Architecture de Ordinateurs

Module 3

Système (didactique) minimum

Dr. Yannick HERVE

V 1.3

Préliminaire

Objectif : deux données sont disponibles dans une mémoire, on veut calculer la somme et la ranger en mémoire juste à la suite

Composants

- Mémoire
- Additionneur (en fait ALU)
- Fils de connexion (BUS)
- Autres composants ?
- Electronique de contrôle ?

Préliminaire

Objectif : deux données sont disponibles dans une mémoire, on veut calculer la somme et la ranger en mémoire juste à la suite

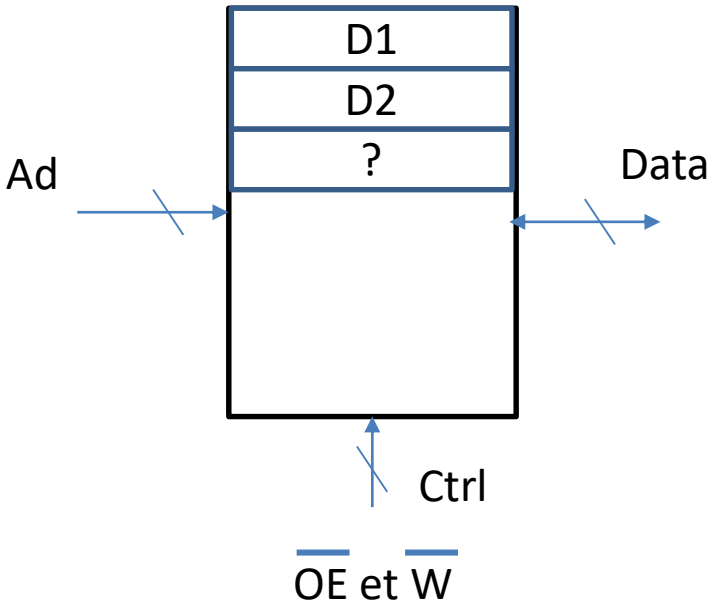
Composants

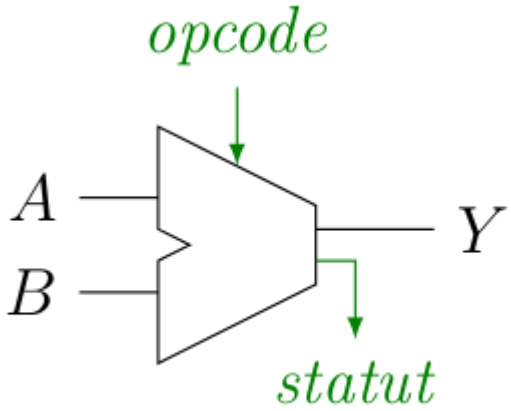
- Mémoire
- Additionneur (en fait ALU)
- Fils de connexion (BUS)
- Autres composants ?
- Electronique de contrôle ?

Avertissement bienveillant

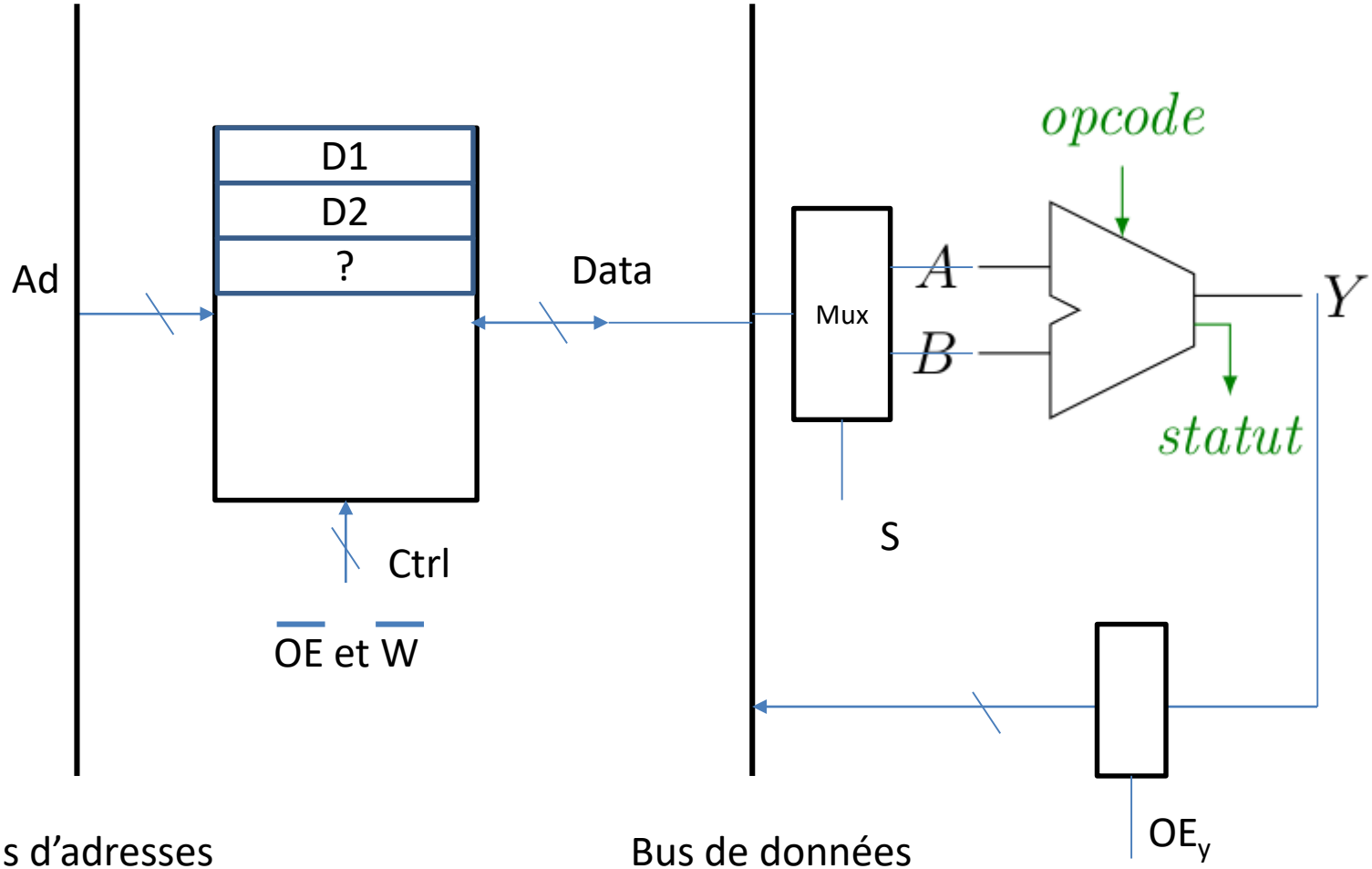
Ce module est primordial et contient tous les concepts de base.
Ne pas le comprendre compromet fortement la suite ….

