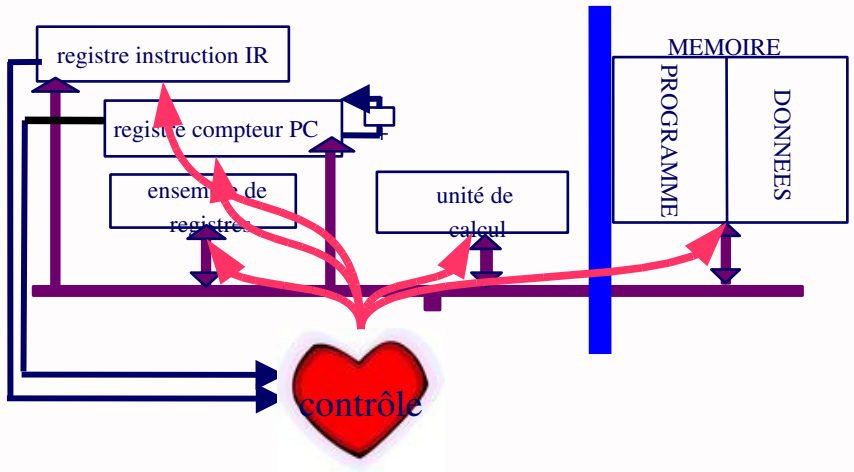
Partie contrôle



La partie contrôle est une ***machine d'états finis*** qui séquence les ***micro-instructions*** afin de réaliser une instruction

Partie contrôle

Chaque étape du cycle d'exécution du processeur correspond à une ou plusieurs opérations élémentaires du chemin de données, exécutables en un top d'horloge.

Le rôle de la partie contrôle est de *séquencer* ces opérations.

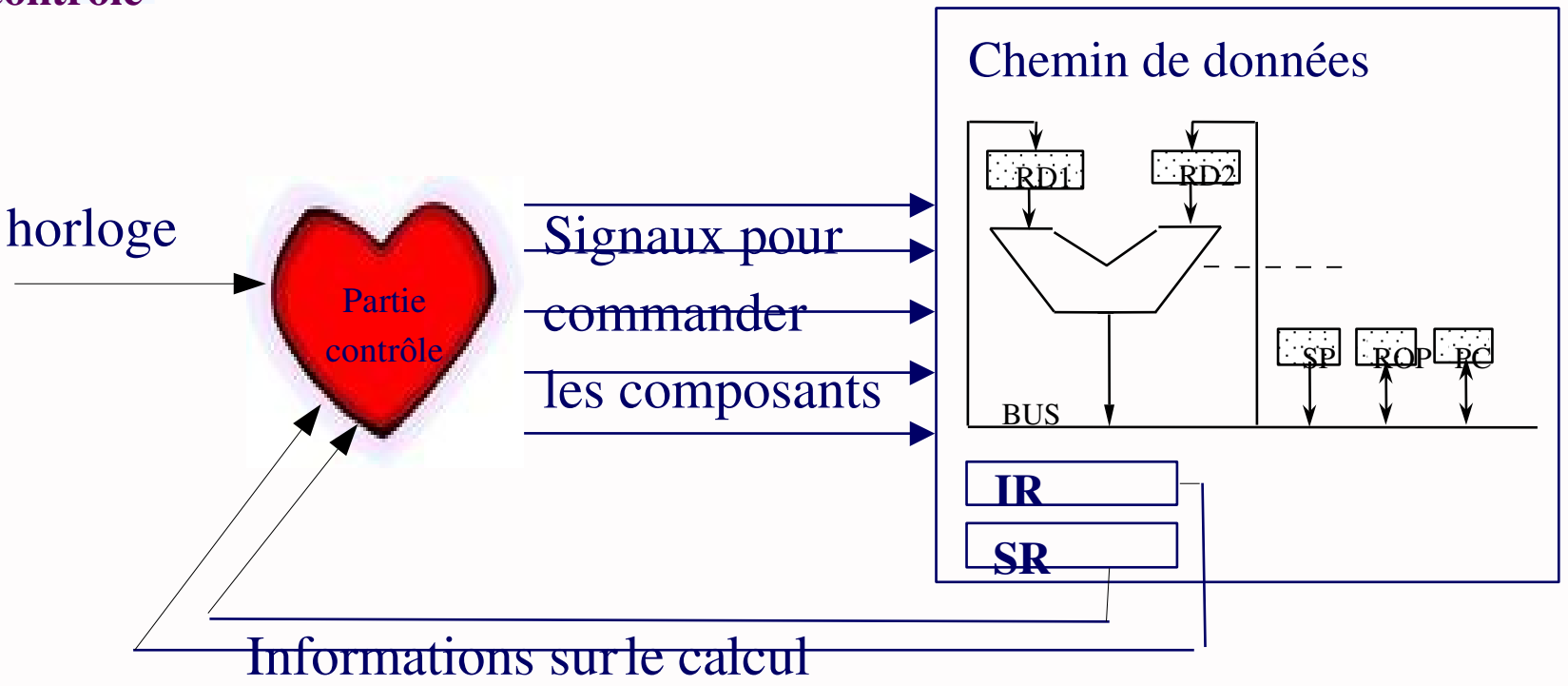
La partie contrôle est une MACHINE D'ÉTATS FINIS qui séquence les MICRO-INSTRUCTIONS afin de réaliser une instruction

- Micro-instruction : opération élémentaire exécutable sur le chemin de données en un top d'horloge
- Machine d'état fini (Finite State Machine FSM) :un circuit séquentiel synchrone

Pentium 4 fréquence de 3 Ghz

≈ 3 milliards de micro-instructions par seconde !!!

Partie contrôle



Sur chaque top d'horloge, la partie contrôle reçoit des infomations du chemin de données et envoie au chemin de données les signaux nécessaires pour éxécuter l'étape de calcul suivante

1. Micro-instruction

Opérations élémentaires exécutables en un cycle d'horloge processeur

- **Transfert de données d'un registre à l'autre**
- **Calculs dans l'ALU**
- **Lecture / écriture dans le banc de registres**
- **Modification des broches mémoire pour accès en lecture /écriture**

Partie contrôle

Les micro-instructions sont commandées par la partie contrôle en activant les signaux de contrôle des composants du chemin de données

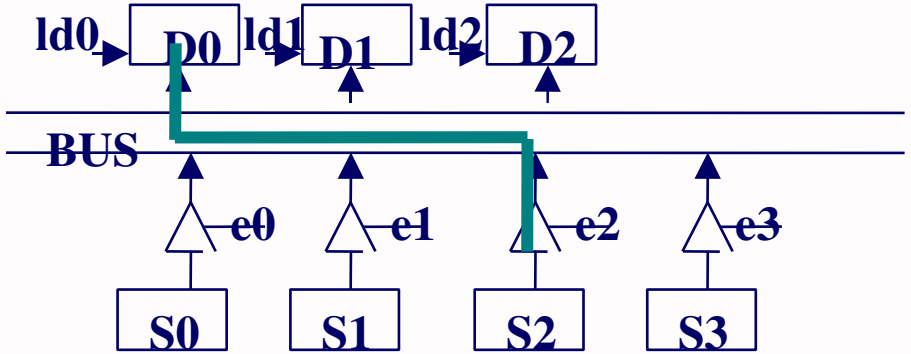
- **Transfert de données d'un registre à l'autre**
  - *Signaux de chargement des registres, signaux de gestion du bus*
- **Calculs dans l'ALU**
  - *Signaux de commande de l'ALU*
- **Lecture / écriture dans la RAM**
  - *Signaux de lecture/écriture, chip select*
- **Modification des broches mémoire pour accès en lecture /écriture**
  - *Signaux de lecture/écriture, chip select*

On confond la micro-instruction (les signaux) et les effets de la micro-instruction (les transferts mis en jeu).

- Ce qui se fait en parallèle dépend de l'organisation du chemin de données (un ou plusieurs bus, connexion directe entre registres, une ou plusieurs unités de calcul, …)

- Codage des micro-instructions: mot binaire constitué de l'ensemble des signaux à émettre sur le chemin de données pour exécuter la micro-instruction (1 bit pour chaque signal du chemin de données)

Exemple:

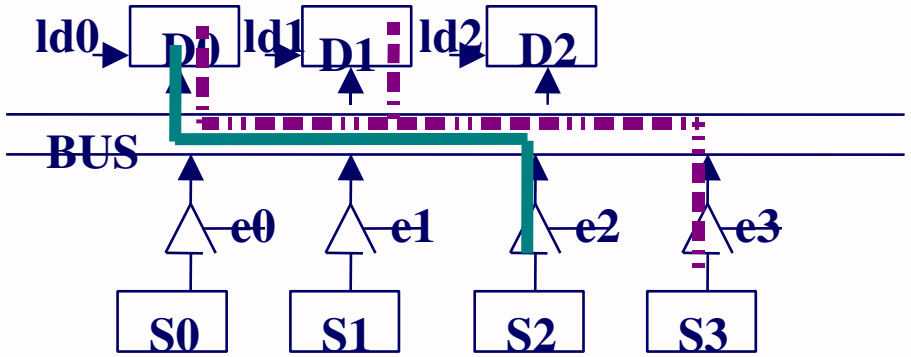


ld_i : signal load du registre i

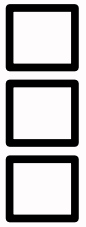
e_i : signal pour rendre la porte 3-états si passante

	ld2	ld1	ld0	e3	e2	e1	e0
D0 <- S2 codé par :	0	0	1	0	1	0	0

Exemple:



		ld2	ld1	ld0	e3	e2	e1	e0
D0 <- S2	codé par :	0	0	1	0	1	0	0
D0 <- S3, D1 <- S3	codé par :	1	1	0	1	0	0	0
		0	1	1	1	0	0	0
		0	1	1	0	0	1	1



Sur un top d'horloge on peut:

- faire passer la valeur d'un registre dans un autre

  regA <- regB

- faire passer la valeur d'un registre dans un circuit combinatoire et charger le résultat dans un autre registre

  regA <- adder (regB, regC)

- faire passer la valeur d'un registre dans un circuit combinatoire et recharger ce registre avec le résultat du calcul

  regA <- adder (regA, regB)

- les valeurs peuvent transiter sur un bus

- tous les registres sont des registres maître/esclave :

- ***ils se chargent sur front***
- ***un registre peut servir à la fois de source et de destination***

Exemple de micro-instructions pour PROCSI

page 5

rd1 ← rd1 + rd2	*f1f2f3m = 1001*	*plus dans l'ALU*
	adbus = 001	*résultat de l'ALU sur le bus*
	lrd1 = 1	*charge rd1*
rd1 ← reg[ir(3..5)]	*mux = 0*	*lecture REG adresse ir(3..5)*
	$\overline{csreg} = 0$	
	$r/\overline{wreg} = 1$	
	adbus = 011	*sortie de REG sur le bus*
	lrd1 = 1	*charge rd1*

2. Machines d'états finis (Finite State Machine)

Modèle de base pour décrire le comportement d'un objet (informatique) qui:

- peut être dans un ensemble fini d'états
- selon son état actuel, selon les évènements extérieurs,
  - envoie des messages à l'extérieur
  - change d'état

Exemples

. distributeur automatique de monnaie
. feu de circulation pour piéton
. automate de scénario d'une IHM
. automate reconnaisseur d'un langage
. circuit séquentiel synchrone

2.1 Définition d'une FSM

Modèle de Moore:

- ensemble fini d'états S1, S2, ..., Sk (dont état initial)
- ensemble fini d'entrées binaires I1, I2, ... Im
- ensemble fini de sorties binaires O1, O2, ..., 0p
- ensemble de règles de transition d'état spécifiant pour chaque valeur de l'état courant Sc et valeur des entrées I1, I2, ... Im la valeur du prochain état
- ensemble de règles de sortie spécifiant pour chaque valeur de l'état courant Sc la valeur de chacune des sorties.

