heig-vd
Haute Ecole d'Ingénierie
du Canton de Vaud

RéDS
heig-vd

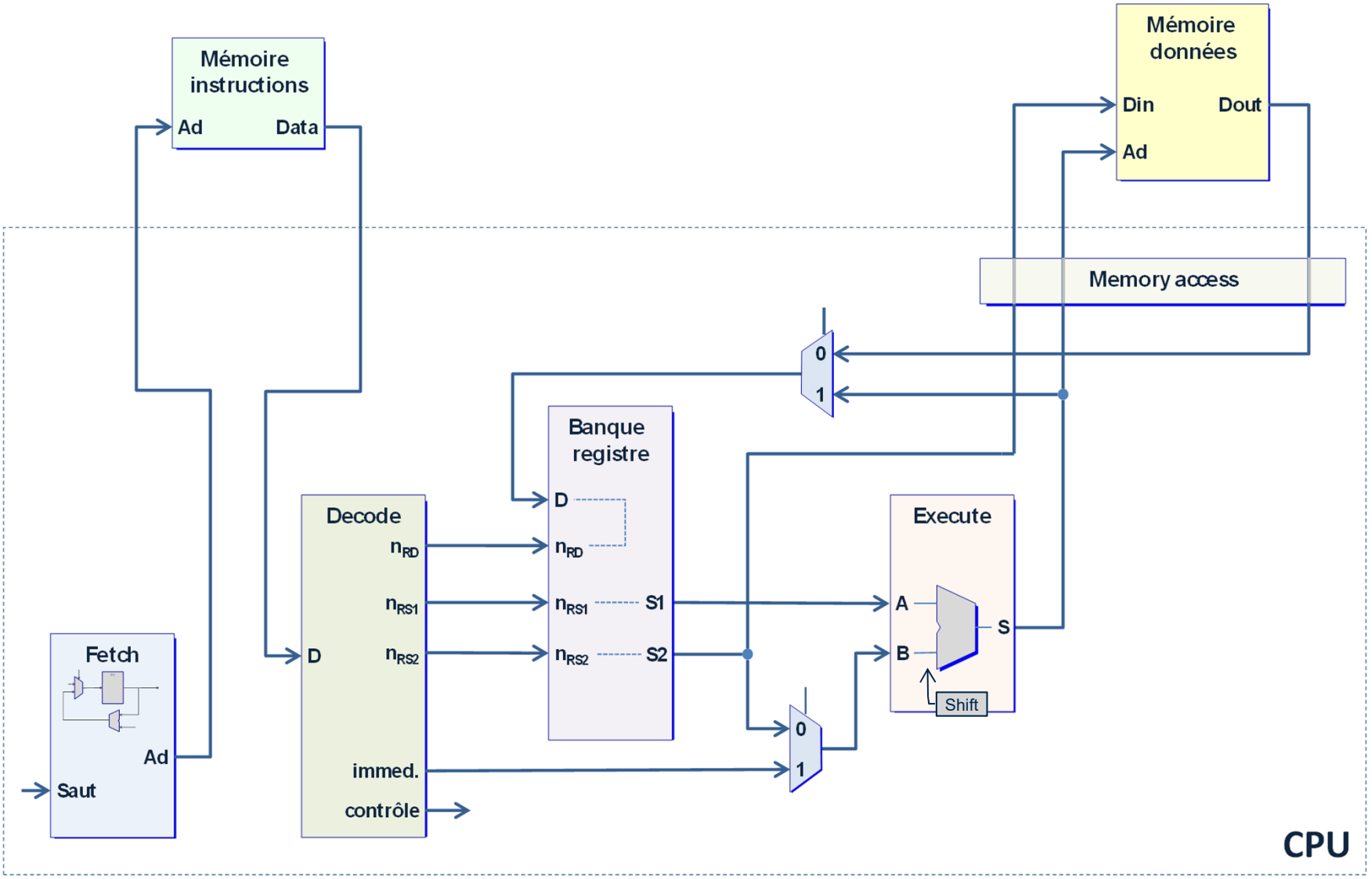
ARO2

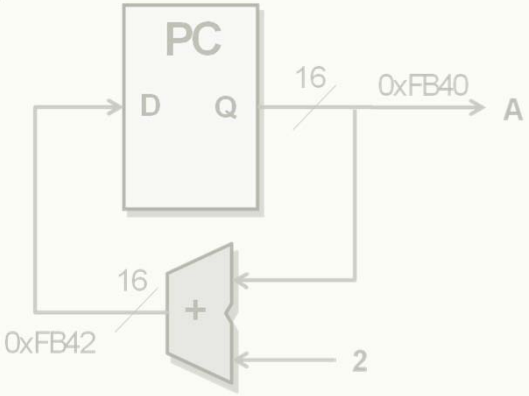
Micro-architecture d'un processeur

Basé sur le cours du prof. E. Sanchez
et le cours ASP du prof. M.Starkier

Romuald Mosqueron

Microarchitecture d'un processeur de type ARM





ARO2

Micro-architecture d'un processeur
La partie FETCH

Basé sur le cours du prof. E. Sanchez
et le cours ASP du prof. M.Starkier

Michel Starkier

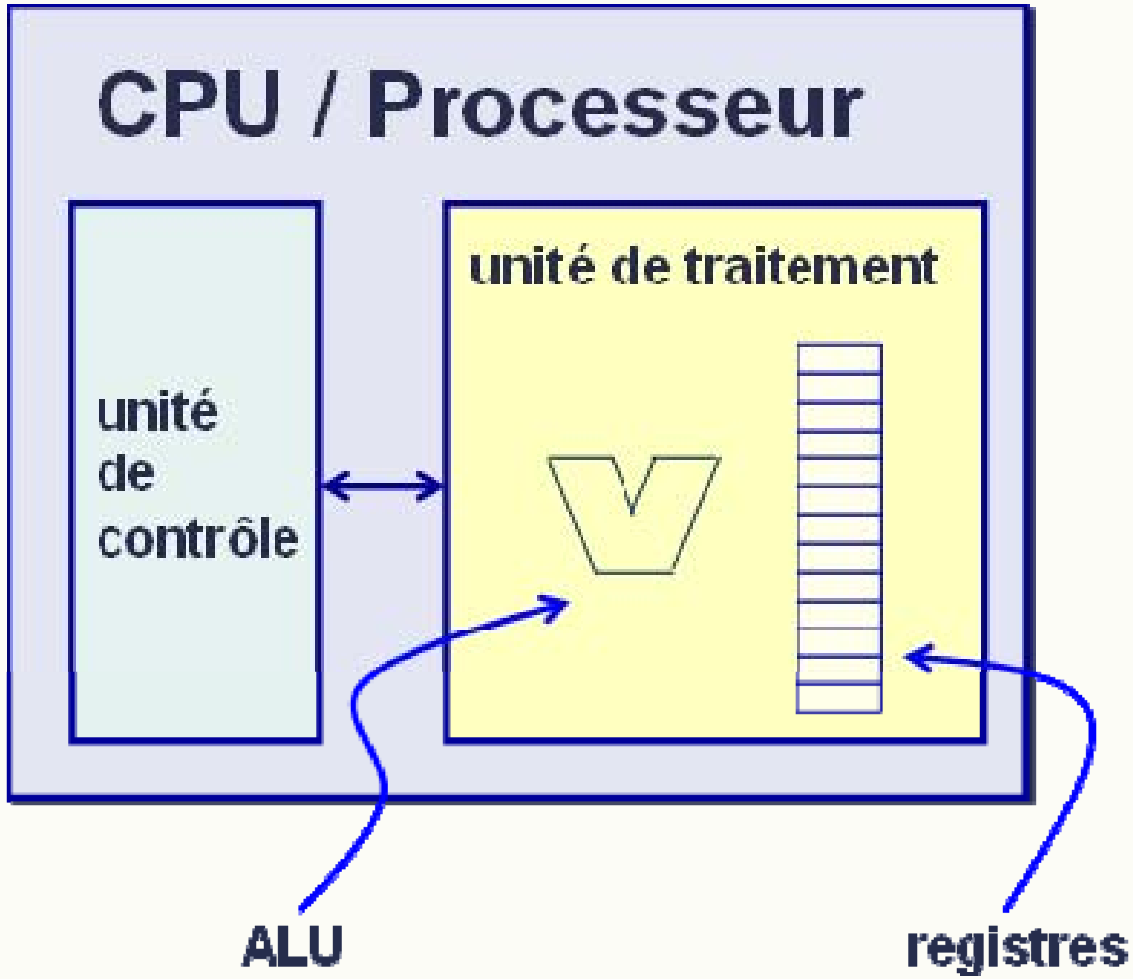
Qu'est-ce que le fetch ?