Comment relier ?

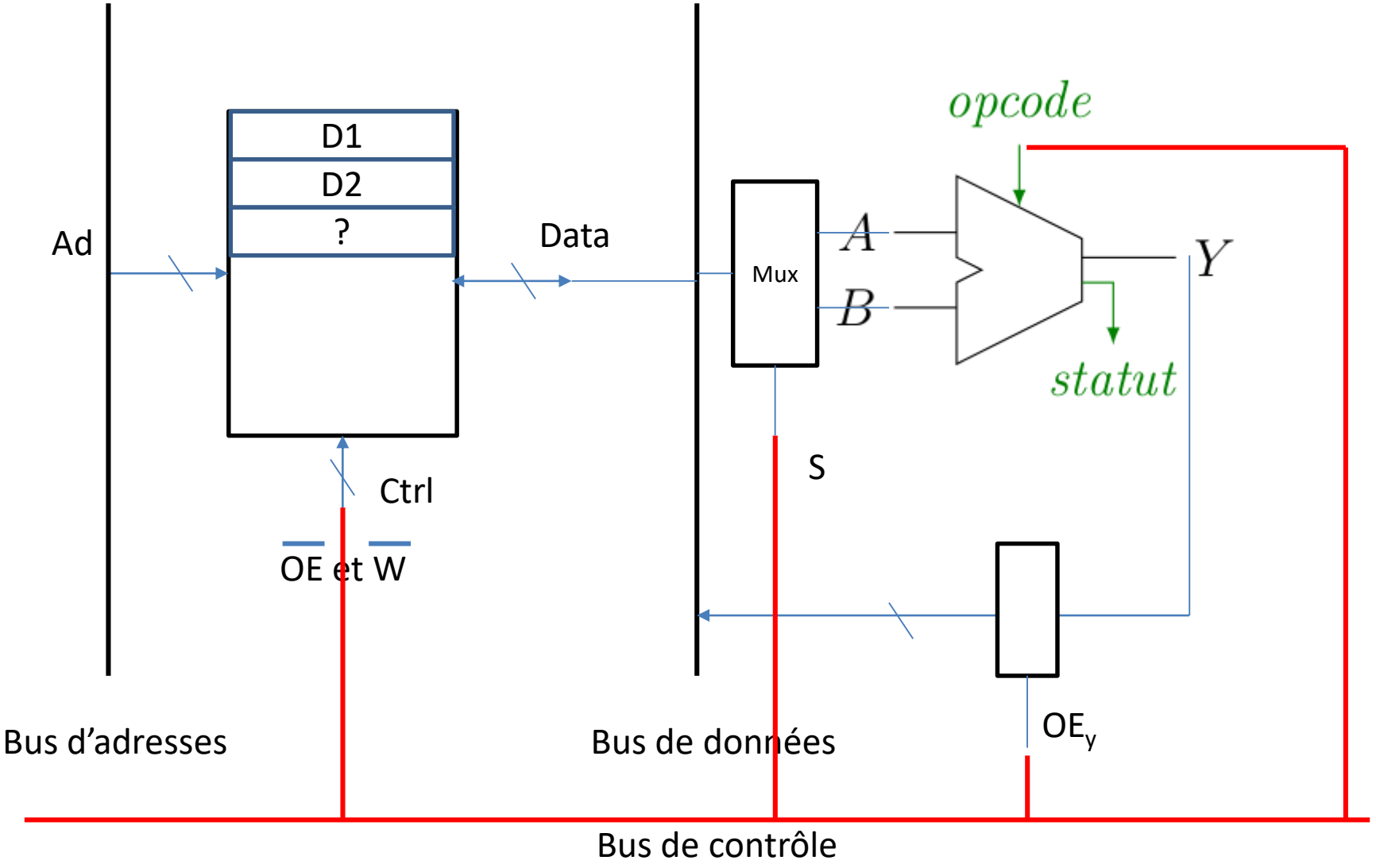




Comment relier ?



Comment relier ?



Séquence de contrôle (1)