Modèle de Mealy:
les sorties dépendent aussi des entrées.

Exemple : distributeur bancaire

Entrées: touches , lecteur carte

Sorties: écran, billets

Etats: attente client
code 1er essai
code 2ème essai
choix montant
distribution billets
confisquer carte

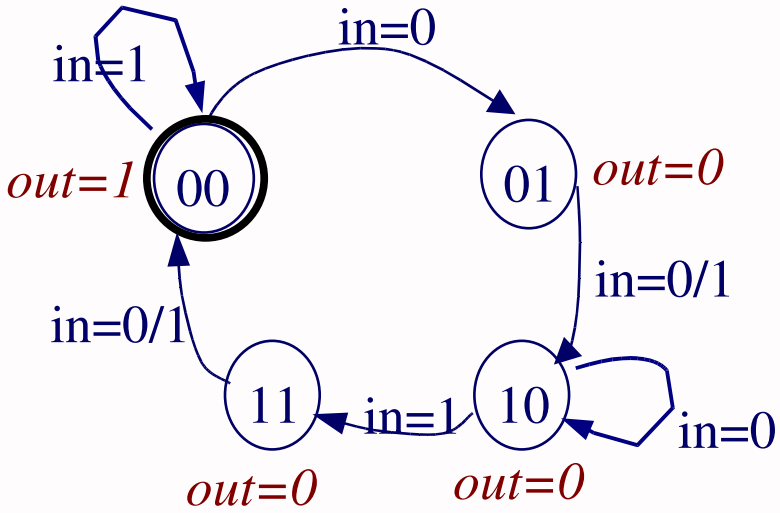
Règles de sortie:
attente client: « bienvenue, introduisez votre carte »
code 1er essai : « votre code (1er essai) »
code 2ème essai : « votre code (2ème essai) »
choix montant : « choisissez le montant »
distribution billets : « n'oubliez pas vos billets »
confisquer carte : « contacter votre banque pour récupérer votre carte »

Transition d'état:

Etat courant	Condition d'entrée	Etat suivant
attente client	*introduction carte* →	code 1er essai
code 1er essai	*code bon* →	choix montant
code 1er essai	*code erroné* →	code 2ème essai
2ème essai	 →	confisquer carte
2ème essai	 →	choix montant
choix montant	*montant* →	distribution billets
distribution billets	→	attente client
confisquer carte	→	attente client

2.2. Représentation des FSM

Diagramme de transition d'état



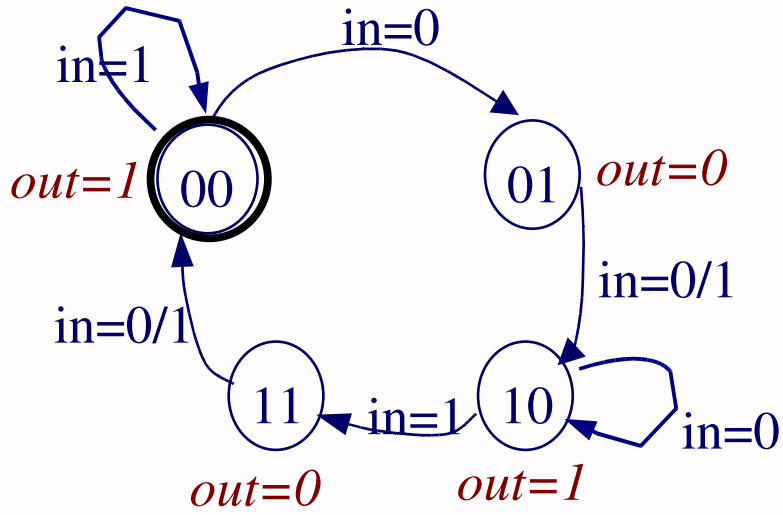
Dans l'état 00 :

- *la sortie vaut 1*
- *si l'entrée est 0 on passe à l'état 01*
- *si l'entrée est 1 on passe à l'état 00*

2.3. Réalisation des FSM

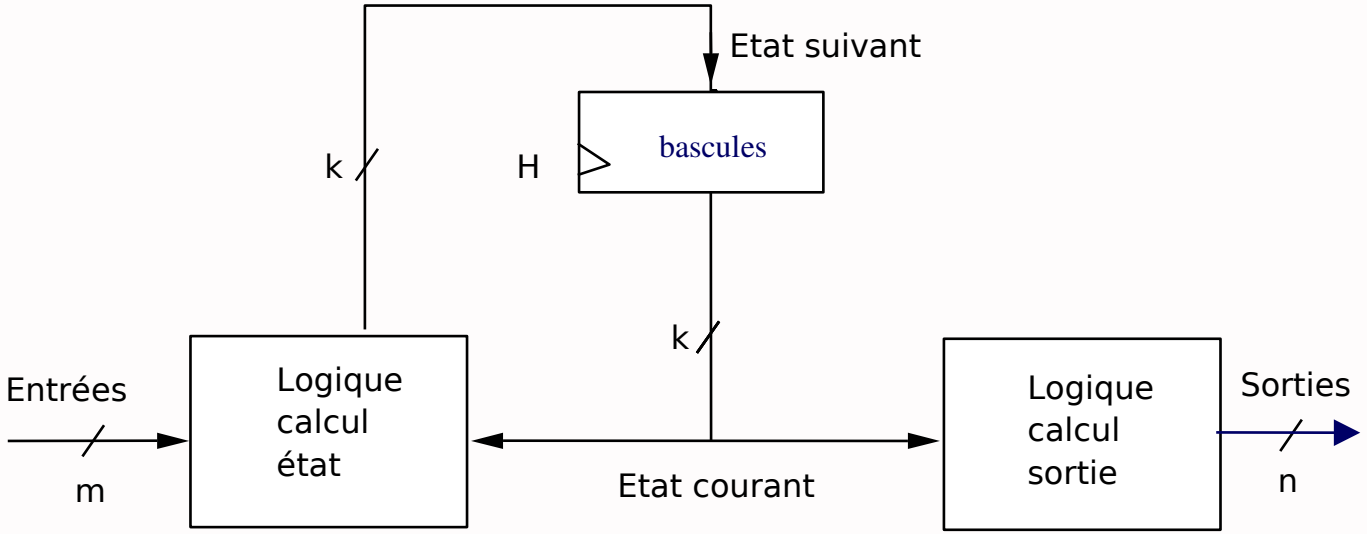
Table de transition d'état

état courant q1q0	entrée in	état suivant q1+q0+	sortie out
00	0	01	1
00	1	00	1
01	0	10	0
01	1	10	0
10	0	10	1
10	1	11	1
11	0	00	0
11	1	00	0



Contrôle cablé : schéma général pour implémenter une machine de Moore

m entrées, n sorties, 2^k états



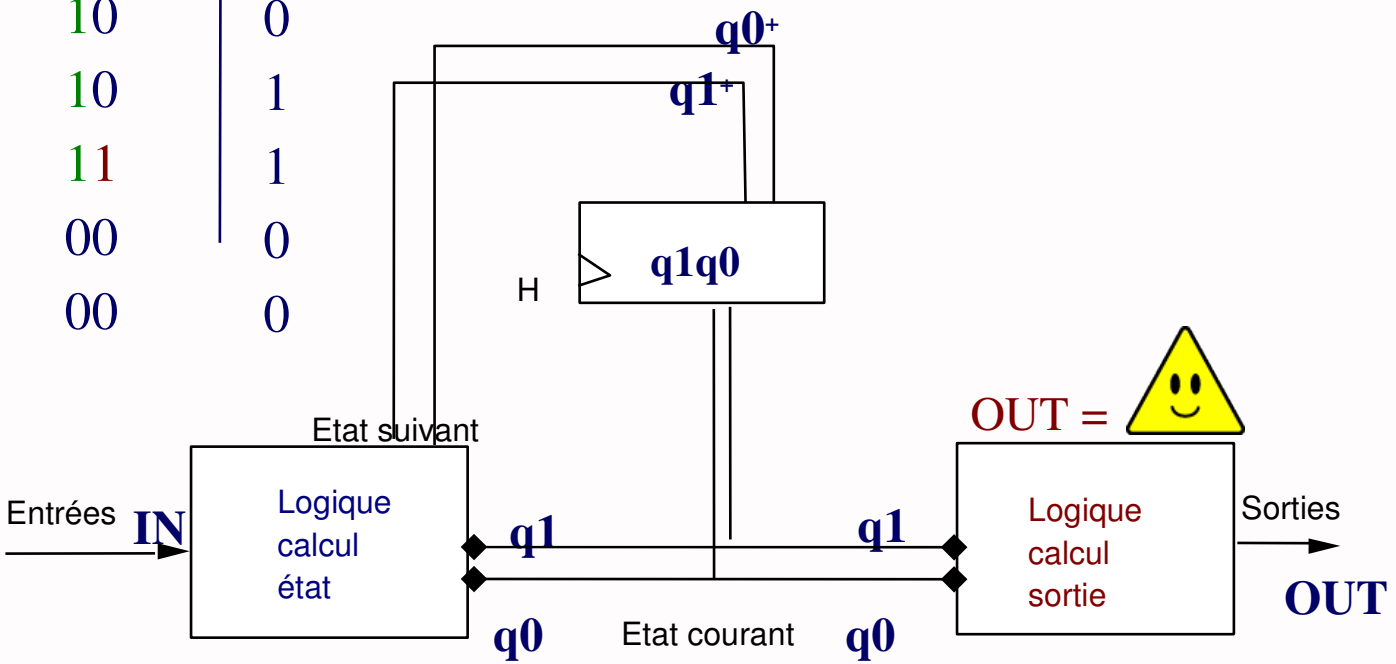
Attention : le fonctionnement est *synchrone*

le changement d'état se fait sur *top d'horloge*

Partie contrôle

état courant q1q0	entrée in	état suivant q1+q0+	sortie out
00	0	01	1
00	1	00	1
01	0	10	0
01	1	10	0
10	0	10	1
10	1	11	1
11	0	00	0
11	1	00	0