- «Fetch» signifie «rapporter, faire amener» Ex :
  Can you fetch the children from school tonight?

- Fetch (informatique) signifie : charger une instruction dans le CPU à partir de la mémoire d'instruction

- L'instruction est en général chargée dans un registre nommé Instruction Register (IR)

Architecture simple

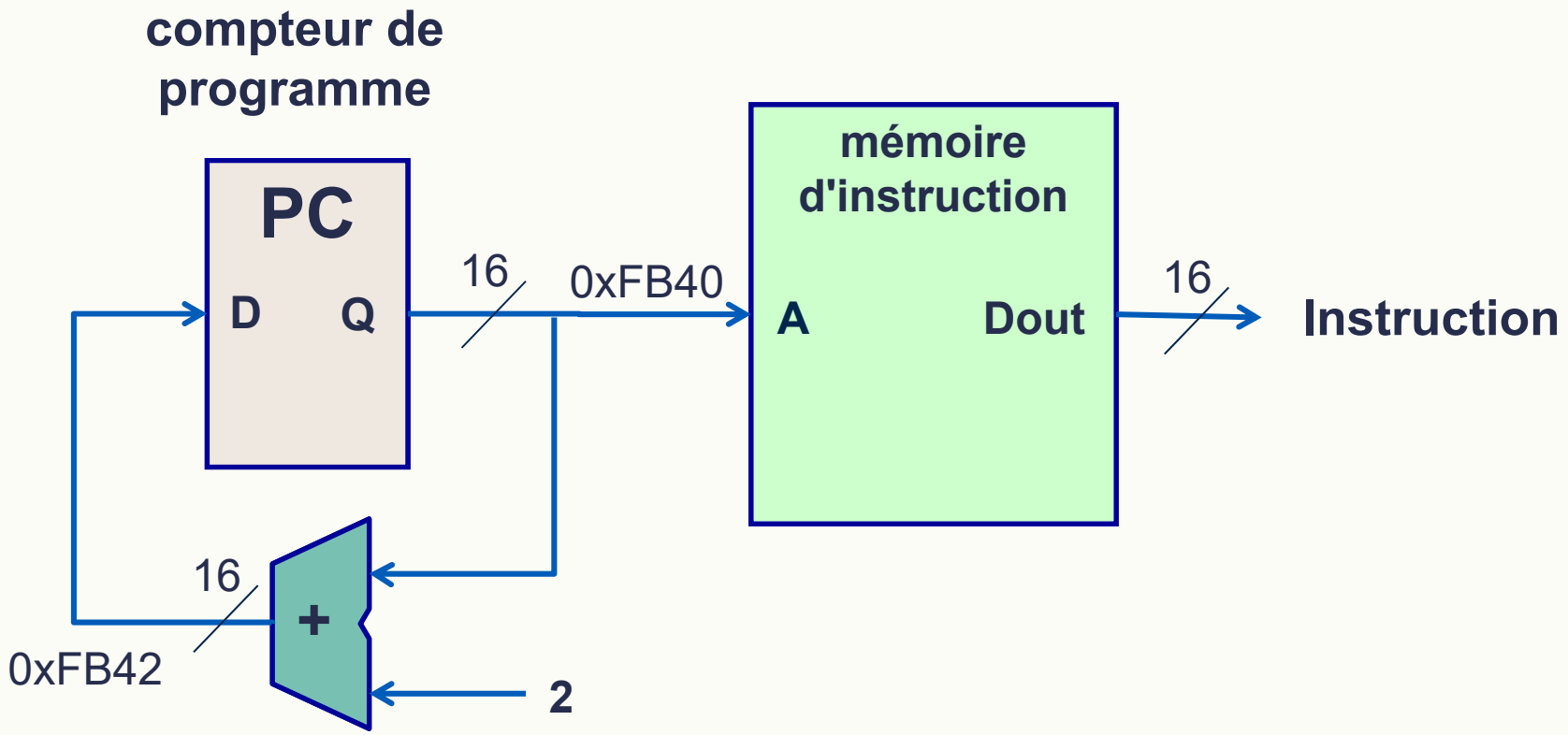


Le bloc FETCH

- Le bloc FETCH fait partie de l'unité de contrôle et contient le compteur de programme (PC: Program Counter)
- Le PC contient l'adresse de l'instruction courante
- Le PC est incrémenté à chaque instruction (sauf saut)
- La valeur de l'incrément correspond au nombre de bytes de l'instruction :
  - 1 pour un code d'instruction 8 bits
  - 2 pour un code d'instruction 16 bits
  - 4 pour un code d'instruction 32 bits

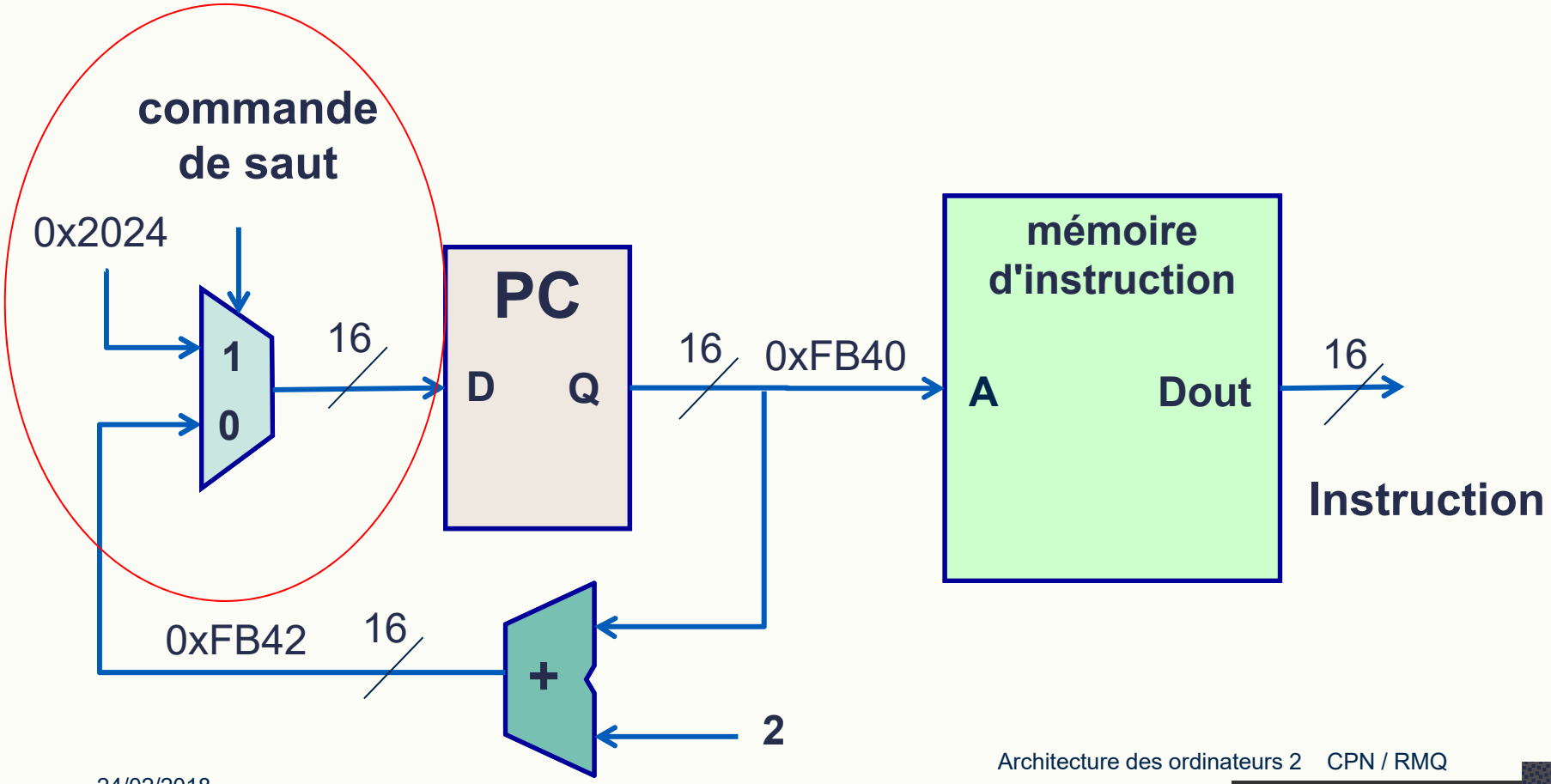
Incrémentation du PC
Lecture des instructions