Adresse, OE/, W/, S, Opcode, OEy

Hyp : Code pour ADD = 0010

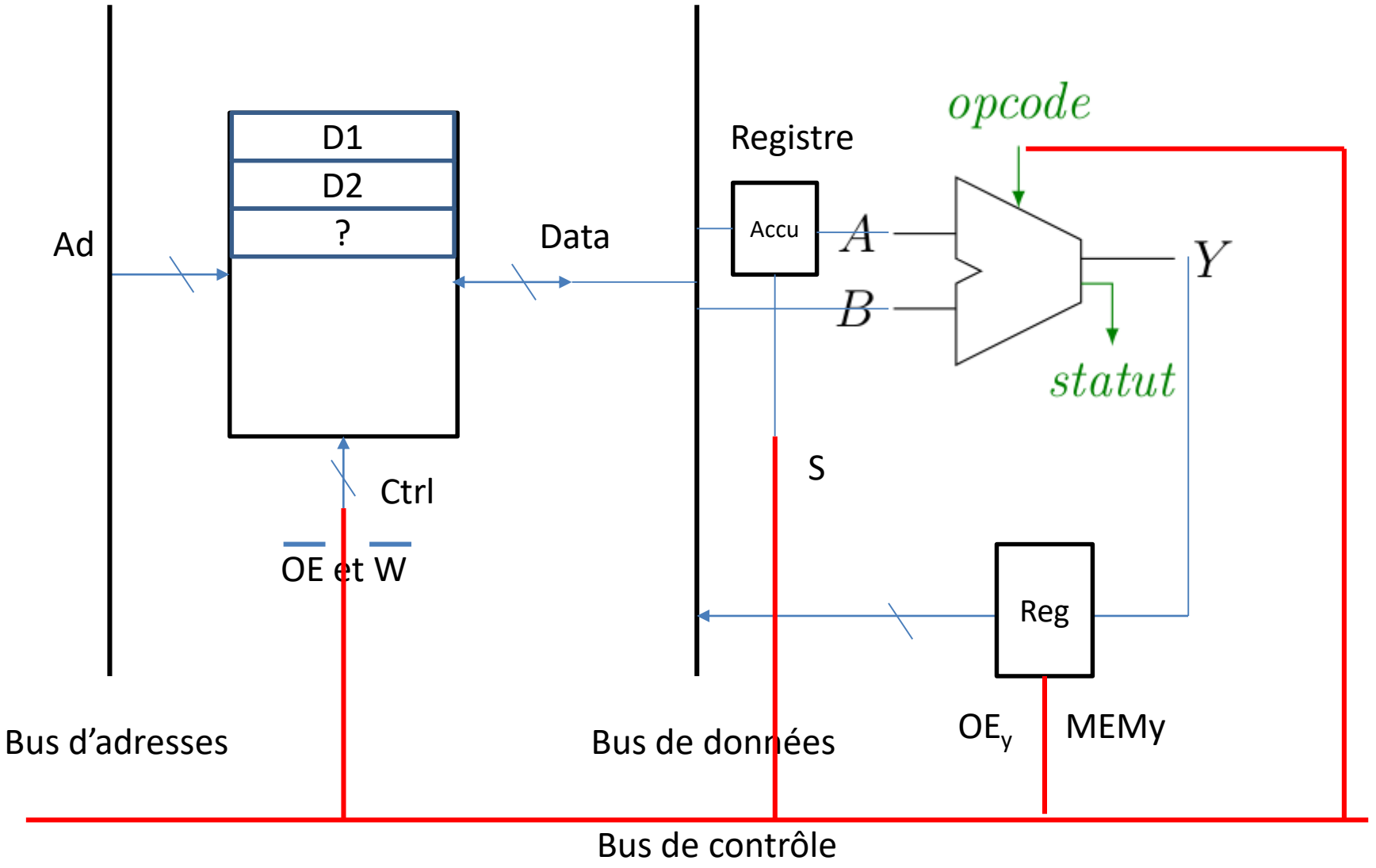
0000,0,1,0,0010,0 mettre D1 sur le bus

0001,0,1,1,0010,0 mettre D2 sur le bus

Problème : on a perdu A

0002,0,0,0,0010,1 écrire le résultat

Modification 1 : accu(mulateur)



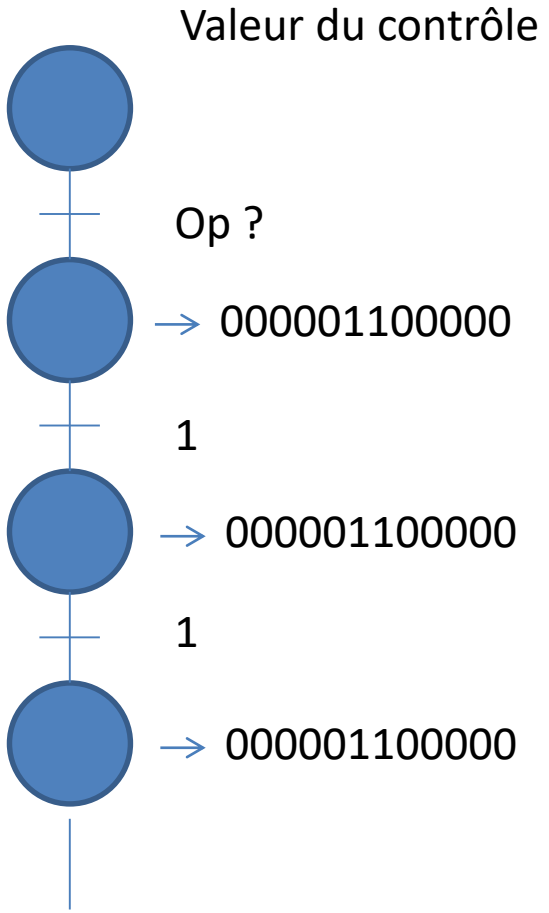
Séquence de contrôle (2)

