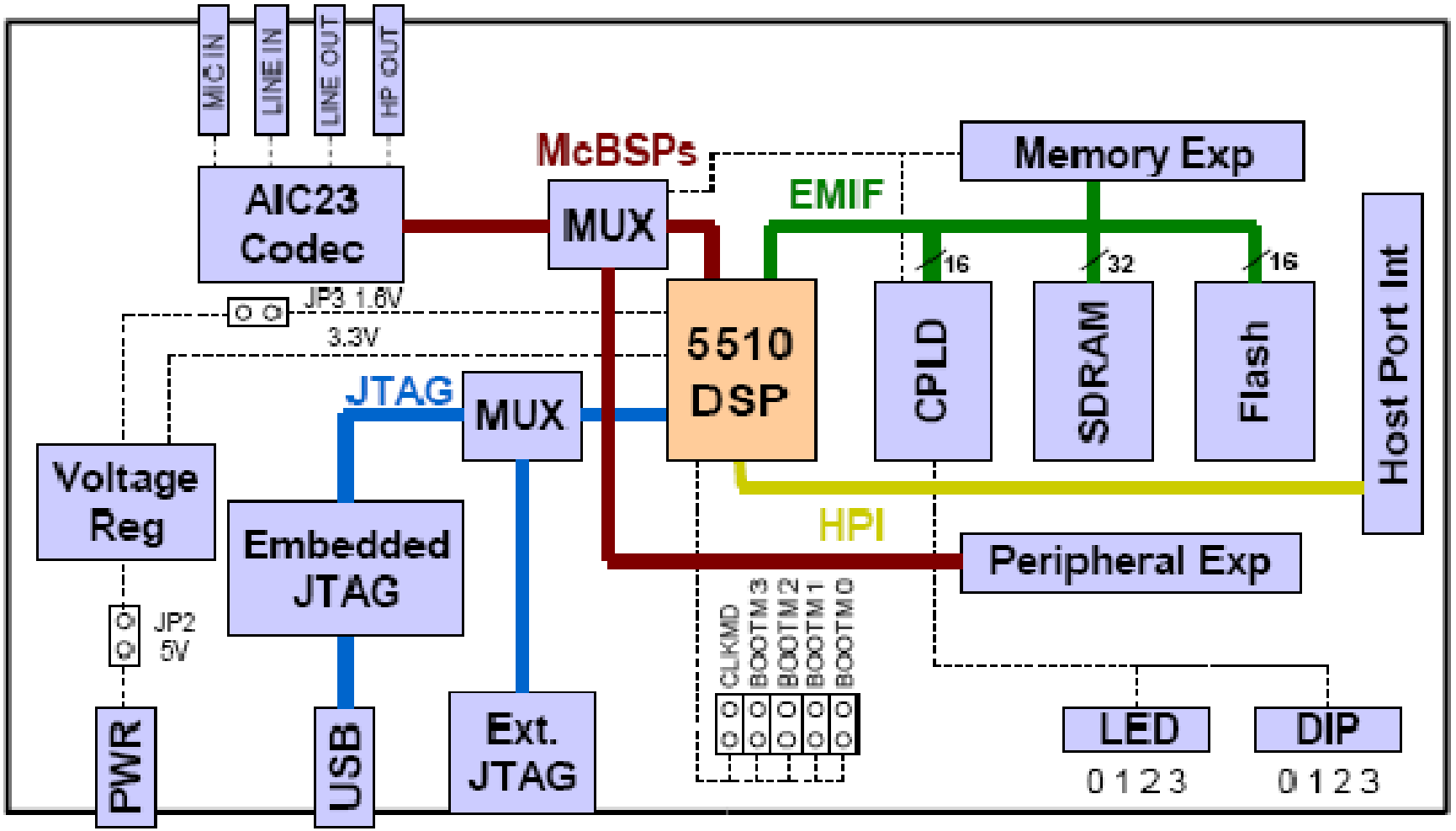
No.	Syntax	Parallel Enable Bit	Size	Cycles	Pipeline
[1]	CALL ACx	No	2	10	X

Opcode		1001	0010	xxxx	xxSS
Operands	ACx				

Pourquoi le nombre de cycles est-il de 10 ?

herve.le-provost@cea.fr

Carte DSK TMS320C5510 (source Texas Instruments Technical Reference)



herve.le-provost@cea.fr

Documentation DSP C5510

- Téléchargeable depuis le site Texas Instruments http://www.ti.com rubrique Products-Processors-Digital Signal Processing,
  - TMS320C55xx Low Power DSPs
    - User Guides :TMS320vc5510 (data Manual), spru312 (functional Overview), spru371f (CPU Reference Guide), spru374g (mnemonic Instruction Set), spru280h (assembly Language Tool User Guide)
    - Logiciel: Code Composer Studio Platinium development tool, évaluation gratuite de 120 jours
- Téléchargeable depuis le site du CEA
  - Ce cours
    http://irfu.cea.fr/Phocea/Membres/Cours/index.php?uid=leprovo

herve.le-provost@cea.fr

Installation du simulateur C5510/CCStudio v3.3

- Installation
  - Décompresser le fichier téléchargé depuis le site www.ti.com sprc119d.zip
  - Vérifier que vous êtes administrateur de votre PC
  - aller dans le répertoire Code_Composer_Studio_v3.3_Build_3.3.38.2_FET et lancer setup.exe
  - Vous pouvez garder les paramètres d'installation par défaut.
- Paramétrage
  - Lancer Setup CCStudio v3.3 (installé sur le bureau)
  - Sélectionner « C5510 Device Simulator» dans la liste « Available Factory Boards » et cliquez sur « add » puis « Save & Quit »
  - Lancer CCStudio V3.3, le programme est prêt à être utilisé. Vous pouvez créer un projet
- Programmes Source C
  - Dans l'onglet Project->Build Options->Compiler->Basic, ajouter l'option –ml sur la ligne ou apparait « Proj_Dir »
  - Dans l'onglet Project->Build Options->Linker->Libraries, ajouter la librarairie rts55x.lib (Incl.Libraries)

herve.le-provost@cea.fr