- **Exemple avec un code d'instruction de 16 bits et un bus d'adresse de 16 bits**



Modification de la valeur du PC

- La valeur du PC est modifiée en cas de saut dans le programme ou à l'initialisation

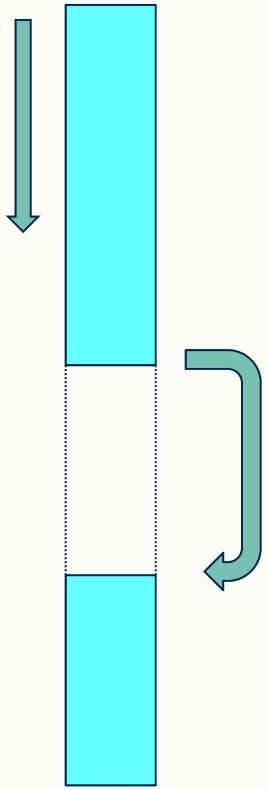


Cours AR02

SAUTS, APPELS DE FONCTIONS ET INTERRUPTIONS

Sauts dans un programme

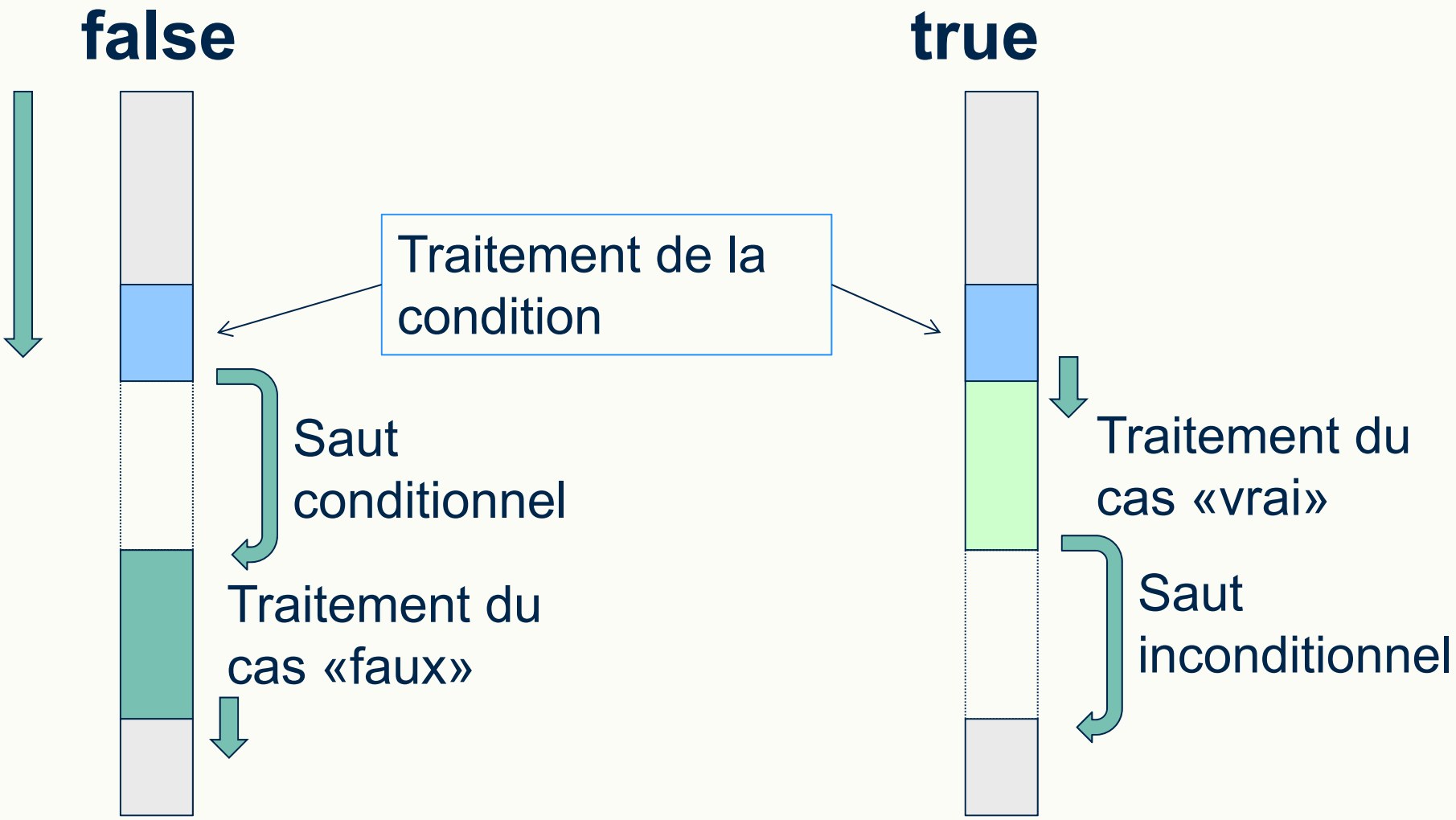
- Déclenchés par une instruction
  - Sauts inconditionnels
  - Sauts conditionnels (= , !=, >, < ....)
  - Appels de fonctions
- Déclenchés matériellement
  - Interruptions
  - Exceptions



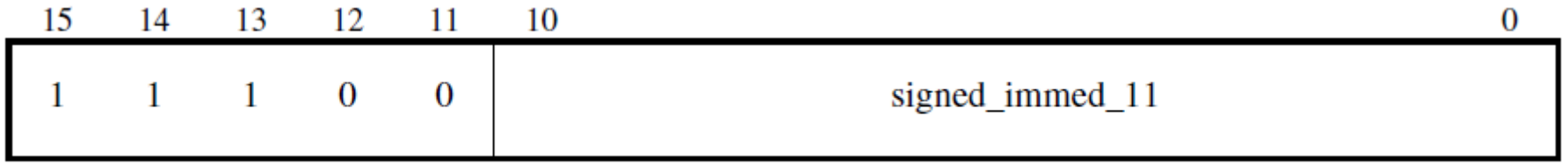
Sauts dans un programme

- Déroulement d'un programme nécessite:
  - Saut inconditionnel
    - Exemple: retour au début d'un programme
  - Saut conditionnel car le déroulement d'un programme est souvent dépendant de conditions
    - Exemple: exécution d'une boucle
- Instructions correspondantes:
  - Instruction de saut inconditionnel (branch)
  - Instruction de saut conditionnel (branch {cond})
    - Permet de tester différentes conditions
      = , !=, >, < ....