Adresse, OE/, W/, S, Opcode, MEMy, OEy

0000,0,1,1, 0010,0,0

0001,0,1,0, 0010,1,0

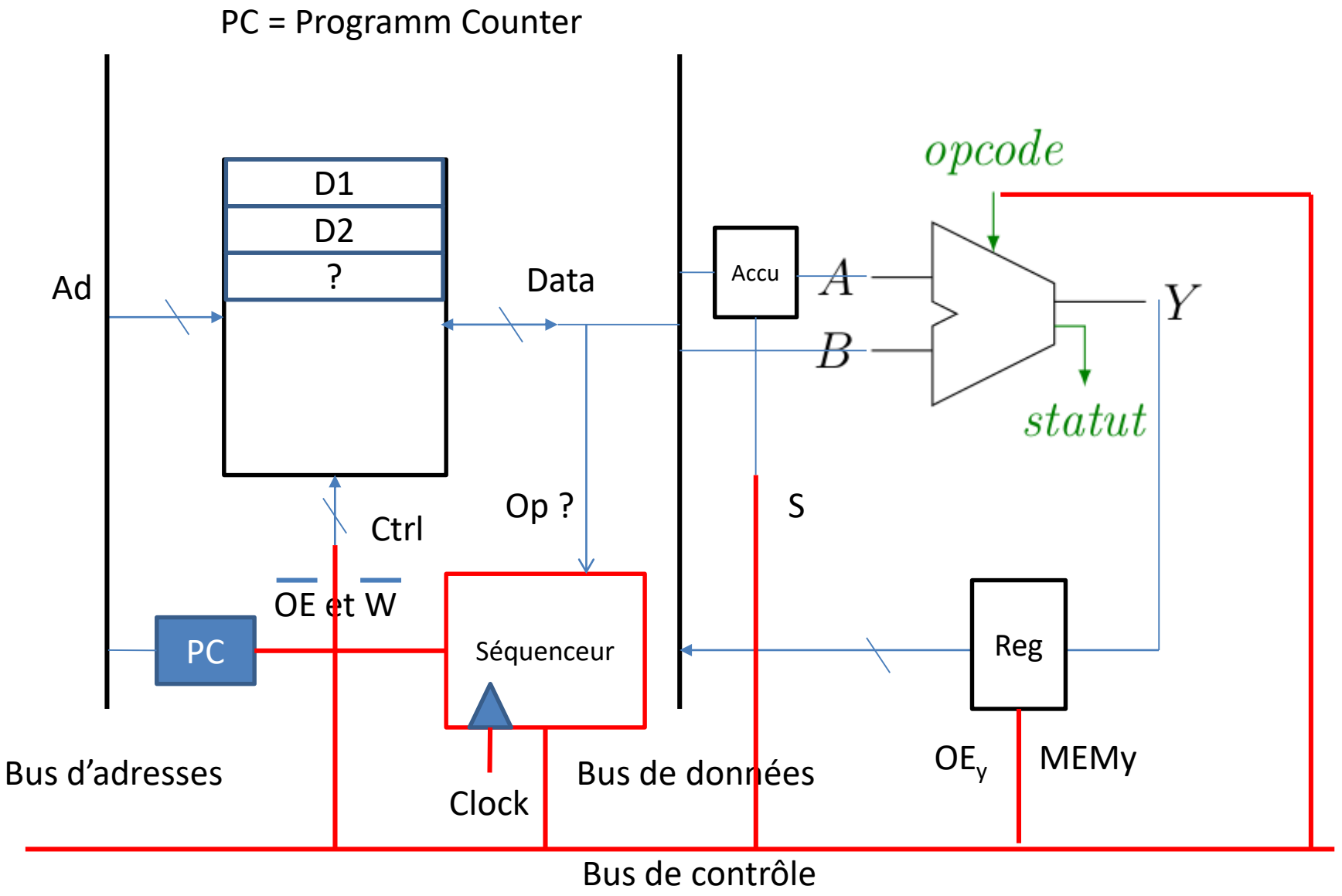
0002,0,0,0, 0010,0,1



Séquenceur associé

➔ Il faut une horloge
➔ Il faut incrémenter l'adresse

Modification 2 : Séquenceur + PC

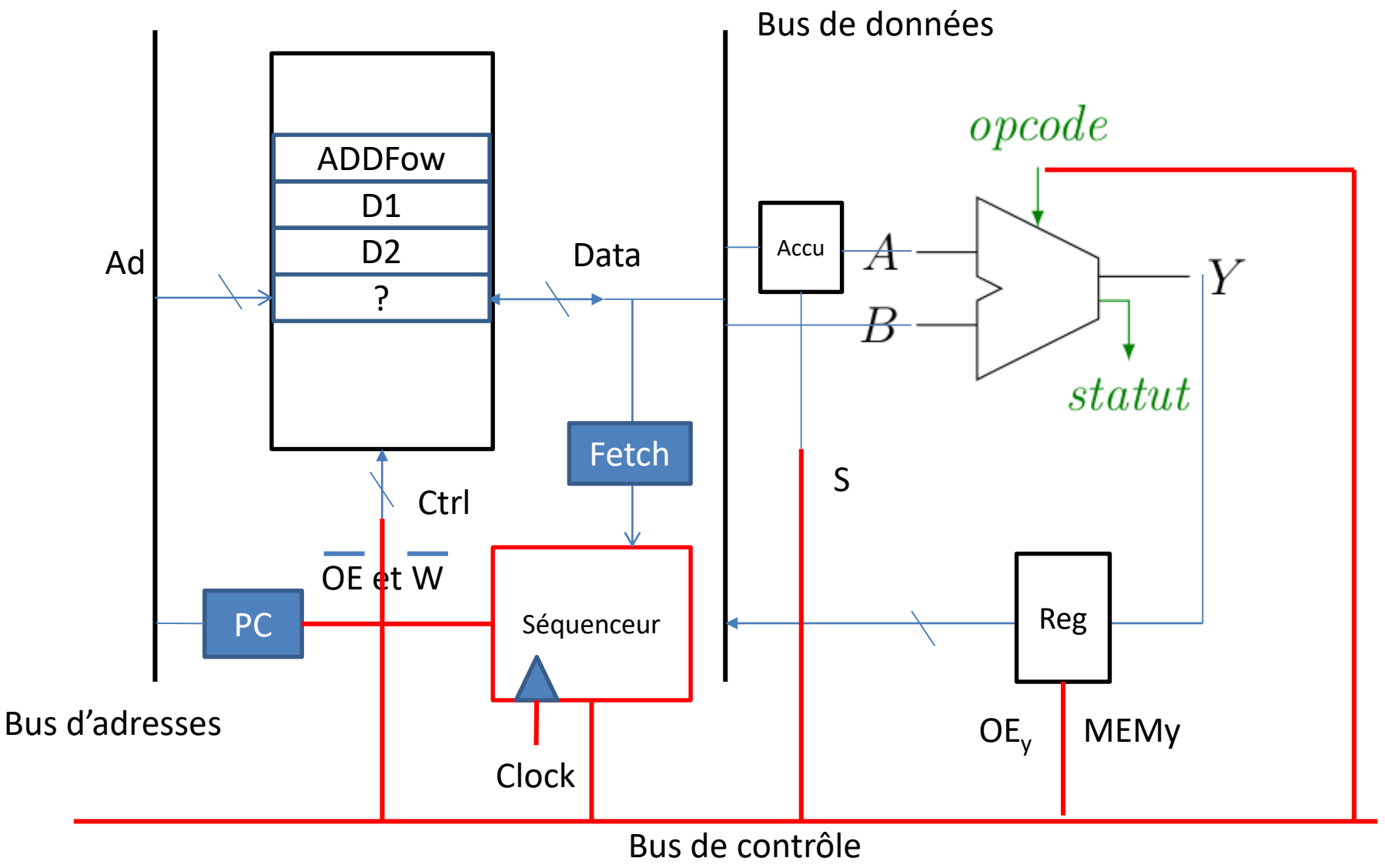


Questions

- Comment détecter l'opération voulue ?
- Comment activer le séquenceur ?
- Comment mettre les données « ailleurs » en mémoire ?

Remarque : Le Compteur de Programme (CP) / Programme Counter (PC)
est aussi appelé Compteur Ordinal (CO) / Ordinal Counter (OC)
(de temps en temps Pointeur d'Instruction)

Modification 3 : Instruction



Fonctionnement

- PC → pointe sur la valeur ADDFow
  - Interprétée comme une instruction
- Fetch → déclenchement du sous-séquenceur (qui pilote le bus de contrôle)
  - PC ←PC+1, lecture data 1 dans accu
  - PC ← PC+1, lecture data 2 sur le bus de données
  - PC ←PC+1, écriture du résultat en mémoire
  - PC ← PC+1
- PC ← pointe sur l'instruction suivante

Fonctionnement

Instruction : mot binaire
Code lisible : mnémonique

- PC ➔ pointe sur la valeur ADDFOW
  - Interprétée comme une instruction
- Fetch ➔ déclenchement du (sous-)séquenceur (qui pilote le bus de contrôle)
  - PC ⬅PC+1, lecture data 1 dans accu
  - PC ⬅ PC+1, lecture data 2 su bus
  - PC ⬅PC+1, écriture du résultat en mémoire
  - PC ⬅ PC+1

Microprogramme de 4 cycles (4 périodes d'horloge)