Circuit logique



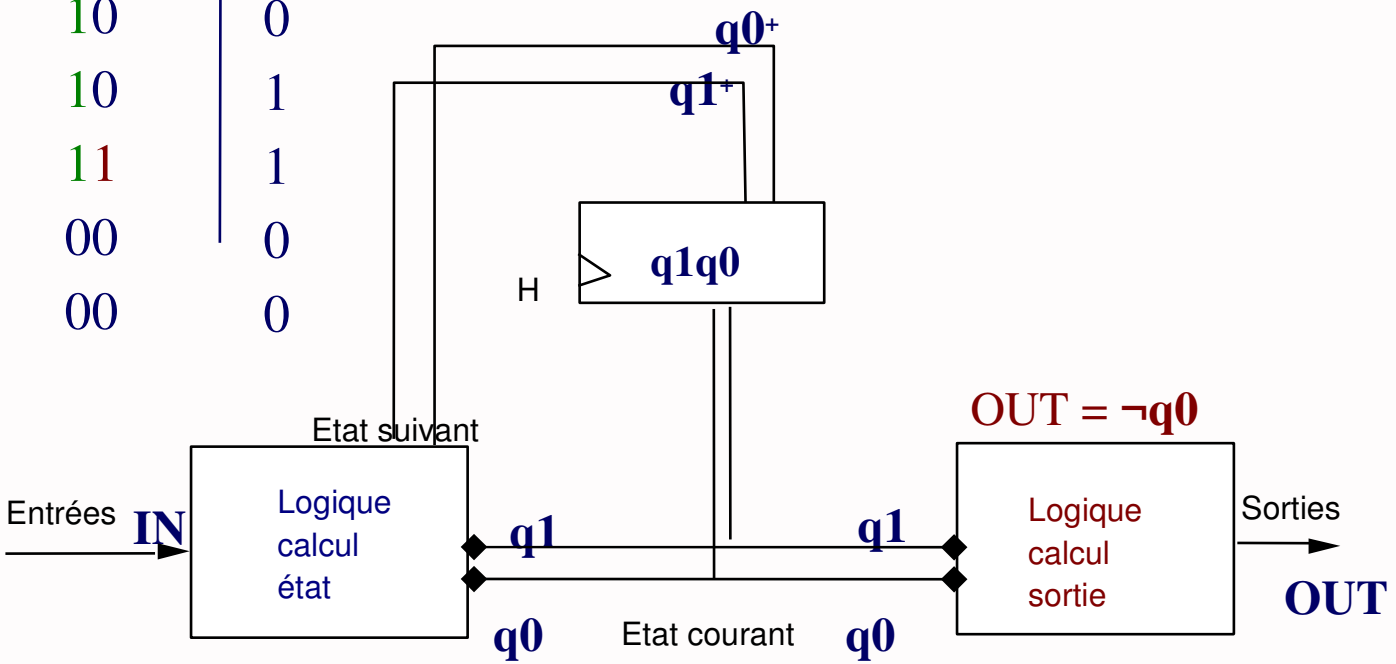
$q0^+ = \neg in.\neg q1.\neg q0 + in.q1.\neg q0$

$q1^+ = \neg q1.q0 + q1.\neg q0$

Partie contrôle

état courant q1q0	entrée in	état suivant q1+q0+	sortie out
00	0	01	1
00	1	00	1
01	0	10	0
01	1	10	0
10	0	10	1
10	1	11	1
11	0	00	0
11	1	00	0