If *condition* then … else …

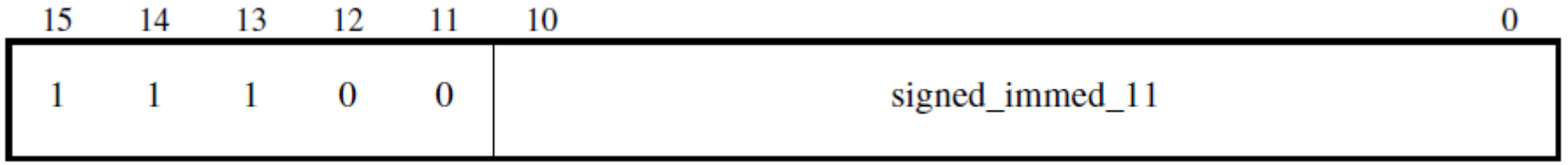


Sauts inconditionnels : Instruction ARM branch B



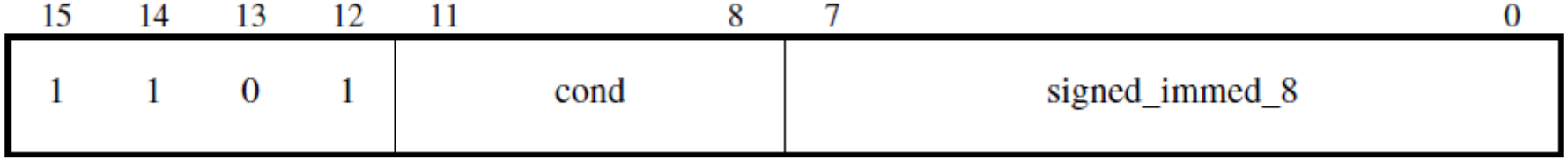
- syntax : B <target_address> (Label=target address)
- adresse relative par rapport au PC (offset)
- Adr = PC + extension_nb_bits_adr($offset_{11}$ x 2) + 4
  - Offset par mots 16 bits => à multiplier par 2 pour avoir un offset en bytes
  - Plage possible: +2050 à -2044
  - Ajout de +4 sera expliqué par la suite dans le cours

Sauts inconditionnels : Instruction ARM branch B



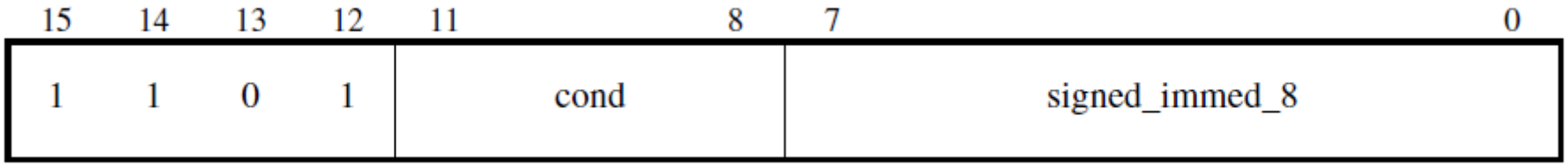

```
ADR_SAUT_2 :
@ instructions (2 ou 3)
……
B ADR_SAUT_1 @ saut inconditionnel en avant
.org 0x30
ADR_SAUT_1:
@ instructions (2 ou 3)
……
B ADR_SAUT_2 @ saut inconditionnel en arrière
```

Sauts conditionnels : Instructions ARM branch Bxx



- syntax : B<cond> <target_address>
- adresse relative par rapport au PC: offset sur 8 bits
- condition de saut (EQ, NE, ...)
- Adr = PC + extension_16bits($offset_8$ x 2) + 4
  - Offset par mots 16 bits => à multiplier par 2 pour avoir un offset en bytes
  - Plage possible: +258 à -252
  - Ajout de +4 sera expliqué par la suite dans le cours

Sauts conditionnels : Instructions ARM branch Bxx



Code	Suffix	Description	Flags
0000	EQ	Equal / equals zero	Z
0001	NE	Not equal	!Z
0010	CS / HS	Carry set / unsigned higher or same	C
0011	CC / LO	Carry clear / unsigned lower	!C
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	!N
0110	VS	Overflow	V
0111	VC	No overflow	!V
1000	HI	Unsigned higher	C and !Z
1001	LS	Unsigned lower or same	!C or Z
1010	GE	Signed greater than or equal	N == V
1011	LT	Signed less than	N != V
1100	GT	Signed greater than	!Z and (N == V)
1101	LE	Signed less than or equal	Z or (N != V)
1110	AL	Always (default)	any

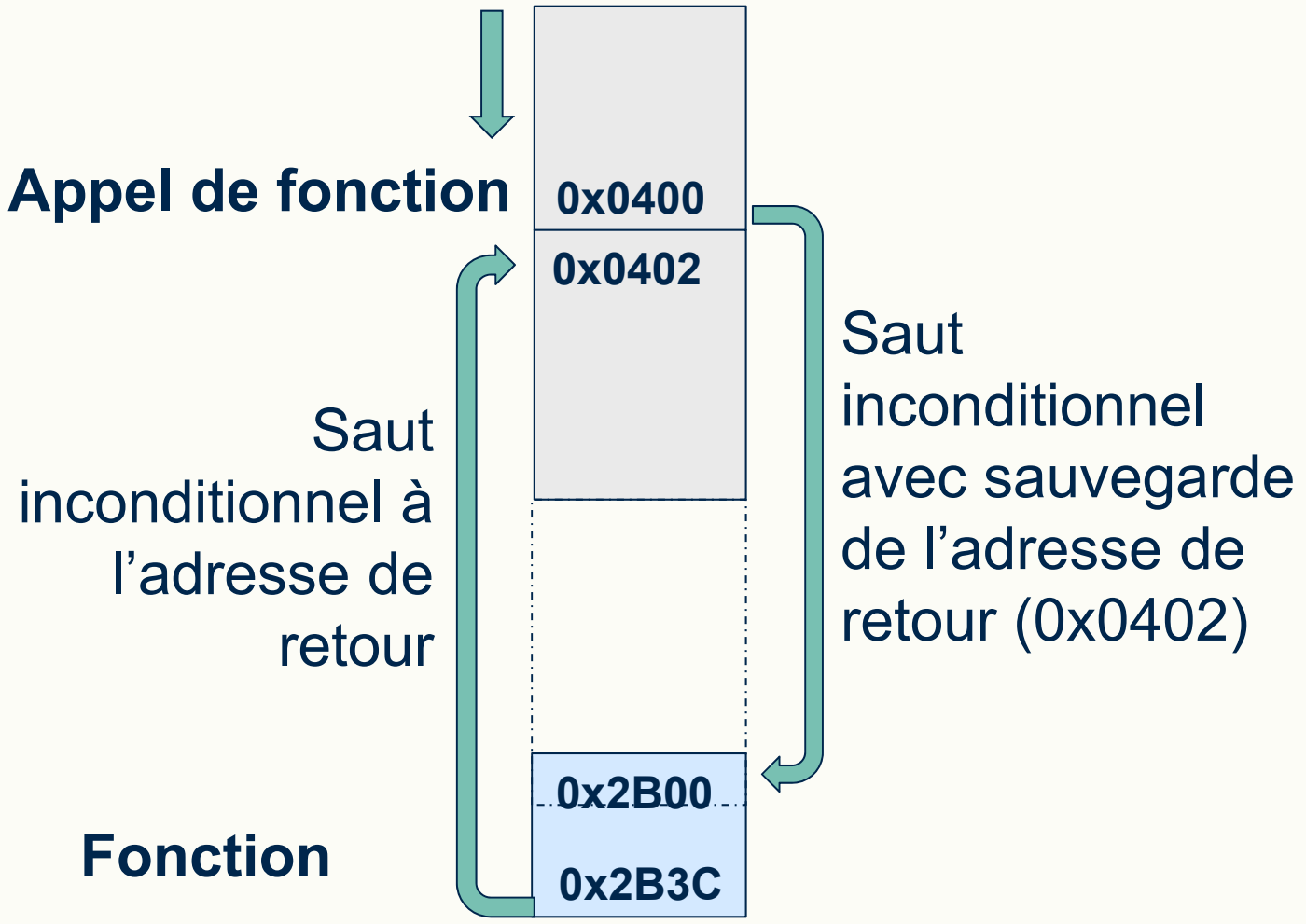
Instructions ARM : Conditions

- Les conditions de saut sont validées à partir de 4 bits du registre de status (CPSR)
- CPSR : Current Program Status Register
  - Bits de condition (N Z C V)
    - N = Résultat d'une opération de l'ALU négatif
    - Z = Résultat d'une opération de l'ALU nul
    - C = Carry(report) provenant de l'ALU
    - V = Overflow (dépassement) d'une opération de l'ALU