- PC ⬅ pointe sur l'instruction suivante

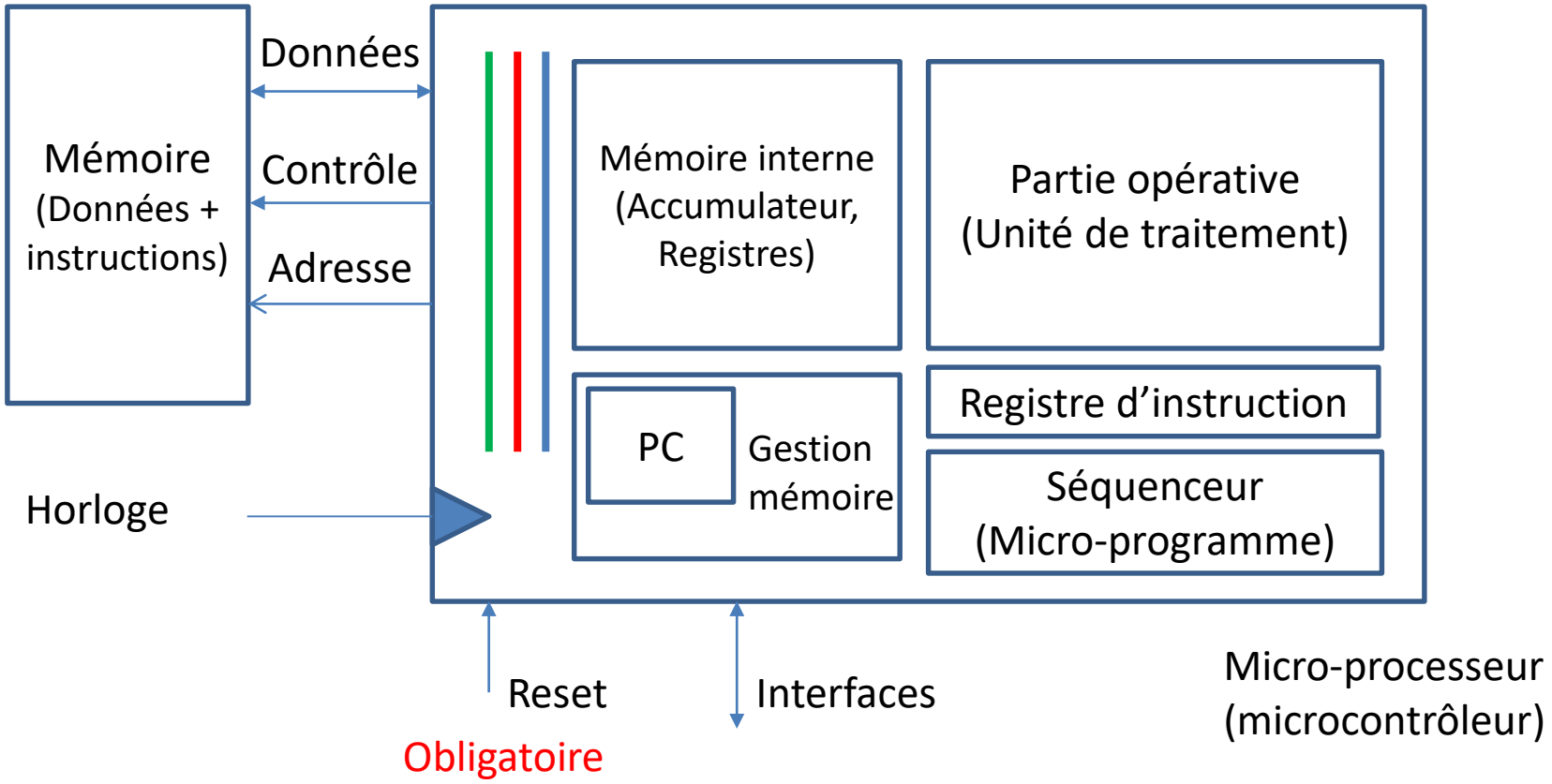
Vocabulaire

- Code machine (ou instruction machine)
  - Mot binaire codant l'instruction
- (Code) mnémonique : pour le programmeur
  - Mot « en clair » pour le manipuler
- Microprogramme
  - Suite de microcommandes réalisant l'instruction
- Assembleur
  - Mnémoniques ➔ codes machines, Calcule sauts
- Désassembleur (fonction inverse de l'assembleur)

Généralisation

Trois bus internes :
- Adresse
- Données
- Contrôle

Architecture de **Von Neuman** micropgrogrammée (CISC)



Exemple extrait de « Comprendre les microprocesseurs »
Gindre & Roux, 1990, Mc Graw-Hill, ISBN : 2-7042-1160-4

Conception d'un mini système

Système simpliste (simplissime) permettant d'illustrer tous les principes utiles à la compréhension des processeurs classiques

Les étapes

- Introduction
- Jeu d’instructions
- Architecture
  - Partie opérative
  - Séquenceur
- Fonctionnement
- Démarrage (boot)
- Evolutions possibles

Introduction

On va définir pour notre processeur

- Les traitements possibles
- Taille/ Format des opérandes
- Capacités de transfert

Ce qui induit

- Jeu d'instructions
- Structure de la partie opérative
- Structure matérielle de dialogue avec l'extérieur

Jeu d'instructions

Trois sous-ensembles

- Traitement des données
  - Exemple : ADD
- Transferts
  - Exemple : MOV
- Modification du déroulement du programme
  - Exemple : BRA

Remarque : les mnémoniques dépendent du constructeur.
L'instruction de branchement pourra se nommer BRA, JMP …
L'instruction de transfert, MOV, LOAD, LD, ST …

Conventions d'écriture

- Exemple : Transfert RAM → Processeur
  - Quelle informations en plus du MOVE ?
  - Convention : Mnémonique source, destination
- Charger registre R1 avec la valeur 05
  - MOVE #05, R1 (R1 ← 05)
- Transfert registre R2 à l'adresse $8000
  - MOVE $8000, R1 (M[8000] ← R2)
- Addition de 05 à R1 stocké dans R1
  - ADD #05,R1 (R1 ← (R1)+05)

Conventions d'écriture

- Exemple : Transfert RAM ➔ Processeur
  - Information en plus du MOVE ?
  - Convention : Mnémonique source, destination
  - Transfert registre R1 avec la valeur 05
  - MOVE #05, R1 (R1 ← 05)
- Transfert registre R2 à l'adresse $8000
  - MOVE $8000, R1 (M[8000] ← R2)
- Addition de 05 à R1 stocké dans R1
  - ADD #05,R1 (R1 ← (R1)+05)

Mnémonique

Equation fonctionnelle

Notations

- # : valeur immédiate
- M[x] : valeur à l'adresse x de la mémoire M
- (R1) : valeur du registre x
- $: hexa

Mode d'adressage (introduction)

Comment accéder aux données

- CLR1
  Action dans mnémonique : adressage implicite
- MOVE #05, R1 (R1 ← 05)
  Codée dans l'instruction : adressage immédiat
- MOVE $8000, R1 (M[8000] ← (R2))
  Adresse dans l'instruction : adressage direct
- MOVE R1,[R2] (M[R2] ← (R1))
  Adresse dans un registre : adressage indirect

Remarques : Les modes d'adressages dépendent des possibilités matérielles des architectures