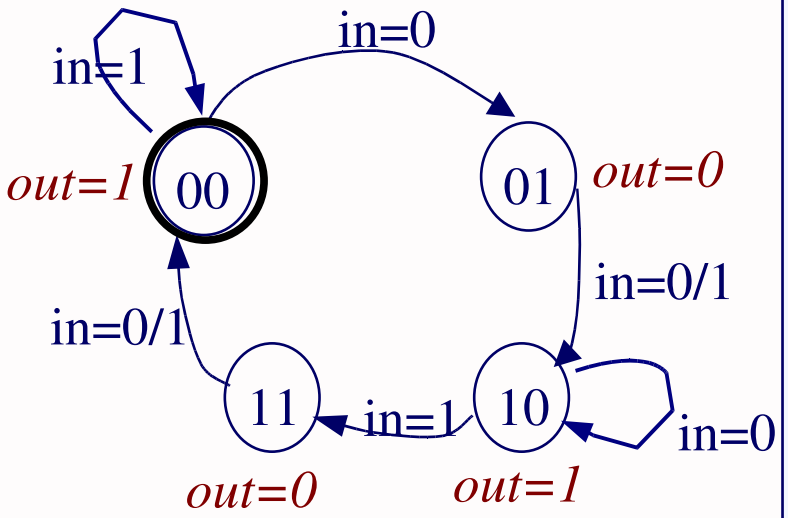
Circuit logique

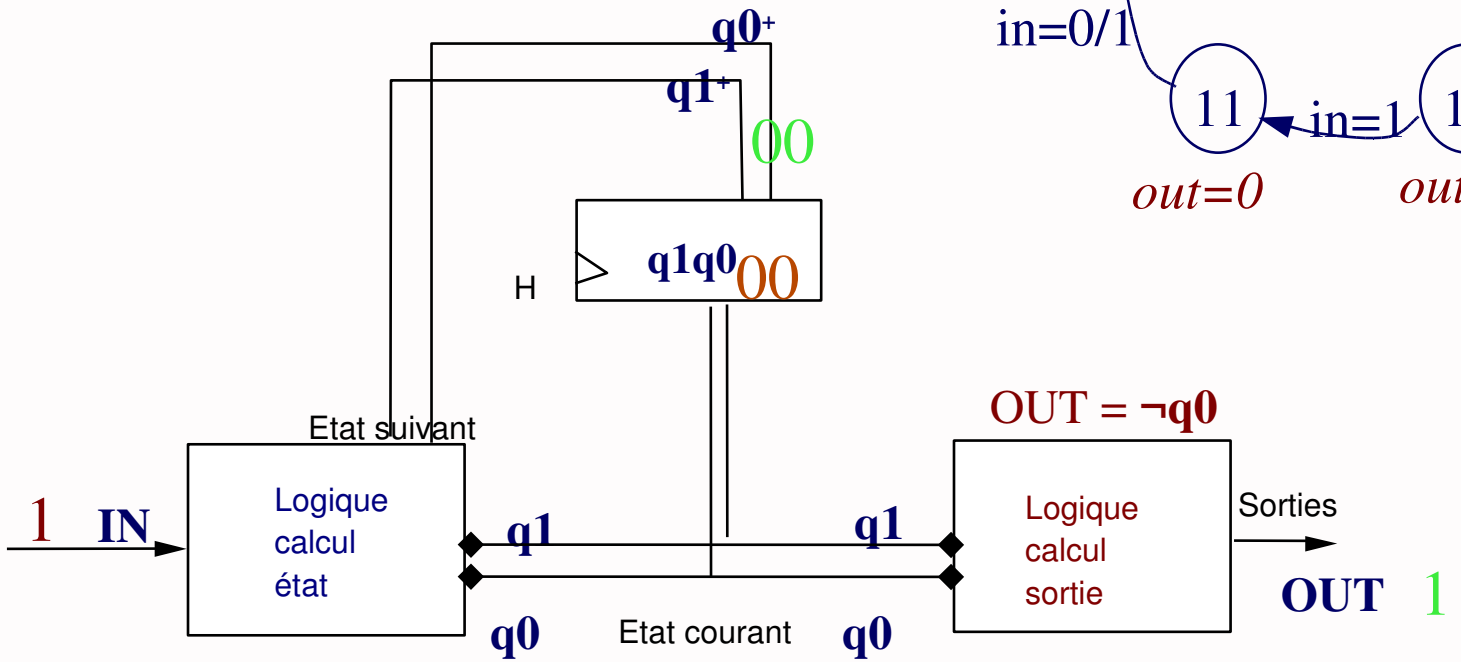


$q0^+ = \neg in.\neg q1.\neg q0 + in.q1.\neg q0$

$q1^+ = \neg q1.q0 + q1.\neg q0$

Etat initial : q1q0 = 00, IN = 1



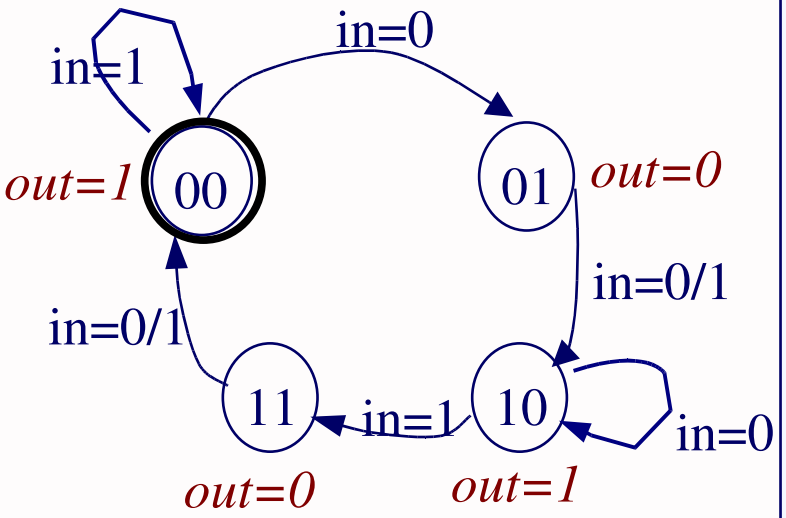


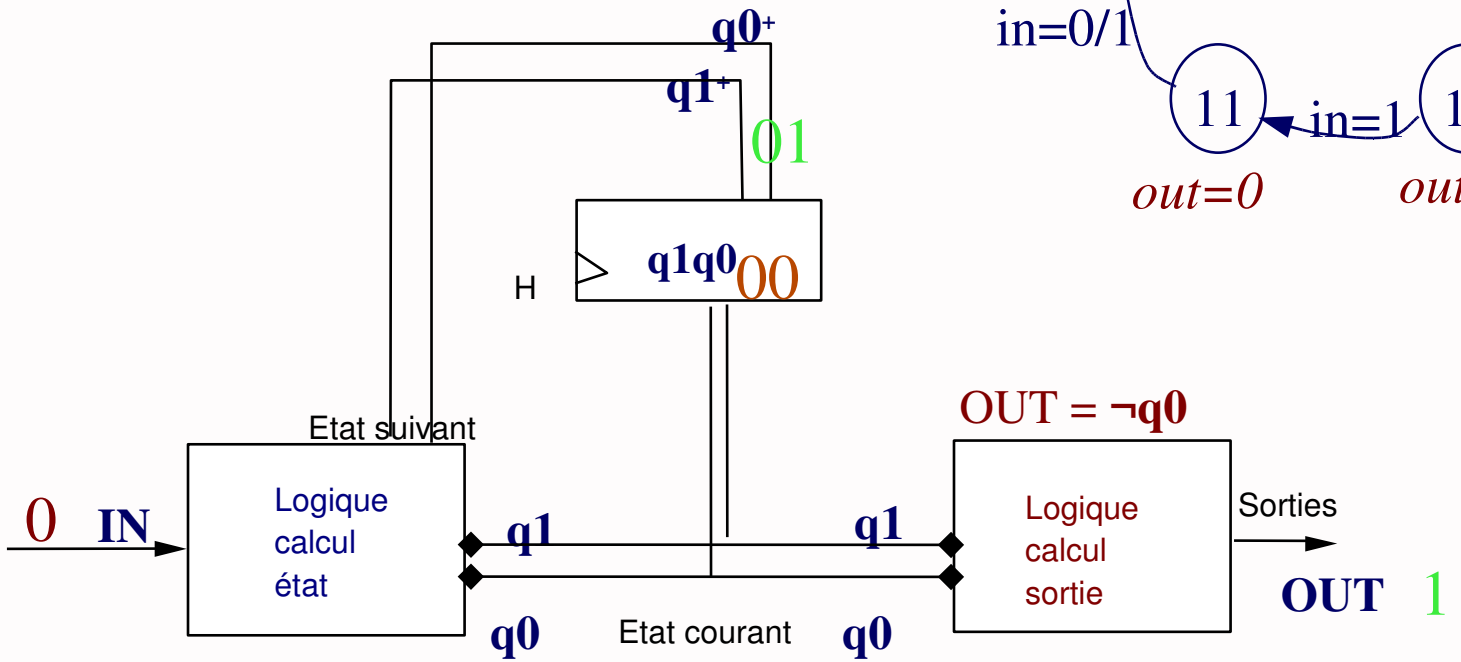
$q0^+ = \neg in.\neg q1.\neg q0 + in.q1.\neg q0$

$q1^+ = \neg q1.q0 + q1.\neg q0$

Après 1 top d'horloge :

état courant : q1q0 = 00, IN = 0



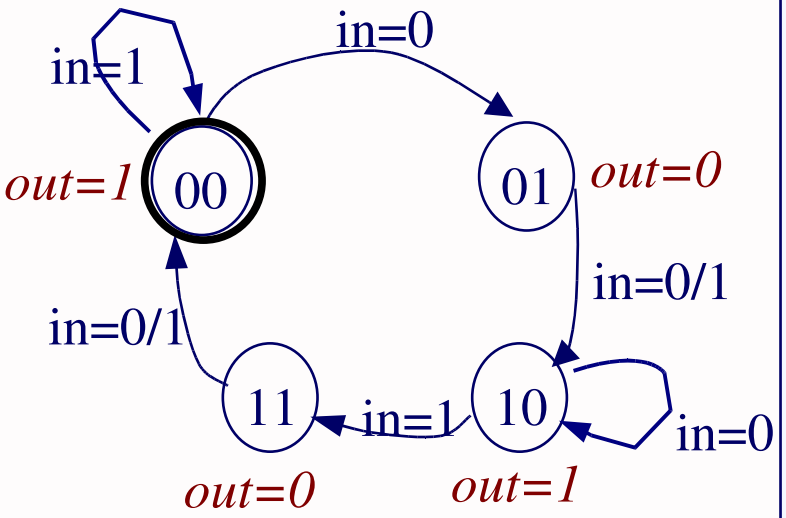


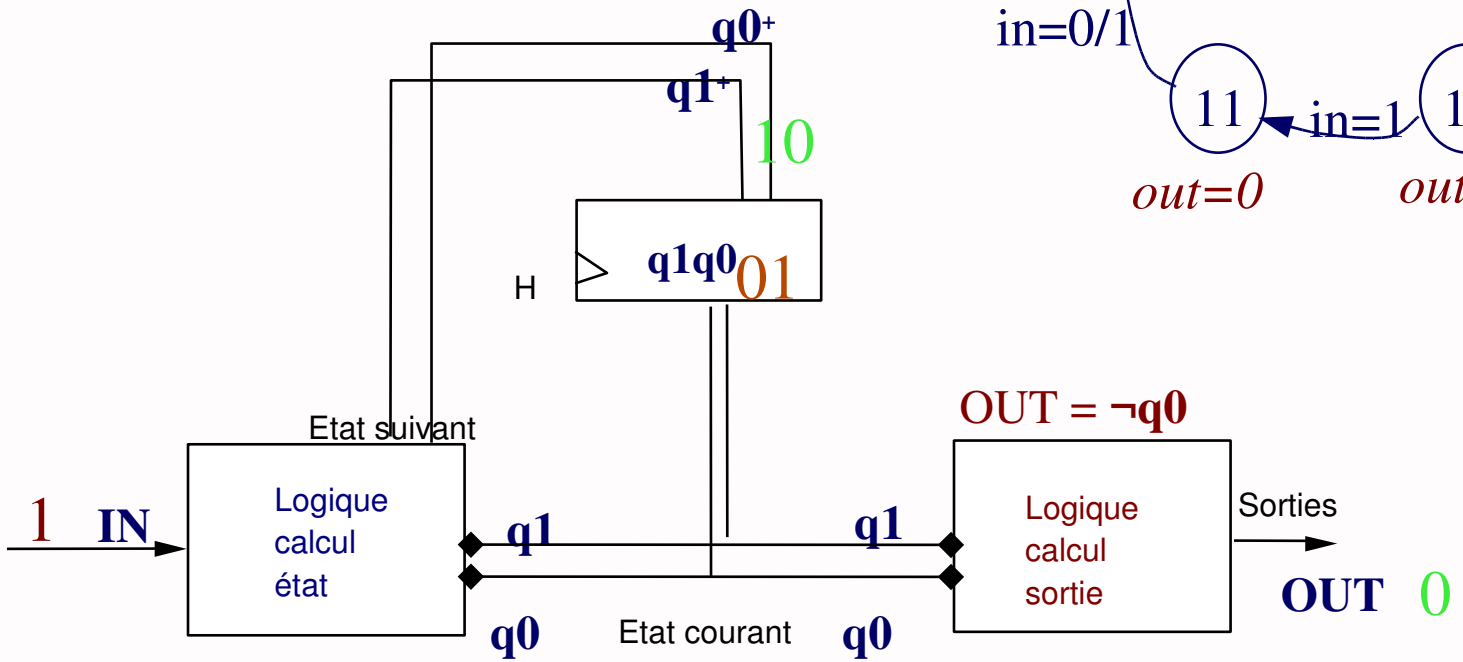
$q0^+ = \neg in.\neg q1.\neg q0 + in.q1.\neg q0$

$q1^+ = \neg q1.q0 + q1.\neg q0$

Après 2 tops d'horloge :