Sous-programme (fonction)

- Gestion de la complexité d'un programme par un découpage hiérarchique
- Fonctionnalités nécessaires:
  - Instruction permettant d'aller dans un sous-programme
    - L'instruction doit permettre de sauter au début du sous-programme ET elle doit sauver l'adresse actuelle + 1 pour le retour
    - L'instruction est souvent nommée CALL
  - A la fin du sous-programme:
    - Il faut retourner a l'adresse qui a été sauvée lors de l'instruction de saut

Appel de fonction
Exemple

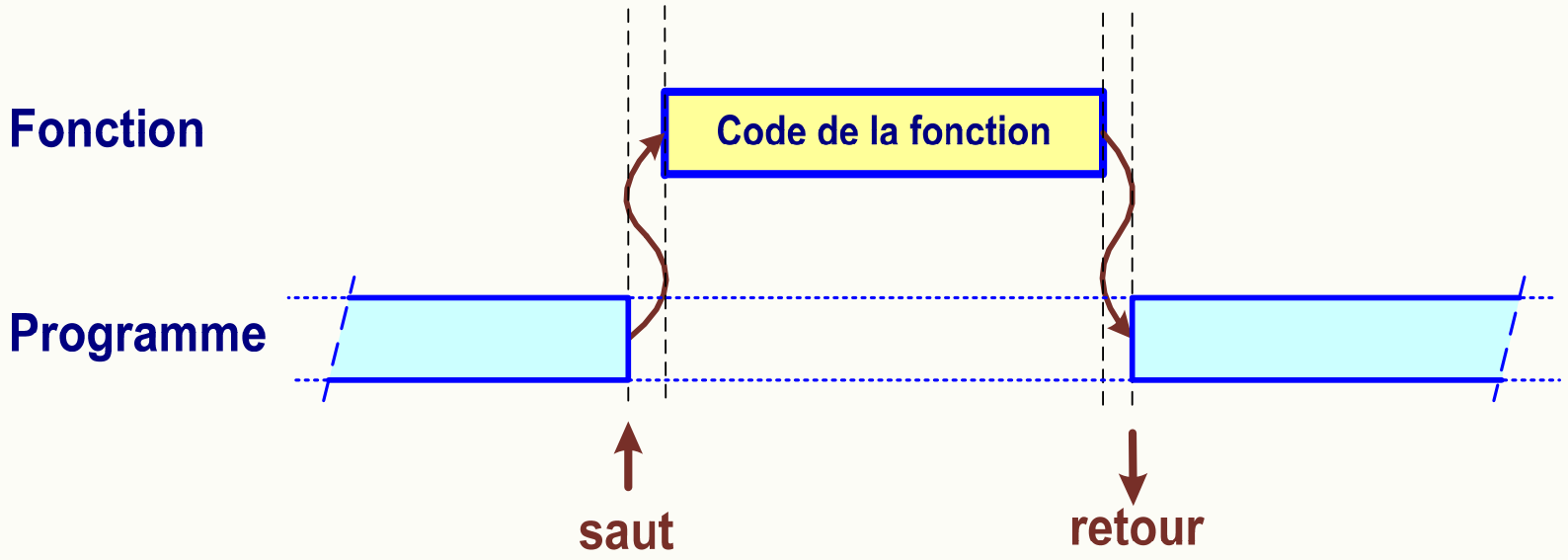


Appels de fonctions: Instructions ARM branch and link

- syntax : BL <target_address>
- sauve l'adresse de retour dans un registre spécial «Link Register» (LR)
- $offset_{11}$: adresse relative par rapport au PC
- retour par l'instruction MOV PC, LR
- nécessite 2 instructions pour avoir un offset sur 22 bits (+/- 4 MBytes de saut)

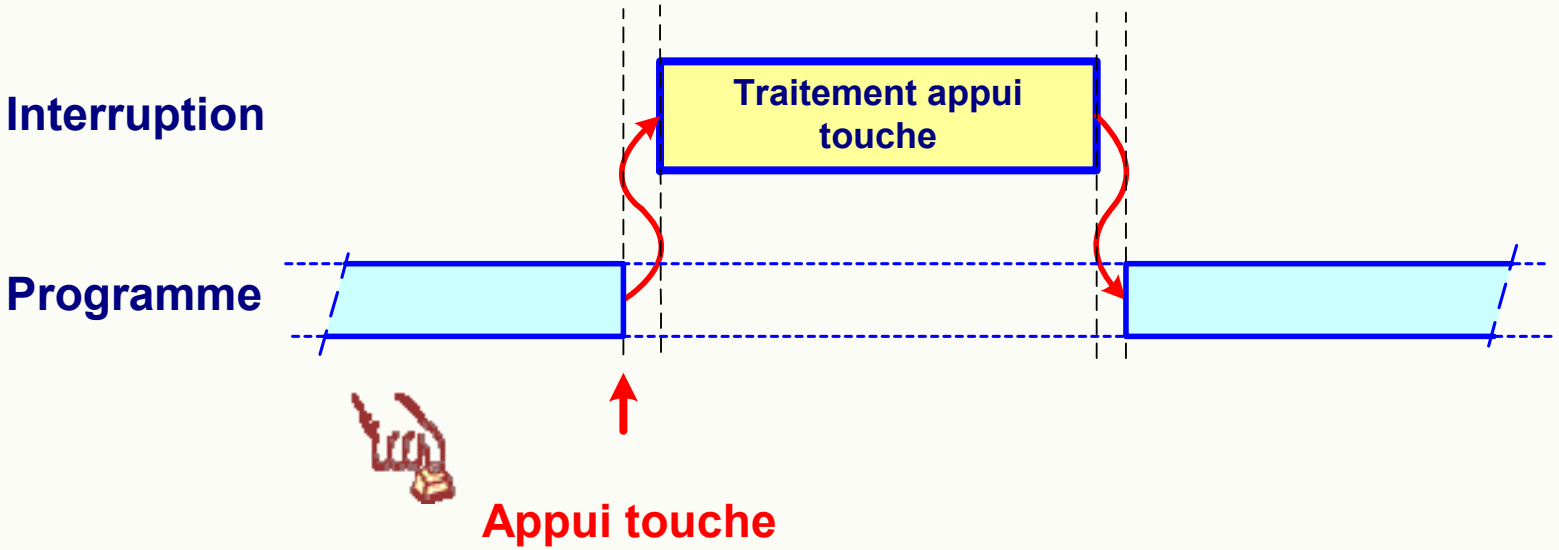
Appel de fonction et interruptions(1)

- **Appel de fonction: une instruction du programme déclenche un saut vers le code de la fonction.**
- **L'instruction de retour (à la fin du code de la fonction) permet de revenir au point de départ dans le code du programme.**



Appel de fonction et interruptions (2)

- **Interruption : un événement matériel déclenche un saut vers le code de la fonction.**
- **L'instruction de retour (à la fin du code de la fonction) permet de revenir au point de départ dans le code du programme.**



Cours ARO2

INTERRUPTIONS ET EXCEPTIONS

Sources d'interruptions

- **Périphériques :**

  timer, clavier, entrées /sorties (port série, parallèle, USB, disques, ….)