Jeu d'instructions

Opérations

- ADD : addition
- SUB : soustraction
- AND : et logique
- OR : ou logique

Transferts

- LD : mémoire → reg
- ST : reg → mémoire

Branchements

- BSC : si dépassement
- BSZ : si résultat nul
- BRA : inconditionnel

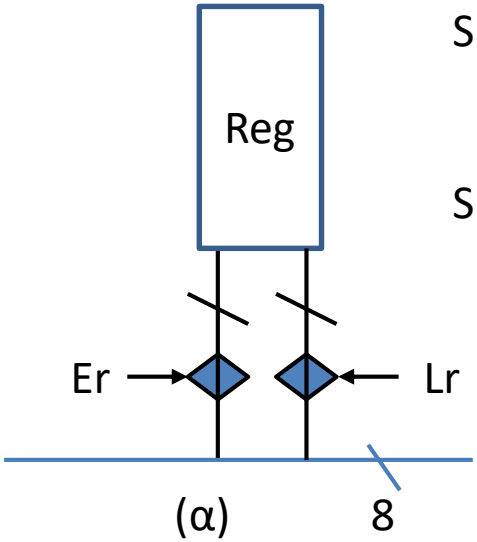
Instructions et opérandes seront codées sur 8 bits
Les bus : 8 bits de large

(Capacité d'adressage : 256 mots)

Simplification de schéma

Les buffers : schéma simplifié

Accès aux registres



SI Er actif

(Reg) ← α

Si Lr inactif

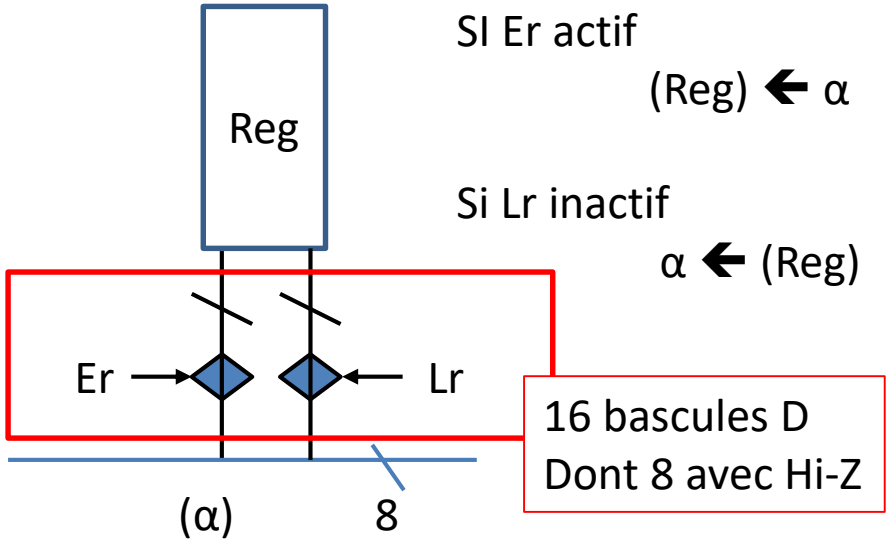
α ← (Reg)