état courant : q1q0 = 01, IN = 1

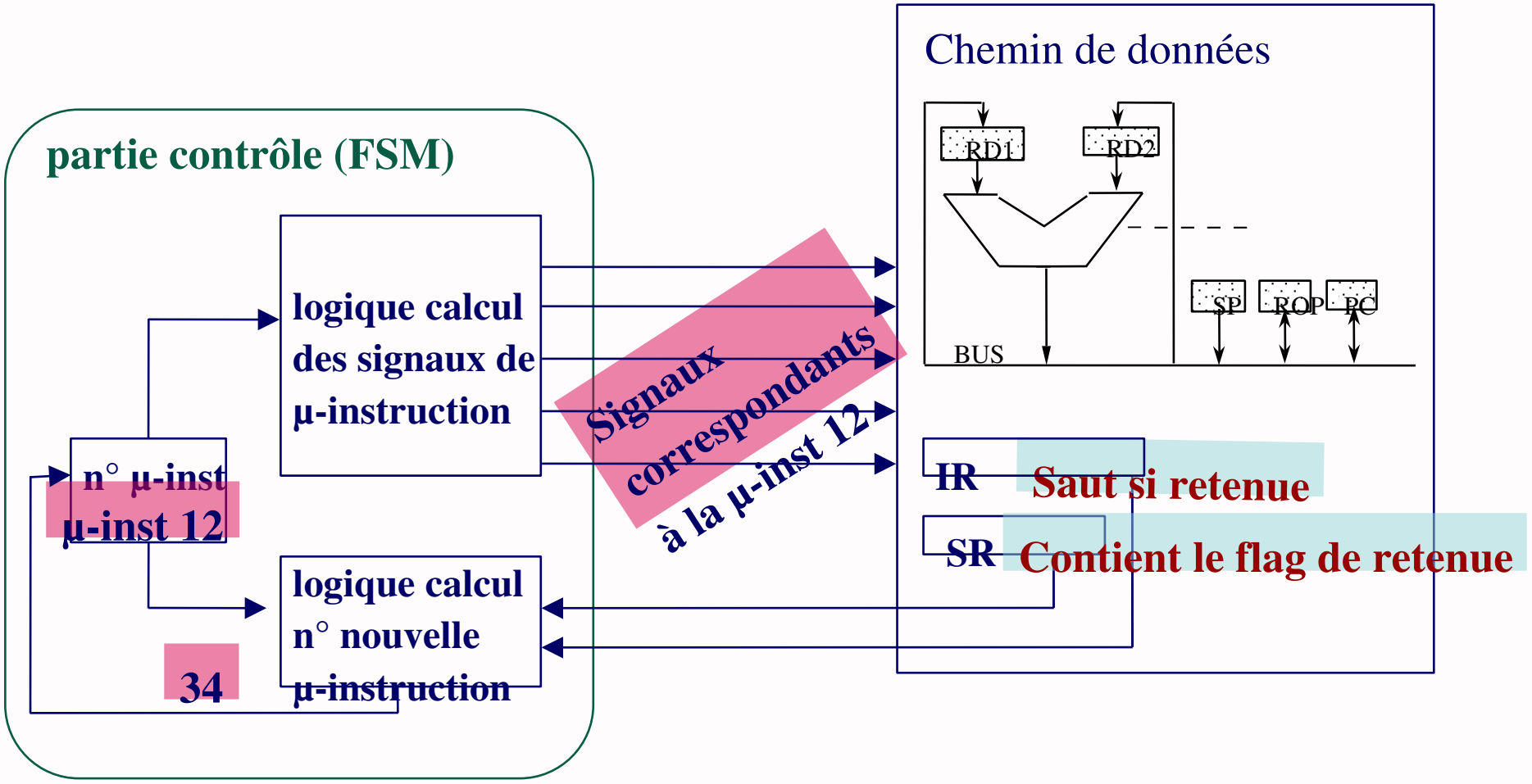




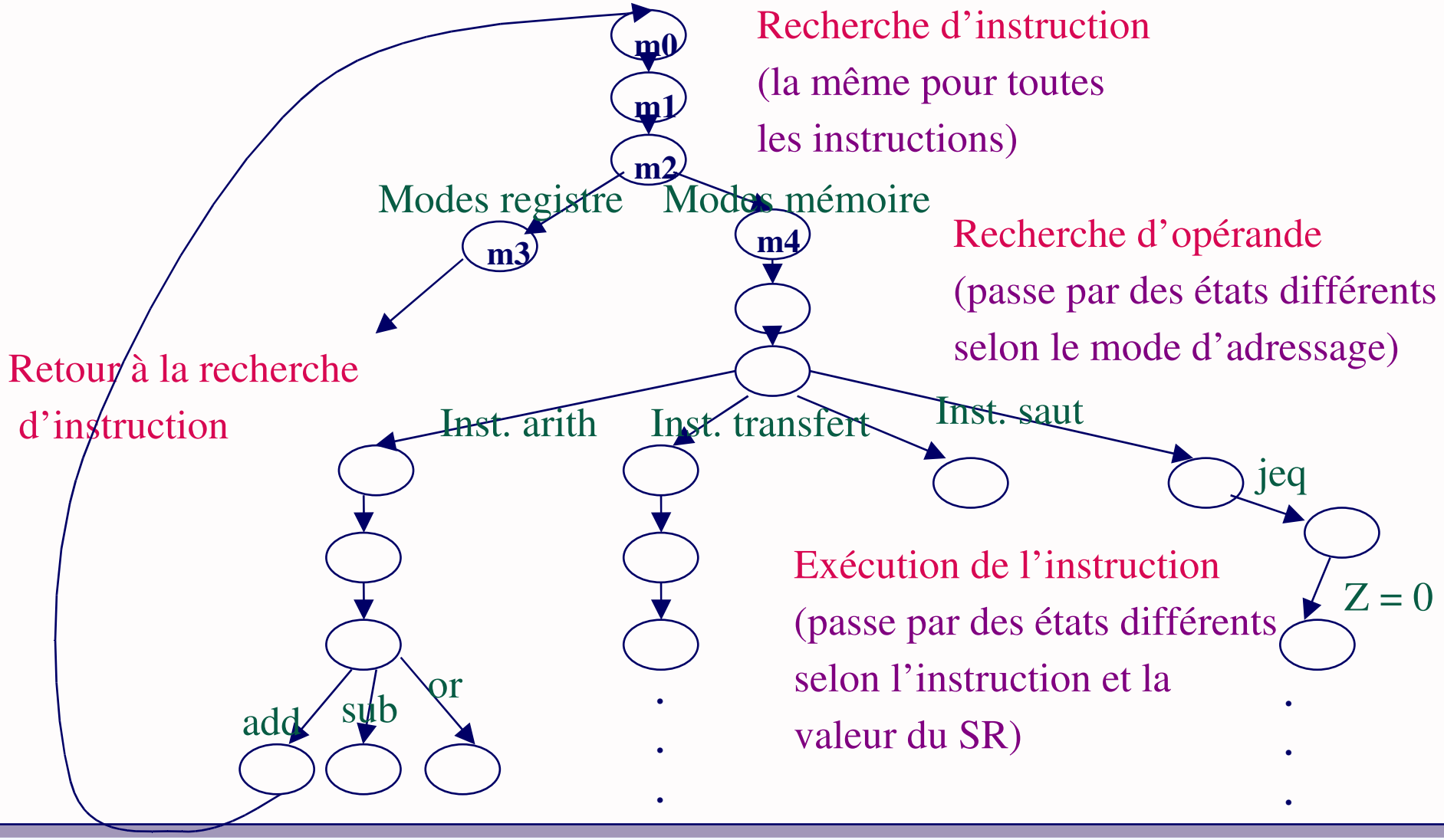
$q0^+ = \neg in.\neg q1.\neg q0 + in.q1.\neg q0$

$q1^+ = \neg q1.q0 + q1.\neg q0$

2.4. FSM d'un processeur

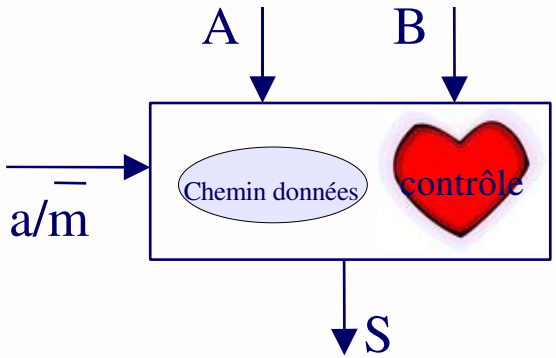


Graphe de transition de la FSM d'un processeur



3. Exemple de partie contrôle: un additionneur / multiplieur

- Entrées de données: A et B
- Entrées de contrôle: $a/\overline{m}$



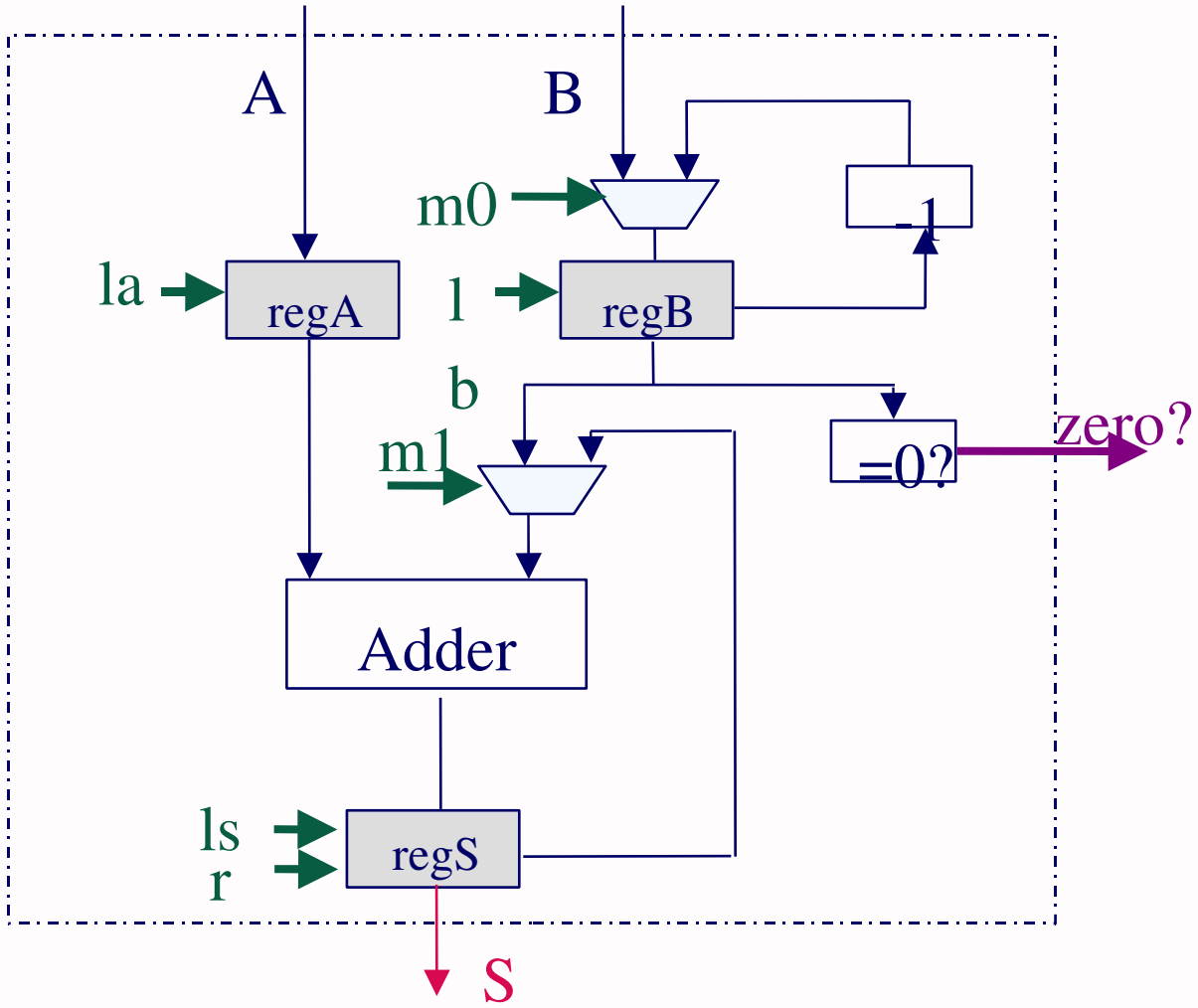
- Sortie de données: S

- Spécification:

  Si $a/\overline{m} = 1$ alors S =A+B après 1 top d'horloge

  Si $a/\overline{m} = 0$ alors S =A*B après B tops d'horloge

Chemin de données



Signaux des composants du chemin de données

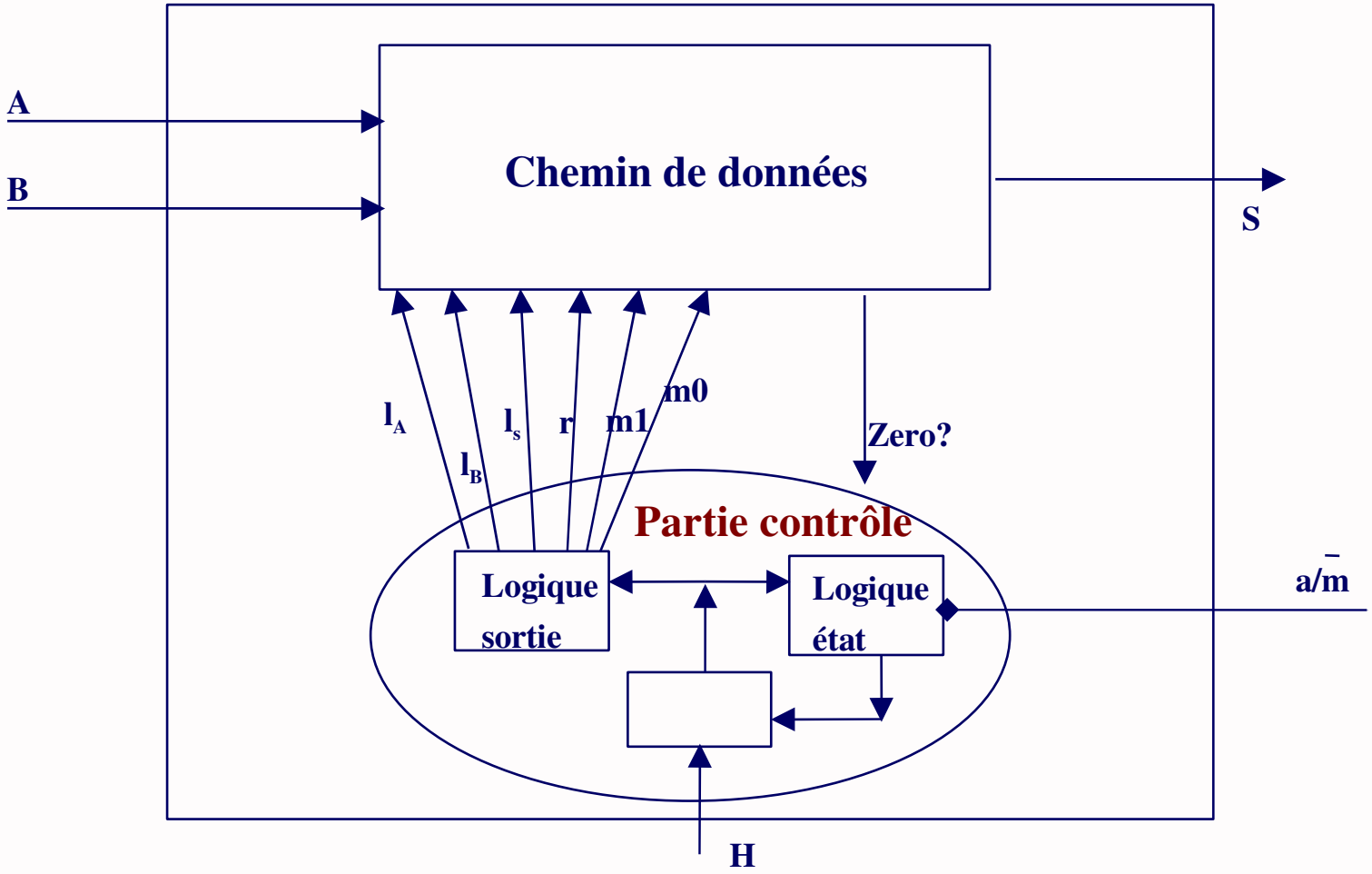
En entrée des composants (sortie partie contrôle) : la lb ls r m0 m1

- la : load registre A
- lb : load registre B
- ls : load registre S
- r : reset S
- m0 : si m0 = 1, B est en entrée de regB, sinon c'est la sortie de l'opérateur -1
- m1 : si m1 = 1, la 2ème entrée de l'adder est regB sinon c'est regS

En sortie des composants (entrée partie contrôle) :

zero? : vaut 1 si regB vaut 0

L'additionneur / multiplieur complet

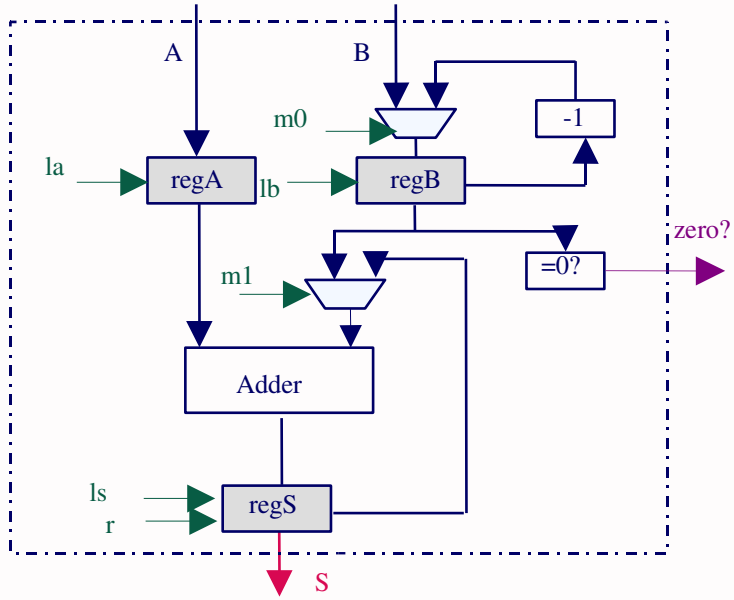


Conception d'une partie contrôle pour que le chemin de données réalise la spécification de l'additionneur / multiplieur

1. "Algorithme"
2. Diagramme de transition d'état
   - Quelles sont les opérations nécessaires et comment sont-elles séquencées ?
   - Quels sont les signaux à activer pour réaliser ces opérations
3. Table de transition d'état
4. *Construire le circuit logique (en utilisant le schéma générique)*

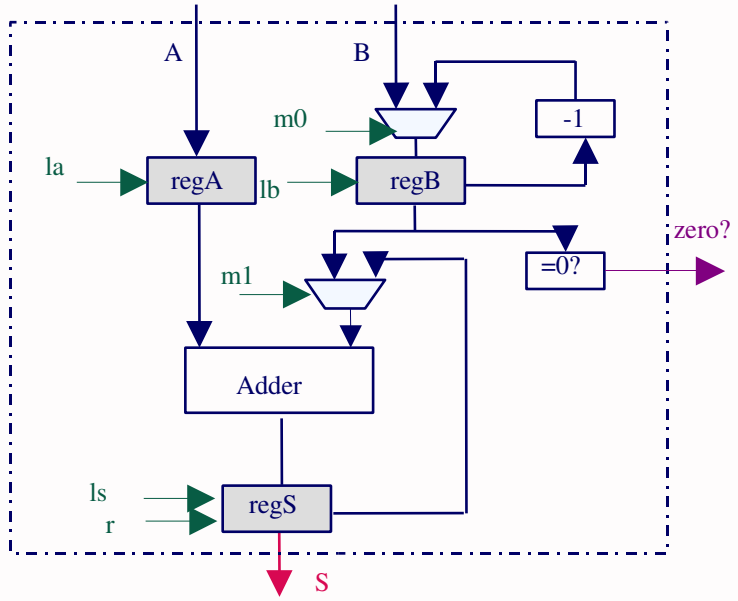
Algorithme

regA = A
regB = B
regS = 0
Si a/m = 1 // addition
    regS = regA + regB
sinon // multiplication
    tant que 
    
    fin tantque

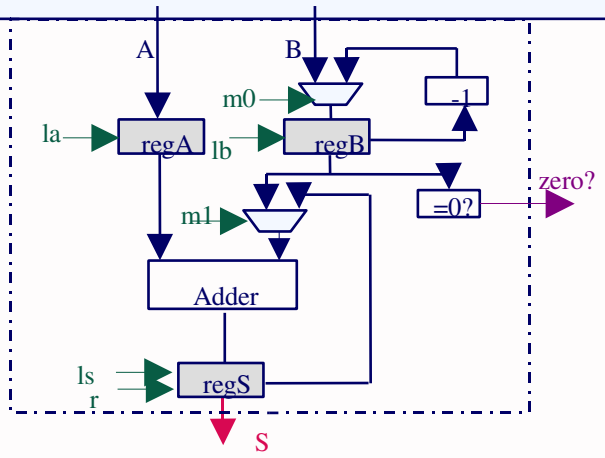


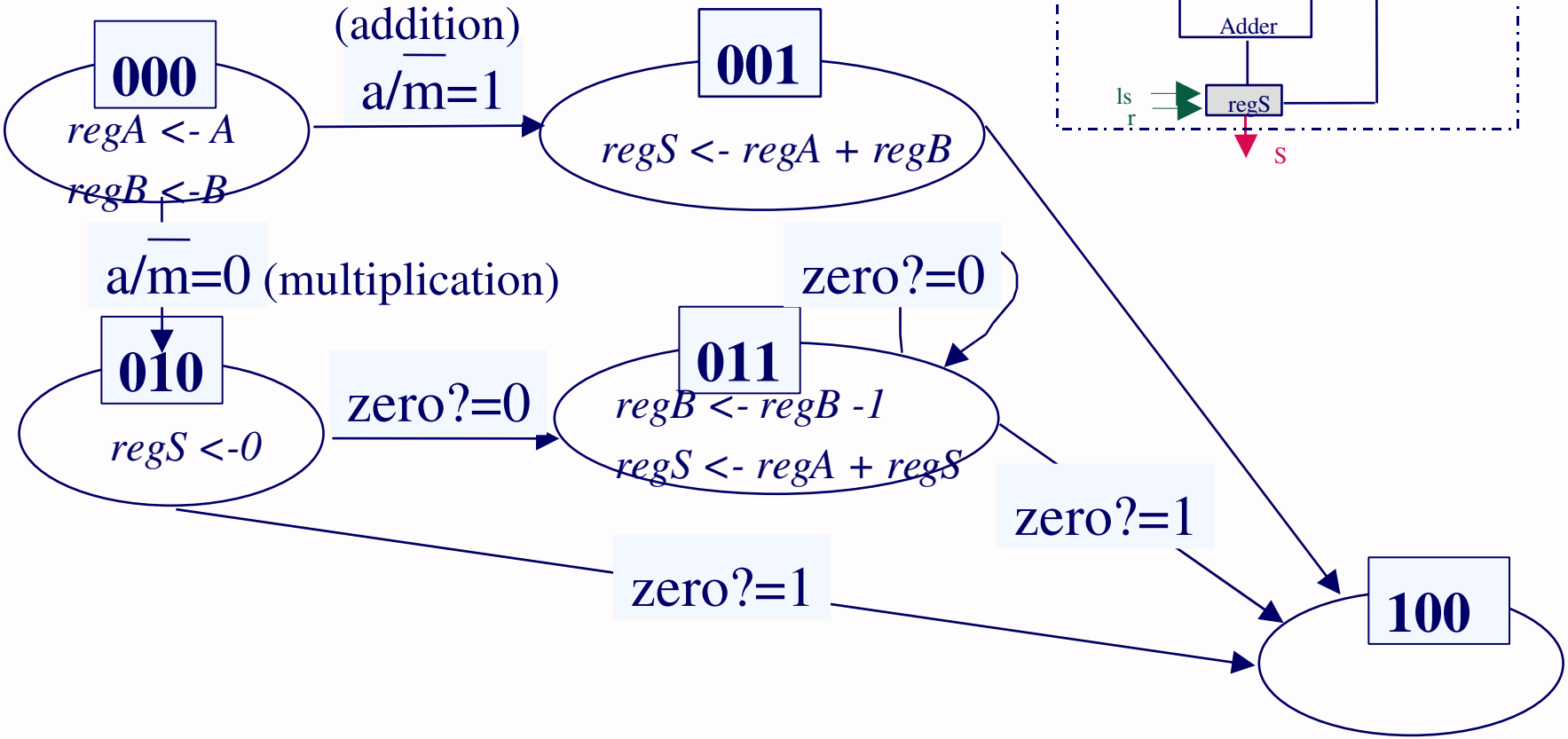
Algorithme

```
regA = A
regB = B
regS = 0
Si a/m = 1          // addition
      regS = regA + regB
sinon               // multiplication
      tant que regB!=0
            regS = regS + regA
            regB--
      fin tantque
```