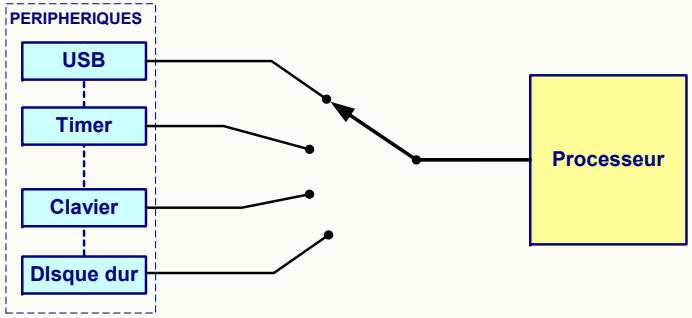
- **Erreurs systèmes (exception traps) :**

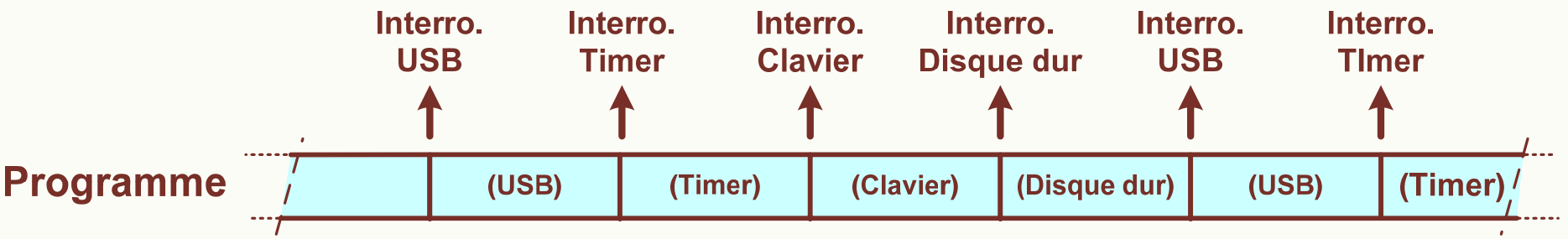
  erreurs mémoires, division par zéro, …

- **Interruption processeur (IPI) :**

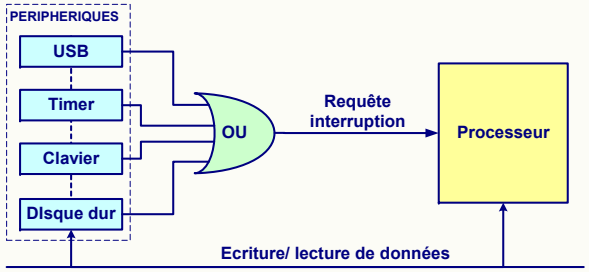
  interruption par un autre processeur (système multi-processeurs)

Scrutation (mode sans interruption)