Le bus a la valeur α

Simplification de schéma

Les buffers : schéma simplifié

Accès aux registres

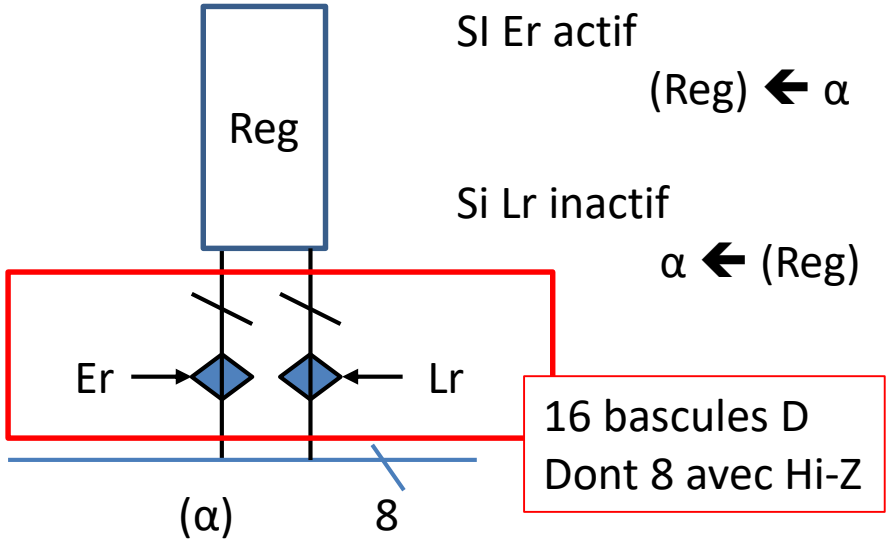


Le bus a la valeur α

Simplification de schéma

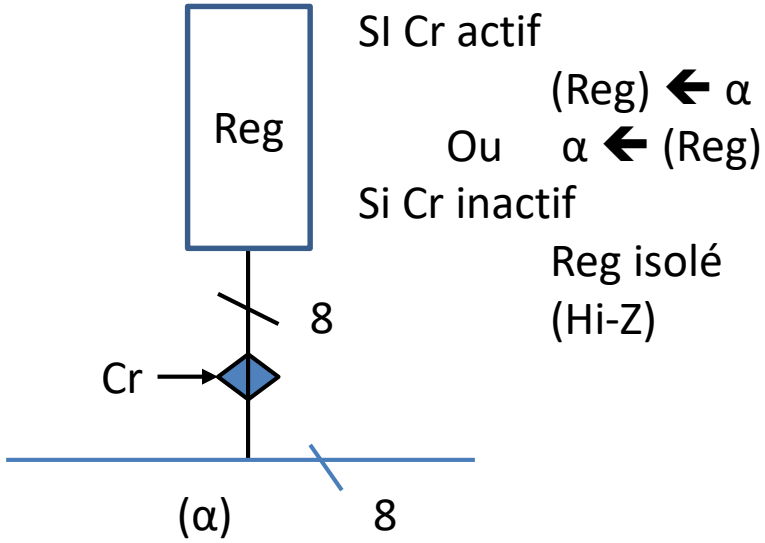
Les buffers : schéma simplifié

Accès aux registres



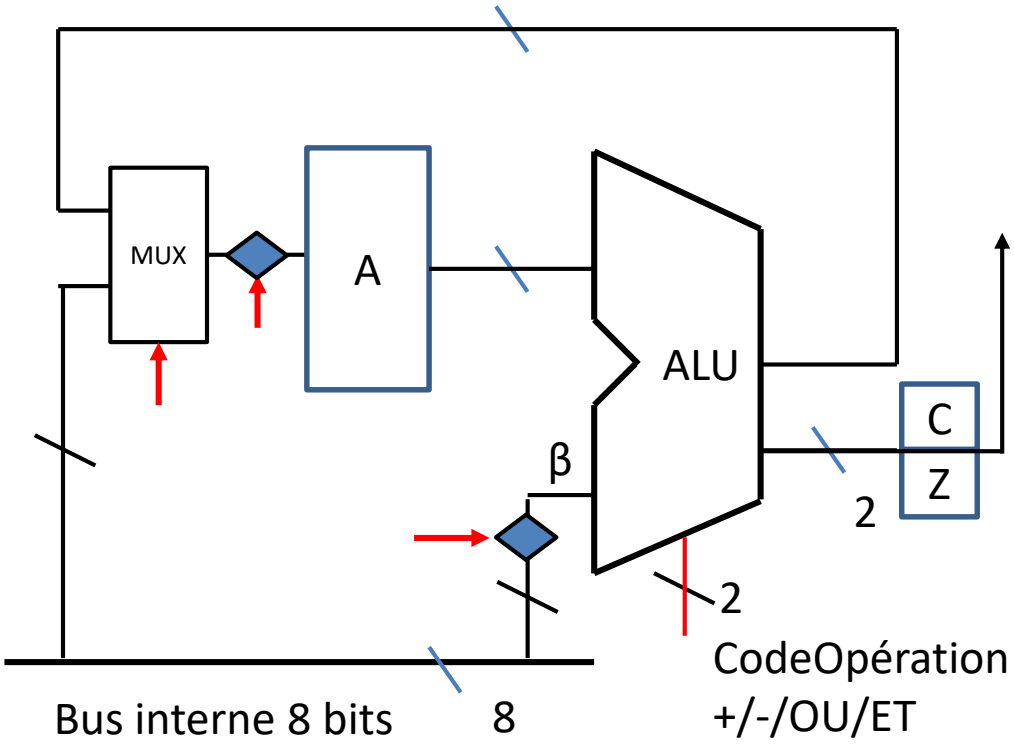
Le bus a la valeur α

Simplifié



Le bus a la valeur α

Structure matérielle
partie opérative (unité de traitement)



Les flèches rouges viennent
de la partie contrôle gérée par le
séquenceur ou microprogramme
(microcommandes)

Vers la partie contrôle
pour les test conditionnels

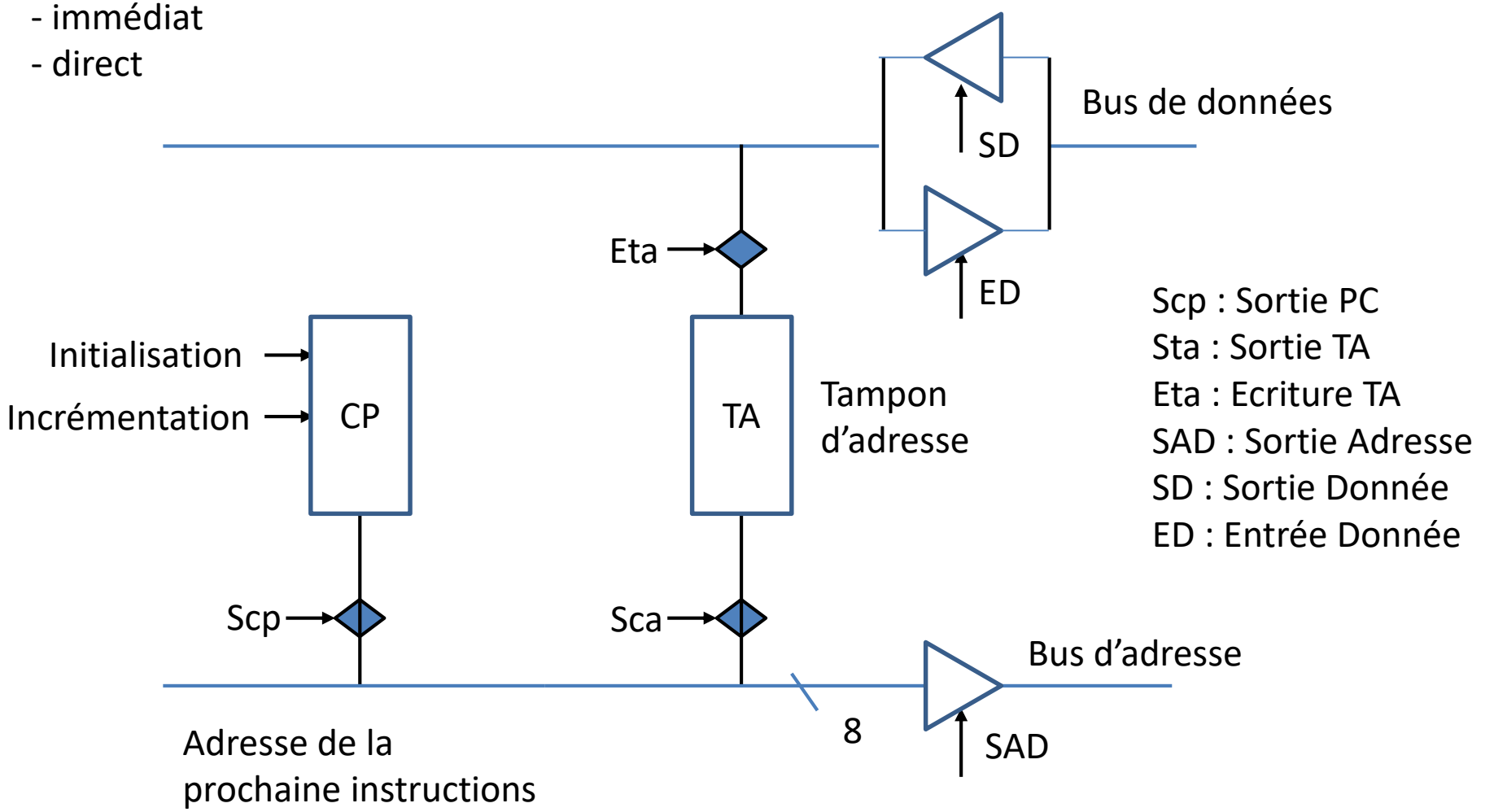
Registre (d'indicateurs) d'état
(Flags)