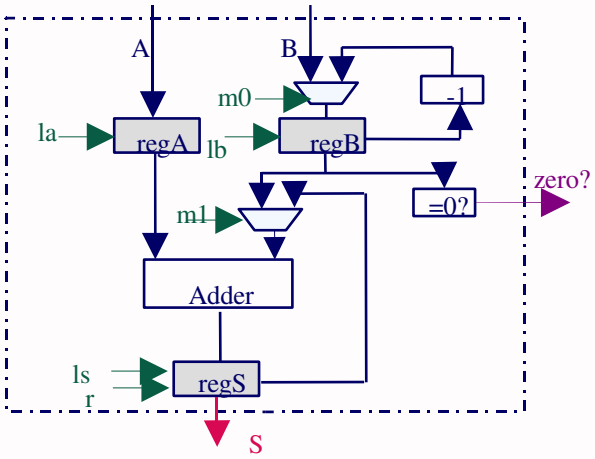


Séquencement des actions





Signaux de contrôle à émettre



la lb ls r m0 m1
1 1 0 0 1 0

(addition)

000
regA <- A
regB <-B

$\overline{a/m}$=1

la lb ls r m0 m1

001
regS <- regA + regB

$\overline{a/m}$=0 (multiplication)

010
regS <-0

la lb ls r m0 m1
0 0 0 1 0 0

zero?=0

011
regB <- regB -1
regS <- regA + regS

la lb ls r m0 m1
0 1 1 0 0 0

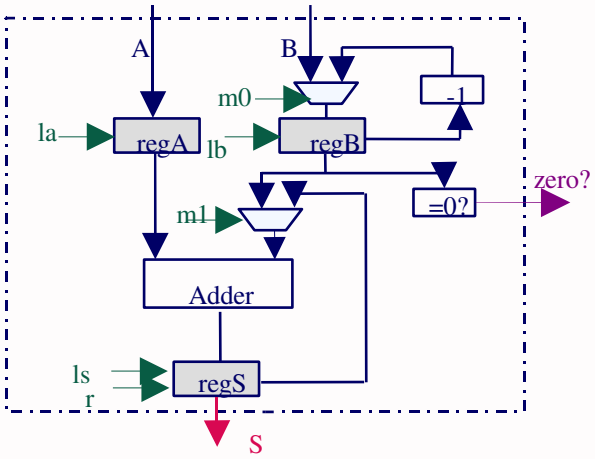
zero?=0

zero?=1

zero?=1

100

Signaux de contrôle à émettre



la lb ls r m0 m1
1 1 0 0 1 0

(addition)

000
regA <- A
regB <-B

$\overline{a/m}$=1

la lb ls r m0 m1
0 0 1 0 0 1

001
regS <- regA + regB

$\overline{a/m}$=0 (multiplication)

010
regS <-0

zero?=0

la lb ls r m0 m1
0 0 0 1 0 0

zero?=0

011
regB <- regB -1
regS <- regA + regS

la lb ls r m0 m1
0 1 1 0 0 0

zero?=1

zero?=1

100

Table de transition d'état de la FSM de la partie contrôle:

Etats		Entrées		Sorties					
q	q+	a/m	zero?	la	lb	ls	m0	m1	r
000	001	1	X	1	1	0	1	0	0
000	010	0	X	1	1	0	1	0	0
001	100	X	X	0	0	1	0	1	0
010	011	X	0	0	0	0	0	0	1
010	100	X	1	0	0	0	0	0	1
011	011	X	0	0	1	1	0	0	0
011	100	X	1	0	1	1	0	0	0

On en déduit les fonctions de calcul des sorties et du nouvel état

5. Les interruptions

Évènement autre qu'un branchement qui modifie le cours normal de l'exécution

Partie la plus difficile du contrôle

Exemples

- **requête d'entrée / sortie (clavier, souris, écran, …)**
- **demande routine système par l'utilisateur**
- **trace ou point d'arrêt dans l'exécution d'un programme**
- **dépassement de capacité, division par 0**
- **défaut de page mémoire**
- **problème matériel**
- **coupure alimentation**

5.1. Interruptions logicielles (instruction trap)

Quand un programme veut faire appel à des fonctions système

APPEL: par une instruction machine TRAP ou INT

TRAITEMENT:

- l'état du processeur est sauvegardé en mémoire
- un nouvel état du processeur est chargé
  - il était stocké dans la partie de la mémoire associée à l'interruption
- l'interruption est exécutée
  - le PC a été chargé avec l'adresse de la procédure d'interruption
  - la procédure d'interruption est en mémoire dans une zone réservée
- retour d'interruption

5.2. Conditions d'erreur (faults)

Quand un programme génère une erreur (exception)

- division par 0
- overflow
- accès à un mot mémoire non existant
- stack overflow

TRAITEMENT identique que celui des interruptions logicielles mais

APPEL de la procédure d'interruption par la partie contrôle,
si une condition de faute est détectée (en fonction du registre SR)

5.3. Interruption matérielle

Quand on active un périphérique

reset, accès périphérique (clavier, souris …)

TRAITEMENT identique que celui des interruptions logicielles mais

APPEL de la procédure d'interruption par la partie contrôle,
si une demande d'interruption est active sur une broche

5.4. Prise en compte des interruptions

Test des interruptions par la partie contrôle -> intégré au graphe de transition d'état

- la séquence de recherche d'instruction doit être suivie du test de présence
  - . d'une demande d'entrée / sortie
  - . mode trace
  - . point d'arrêt

- chaque micro-instruction susceptible de générer une interruption doit être suivie d'un test de présence de l'interruption

Exemple: après accès mémoire, test de défaut de page
après opération ALU, test d'un débordement
entre chaque instruction, test d'une demande d'interruption matérielle

5.5. Mise en oeuvre des interruptions

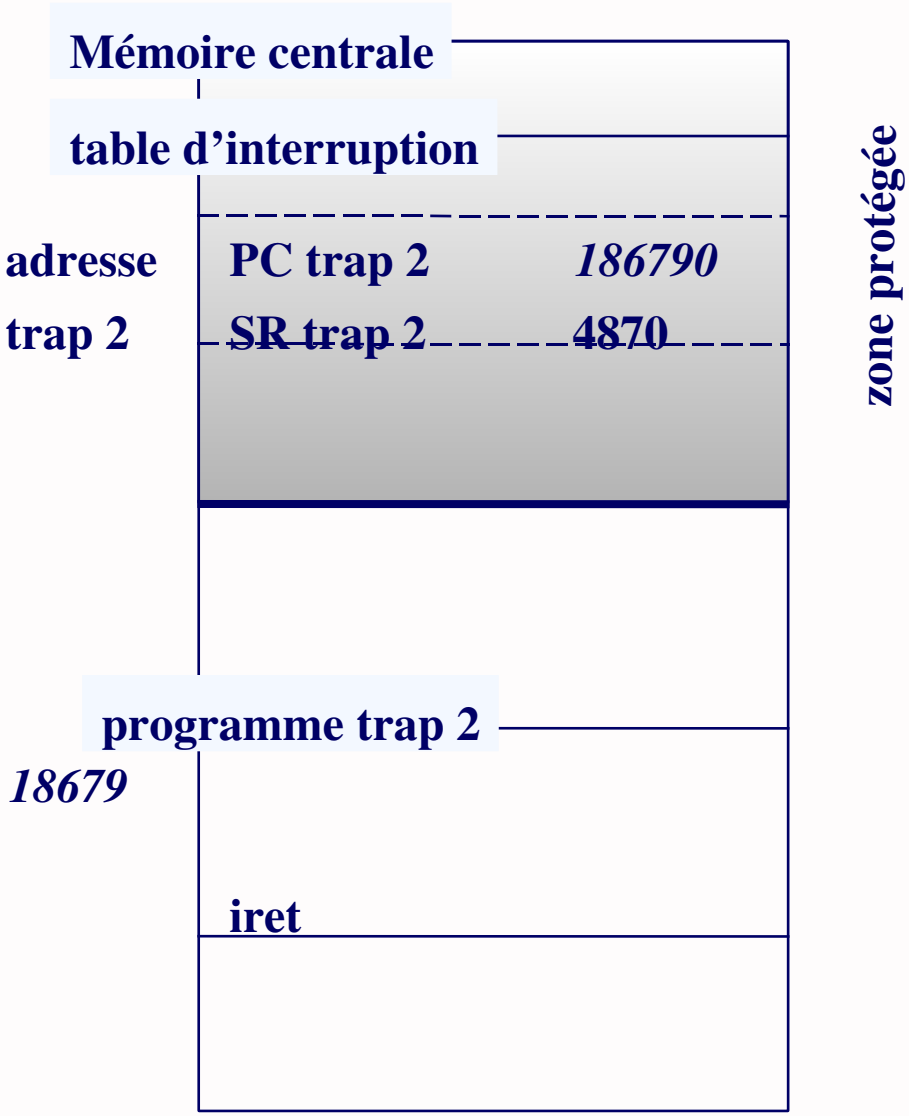
• Les routines d'interruption système sont stockées dans une zone protégée de la mémoire
• La correspondance entre l'interruption détectée et la routine à appeler est stockée dans une table d'interruption

n° interruption	adresse routine interruption
0	8000h

Quand l'interruption n° i est détectée il y a :

- **Sauvegarde de l'état courant** : PC et SR sont empilés
- **Branchement à l'adresse de la routine i** : PC prend la valeur associée à i
- **Retour au programme interrompu** : PC et SR sont restitués à partir de la pile
  - On continue l'exécution (micro-instruction suivante)
  - ou on abandonne -> exécuter la première micro-instruction de recherche d'instruction du programme de prompt