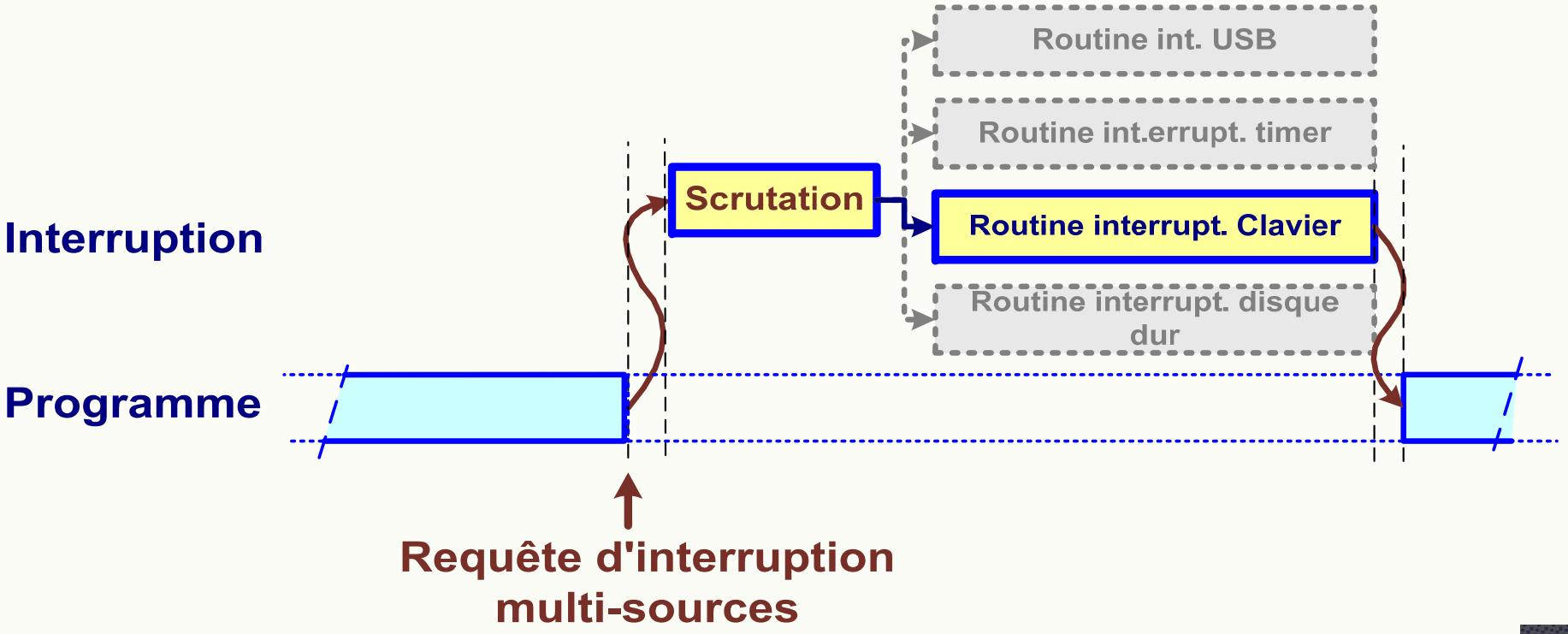




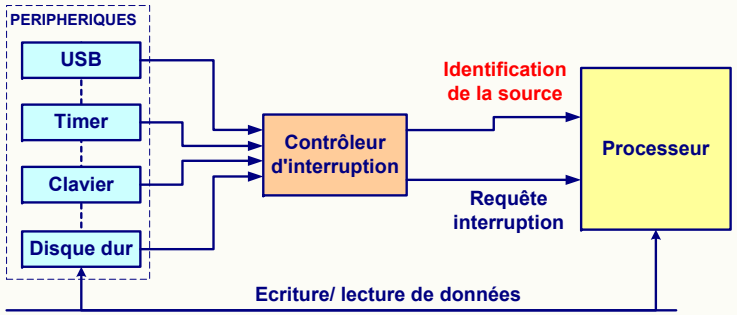
Interruption multi-sources



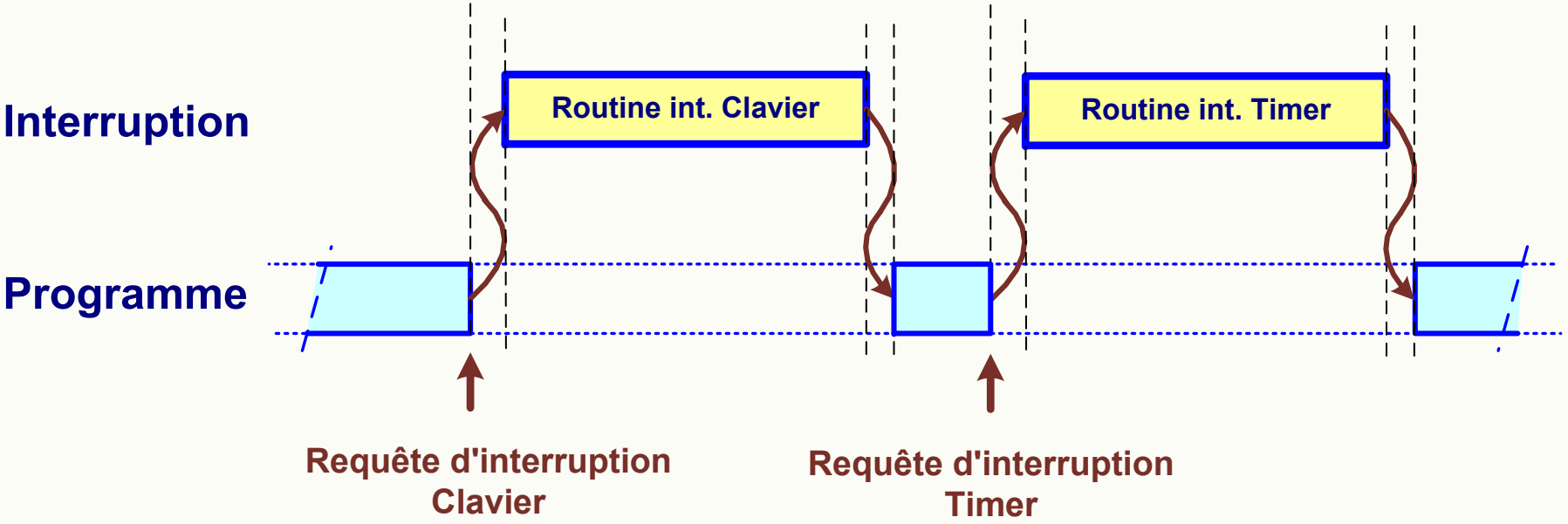
Sans identification de la source



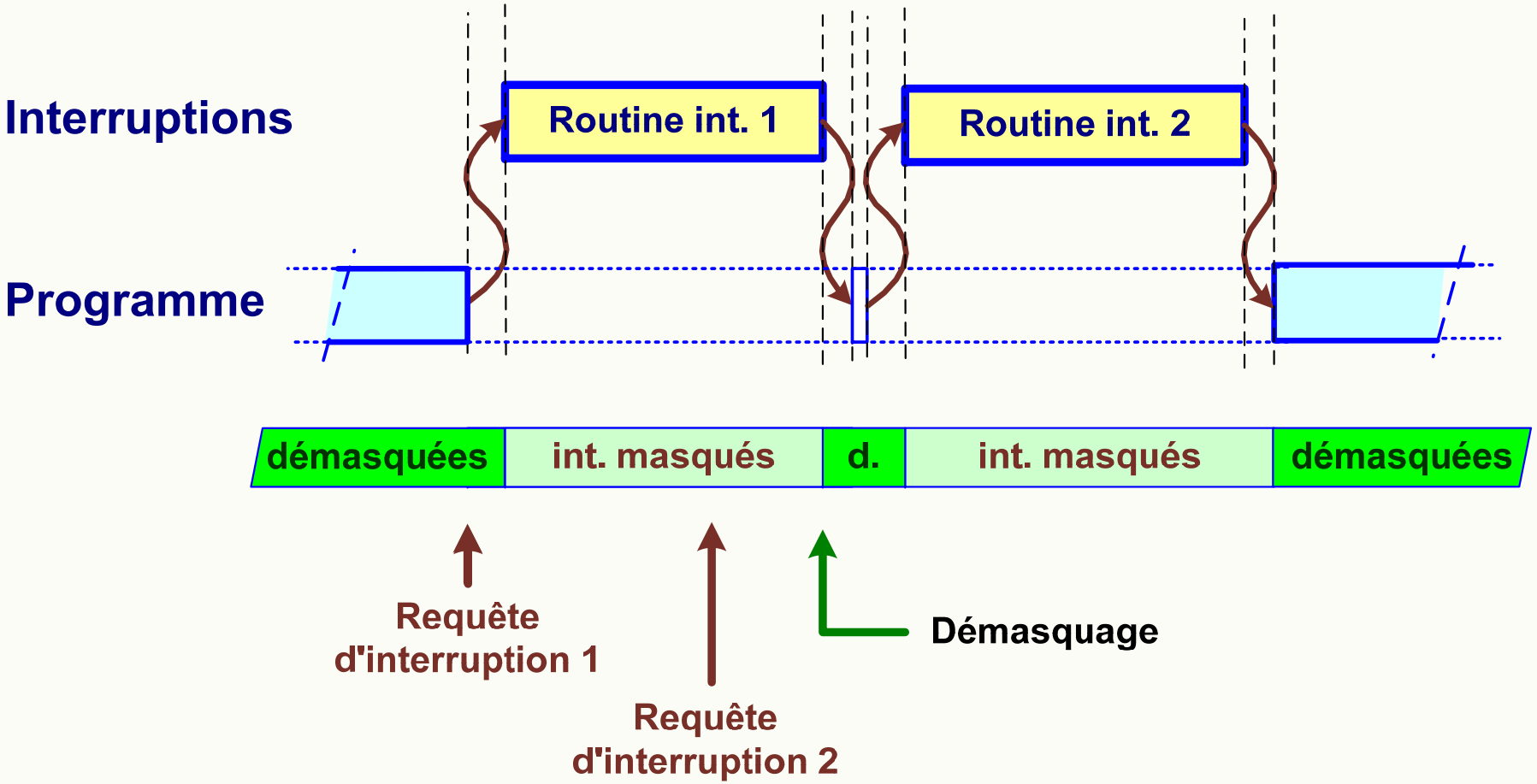
Interruptions multi-sources



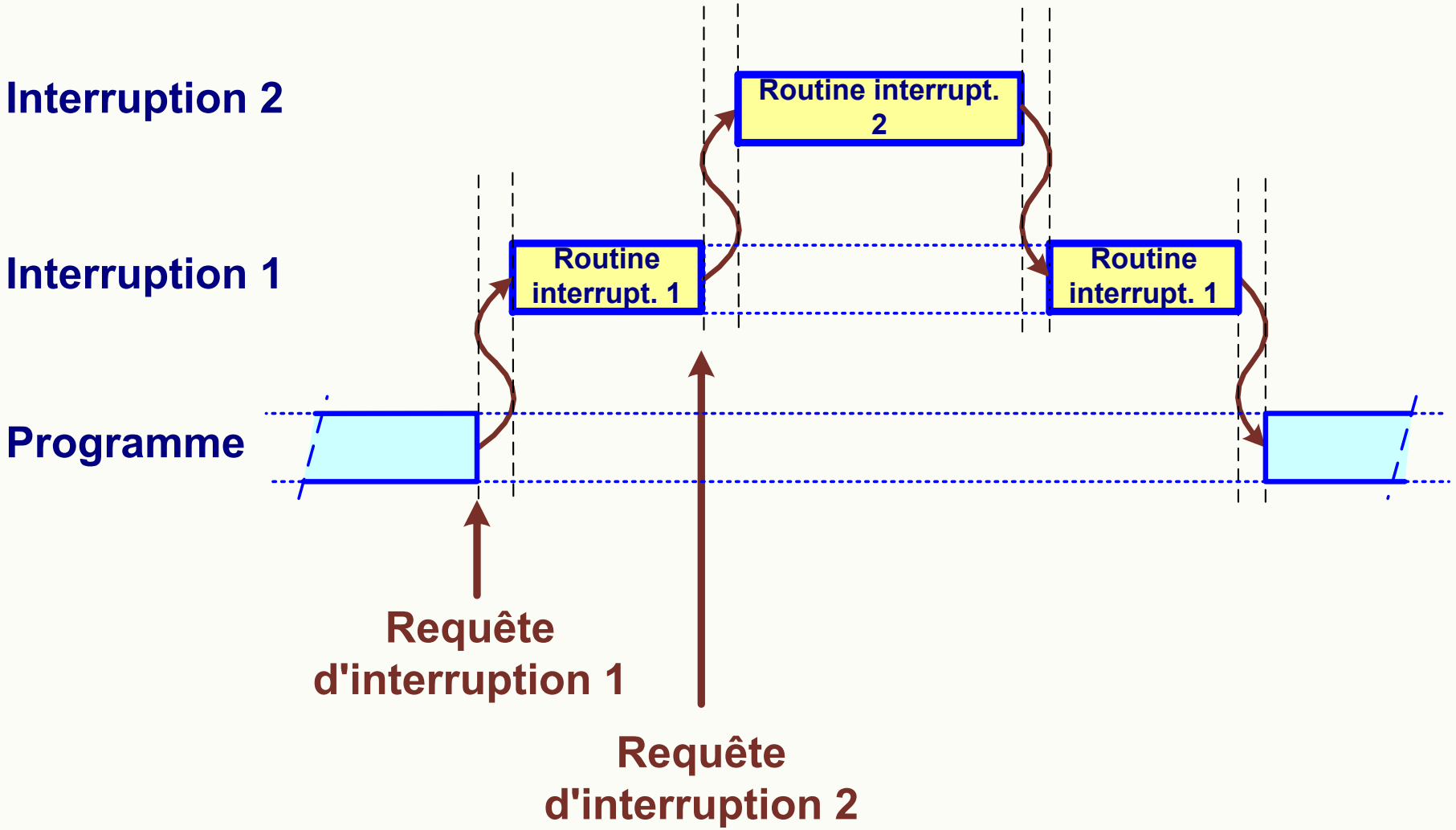
Avec identification de la source



Interruptions chaînées

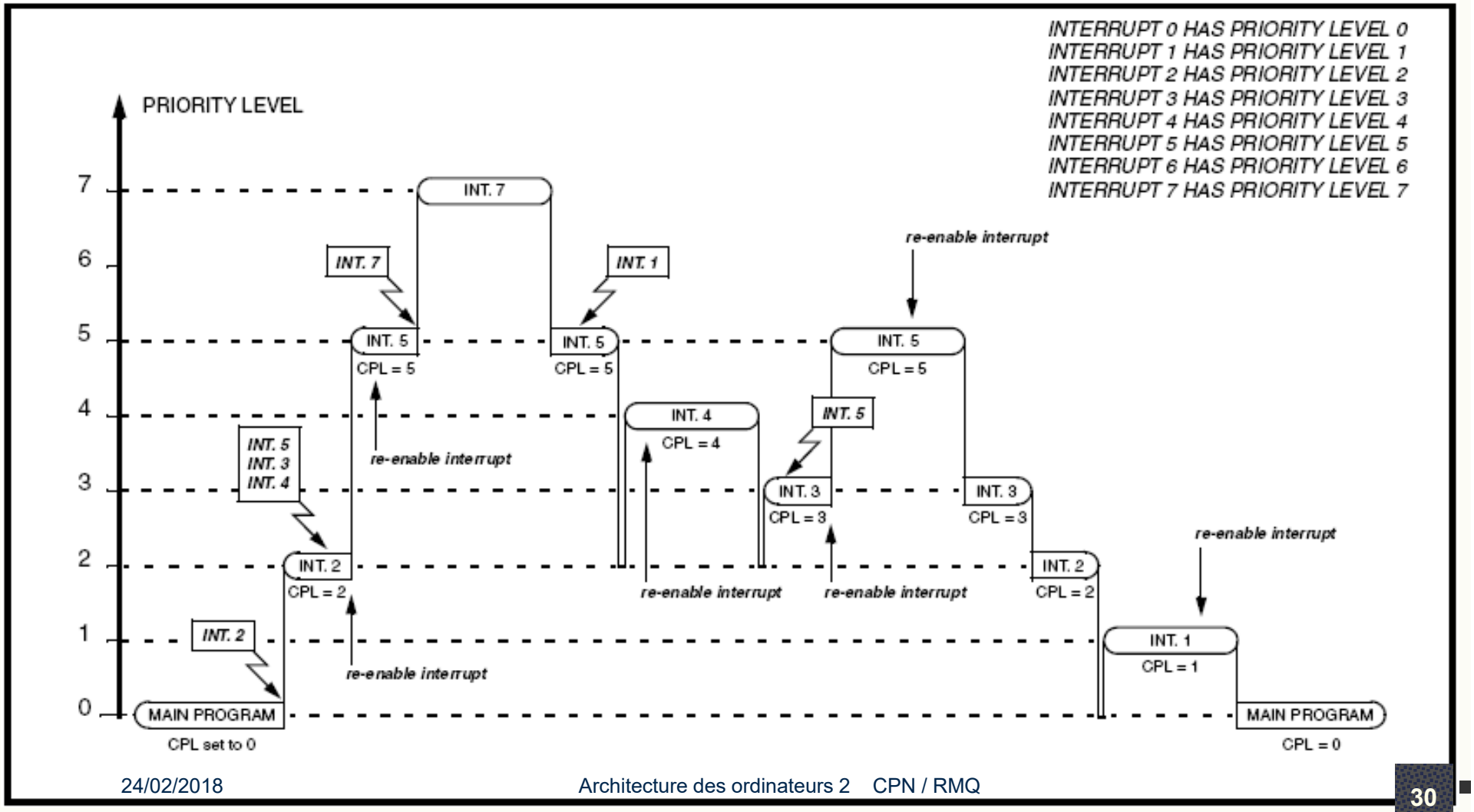


Interruptions imbriquées



Priorités - Interruptions imbriquées

Exemple



Vecteurs d'interruption

- Contrôleur d'interruption

=> nombre = identifiant unique d'une source d'interruption

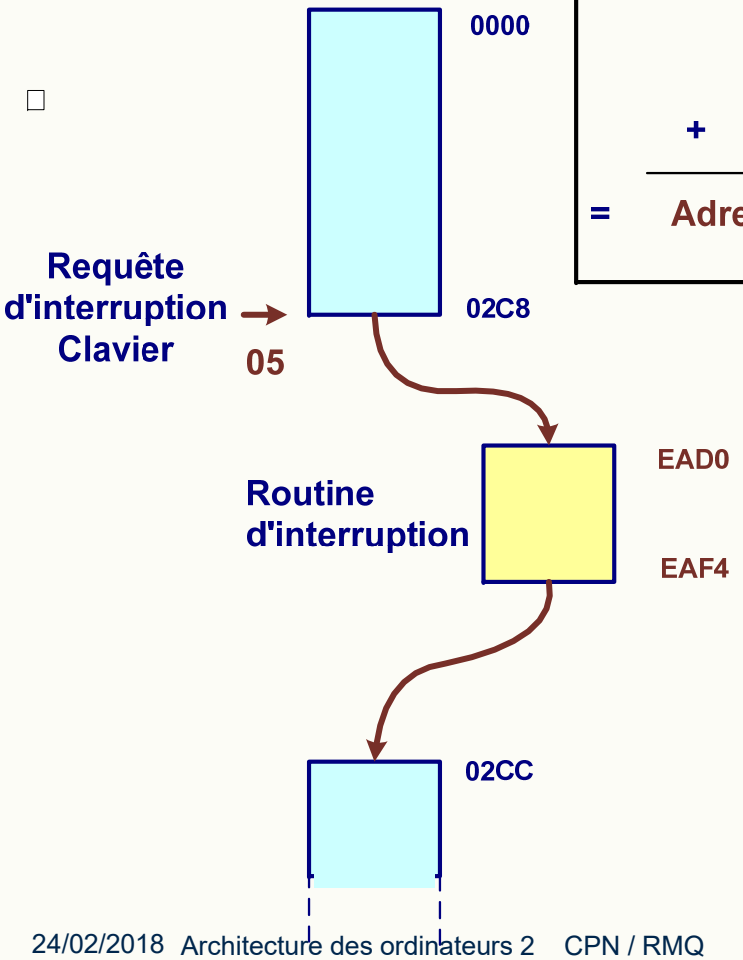
- Nombre ou adresse de la routine d'interruption ou instruction de saut

= vecteur d'interruption

- L'association entre les identifiants et les vecteurs d'interruption se fait par une table des vecteurs d'interruption (mémoire programme)

Vecteurs d'interruption

- La position du vecteur d'interruption dans la table correspond à l'identifiant de la source => 5 pour le clavier



	Adresse _Base _Table_des_vecteurs	FC40
+	Identifiant_interruption_clavier	05 x 4
=	Adresse_Vecteur_interruption_clavier	FC54

Table des vecteurs d'interuption

Horloge Système	E100h	FC40 = Adresse_Base
Port série	E200h	FC44
USB	F100h	FC48
Timer	E5D0h	FC4C
Afficheur	E750h	FC50
Clavier	EAD0h	FC54 =Adresse_Base + (05 x 4)
Joystick	EB00h	FC58
DIsque dur	F000h	FC5C

Vecteurs d'interruption

Exception (Interruption)	Mode	Reg.	Priorité	Adresse du vecteur
Reset	Supervisor	_svc	1	0x00000000
Undefined instruction	Undefined	_und	6	0x00000004
Software interrupt	Supervisor	_swc	6	0x00000008
Prefetch Abort	Abort	_abt	5	0x0000000C
Data Abort	Abort	_abt	2	0x00000010
(not assigned)				0x00000014
Interrupt	IRQ	_irq	4	0x00000018
Fast interrupt	FIQ	_fiq	3	0x0000001C