Schéma d'exécution du Trap 2



zone protégée

PC = 308 : trap 2

PC = 310 : add

valeur initiale de SR: 1230

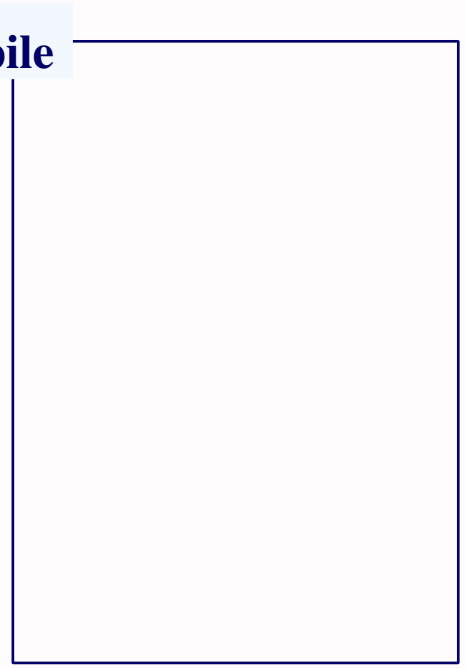
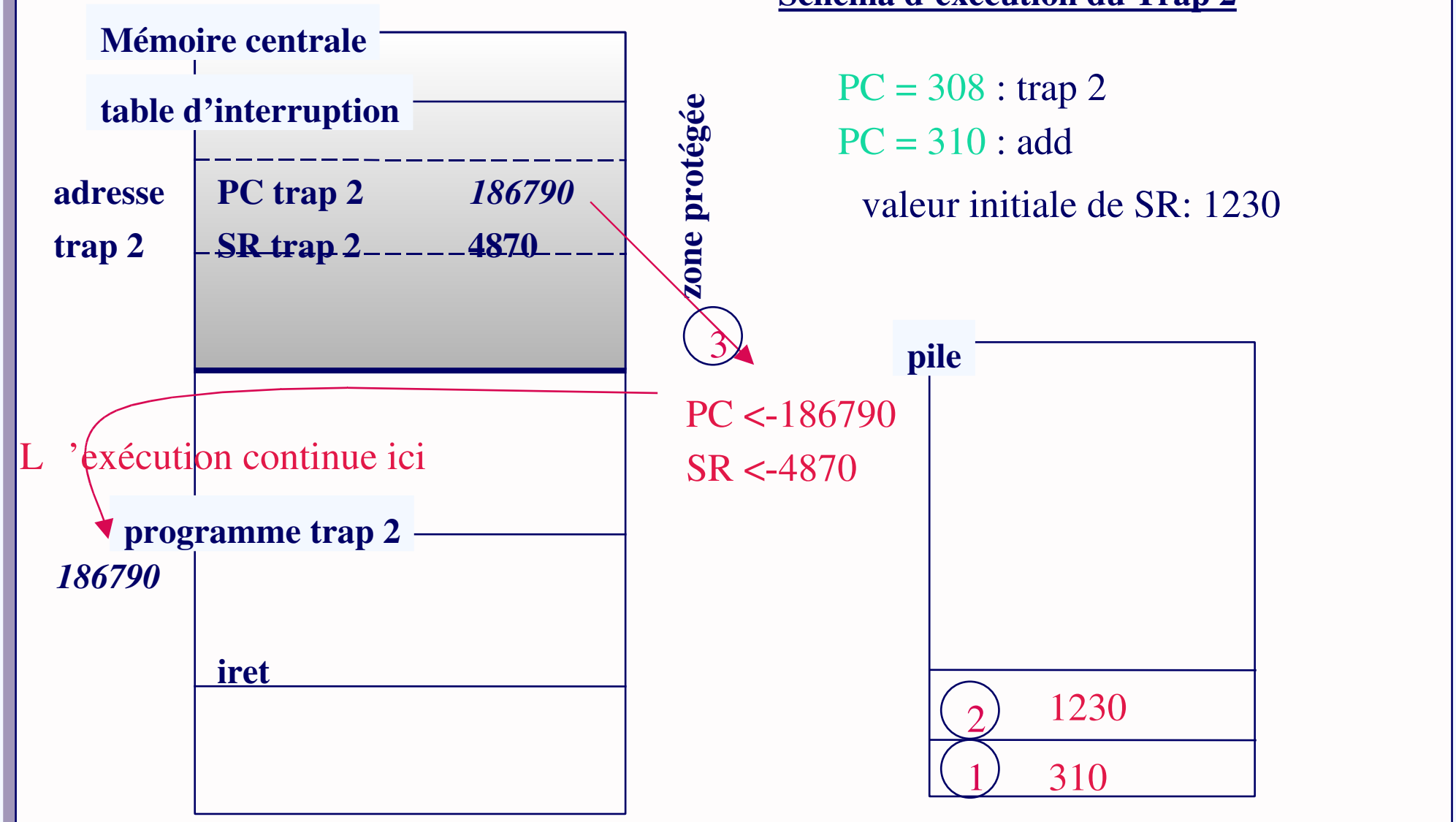
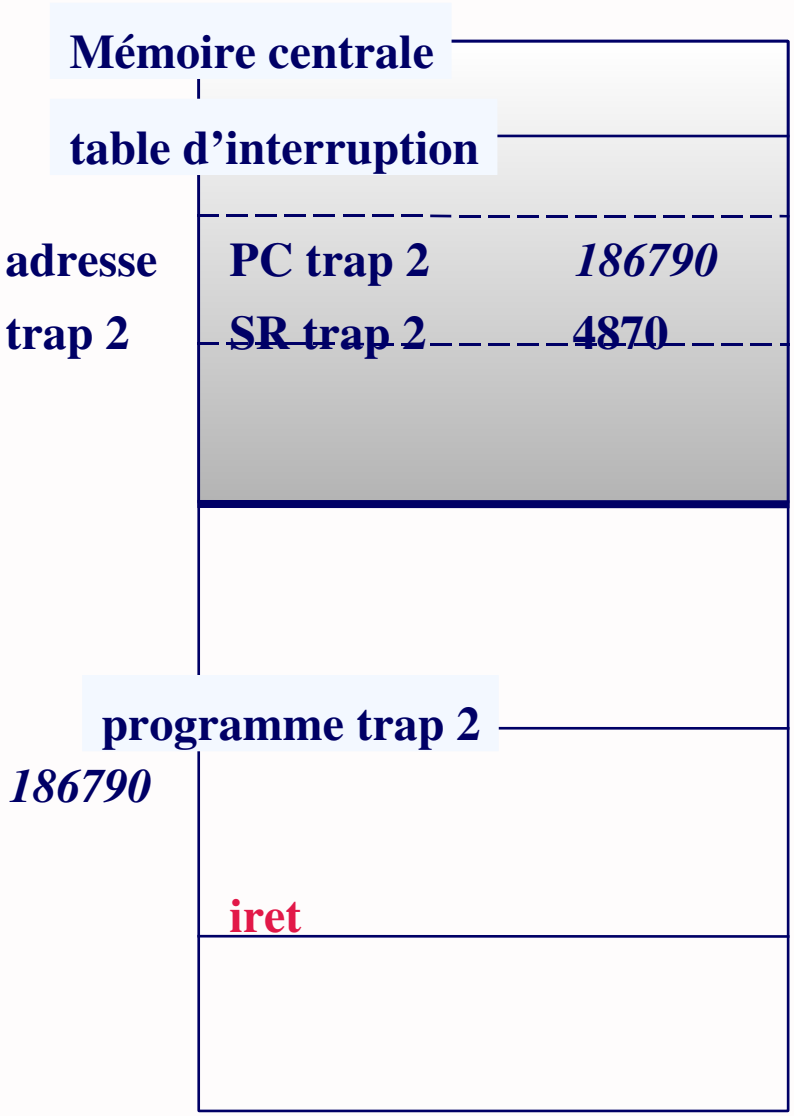
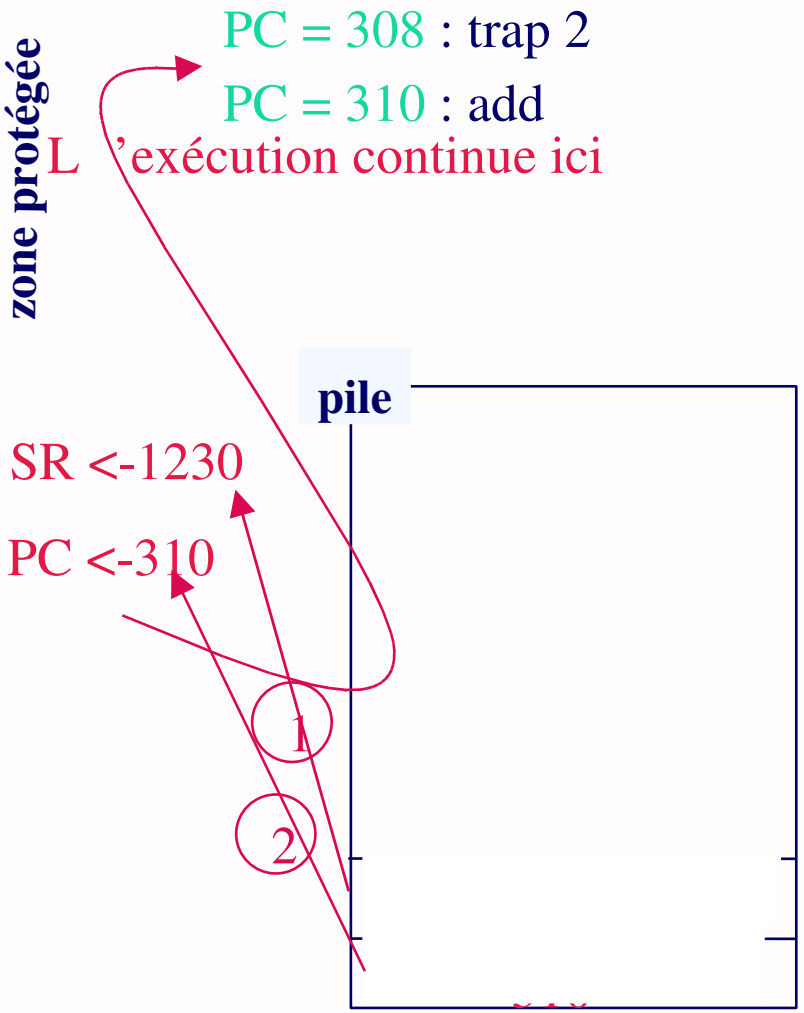


Schéma d'exécution du Trap 2



Sortie du Trap 2: exécution du iret





Récapitulatif

Exécution d'une instruction

Chaque instruction est décomposée en étapes
Chaque étape est décomposée en micro-instructions (opération élémentaire exécutable en un cycle d'horloge)

La partie contrôle

Gère le séquencement des opérations à exécuter dans le chemin de données sur chaque cycle d'horloge du processeur; est implémentée par une FSM

Interruption

Évènement (interne ou externe) qui interrompt le programme en cours