Le registre « collé » à l'ALU
s'appelle l'accumulateur ou accu

Equations fonctionnelles possibles
A ← op(A)
A ← (A) op β

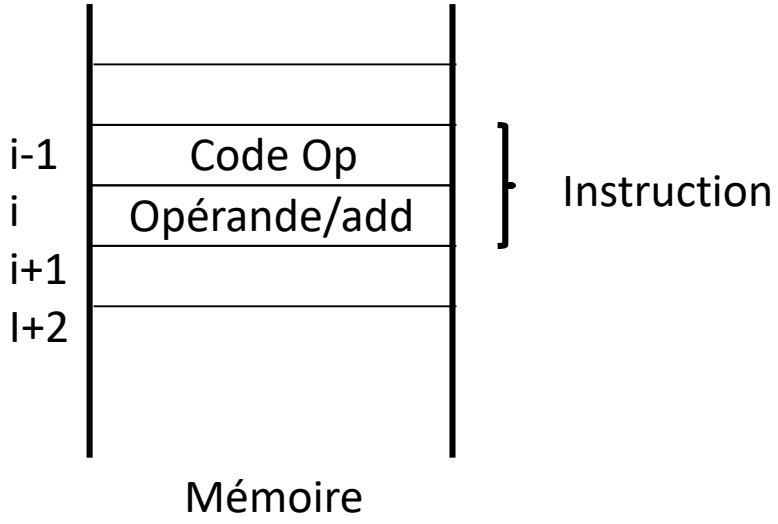
Structure matérielle
gestion des adresses

Adressages :
- implicite
- immédiat
- direct



Structure des instructions

- Chaque instruction est codée sur 2 octets
  - Code opération : pour le registre d'instruction (définit le mode d'adressage)
  - Opérande ou adresse



Instructions : opérations

- ADD # <valeur> (A ← (A)+<valeur>)
- ADD <adresse> (A ← (A)+M[<valeur>])

Même structure pour SUB, OR, ET

Remarque : On dit qu'un jeu d'instructions est orthogonal quand toutes les instructions peuvent utiliser tous les modes d'adressage

Instructions : transferts

- LD # <valeur> (A ← <valeur>)
- LD <adresse> (A ← M[<adresse>]
- ST <adresse> (M[<adresse>] ← (A))

Remarque : On dit qu'un jeu d'instructions est orthogonal quand toutes les instructions peuvent utiliser tous les modes d'adressage

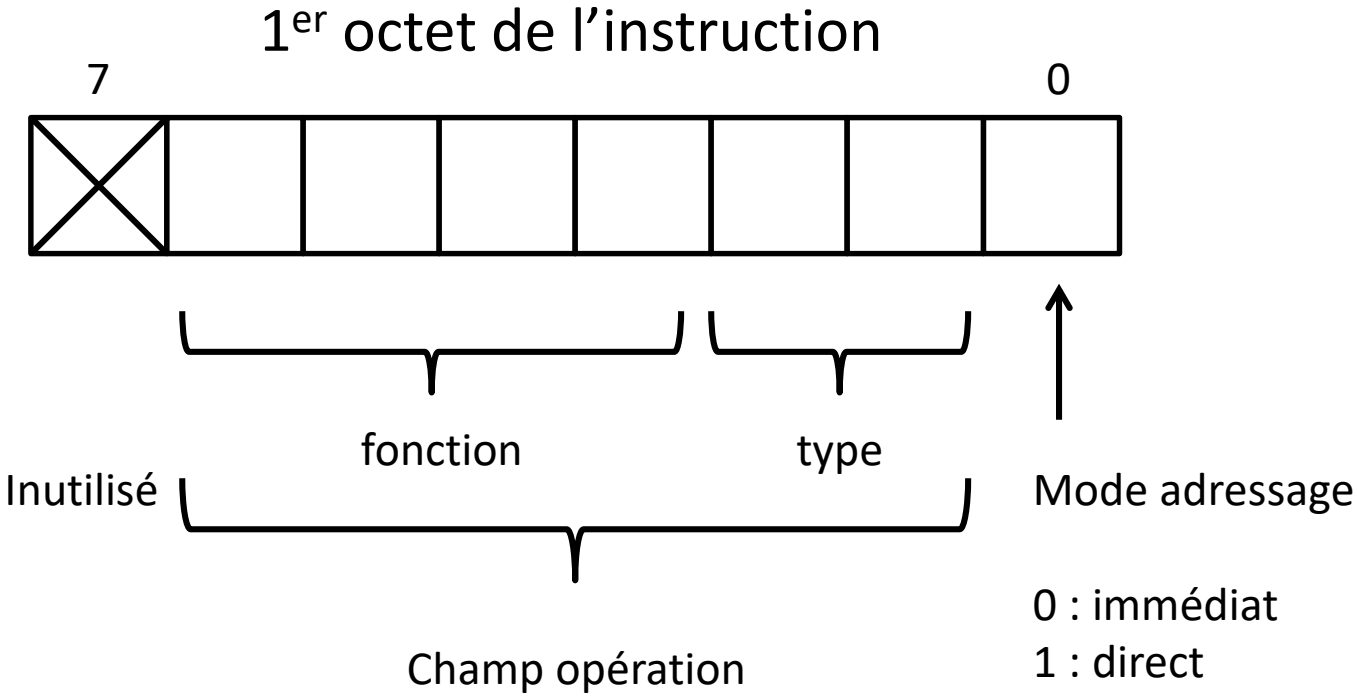
Instructions : branchements

- BRA <adresse> (CP ← <adresse>)
- BSC <adresse> si C=1 : CP ← <adresse>
  sinon CP ← CP+1
- BSZ <adresse> si Z=1 : CP ← <adresse>
  sinon CP = CP +1

Remarque : On dit qu'un jeu d'instructions est orthogonal quand toutes les instructions peuvent utiliser tous les modes d'adressage

Codage des instructions (1)

- 9 instructions / 2 modes d'adressage



Remarque: Ce format en « champs » facilite le décodage par le séquenceur

Codage des instructions (2)

Codes fonction

ADD : 0001

SUB : 0010

AND : 0100

OR : 1000

LD : 0001

ST : 0010

BRA : 0001

BSZ : 0010

BSC : 0100

Codes type

ALU : 01

Transfert : 10

Branchement : 11

Codage 1er octet : exemples

ADD # <valeur> : x 0001 01 0

ADD <adresse> : x 0001 01 1

BRA <adresse> : x 0001 11 x

Codage des instructions (3)

Codage instruction : exemples

ADD #05 : 00001010 00000101 = 0A 05

ADD $83 : 00001011 10000011 = 0B 83

BRA $54 : 00001110 01010100 = 0E 54

Codage : synthèse

	Immédiat	Direct
ADD	0A	OB
SUB	12	13
AND	22	23
OR	42	43
LD	OC	OD
ST	X	15
BRA	X	0F
BSC	X	17
BSZ	X	27

Structure matérielle
séquenceur ou unité de commande

Circuit séquentiel complexe

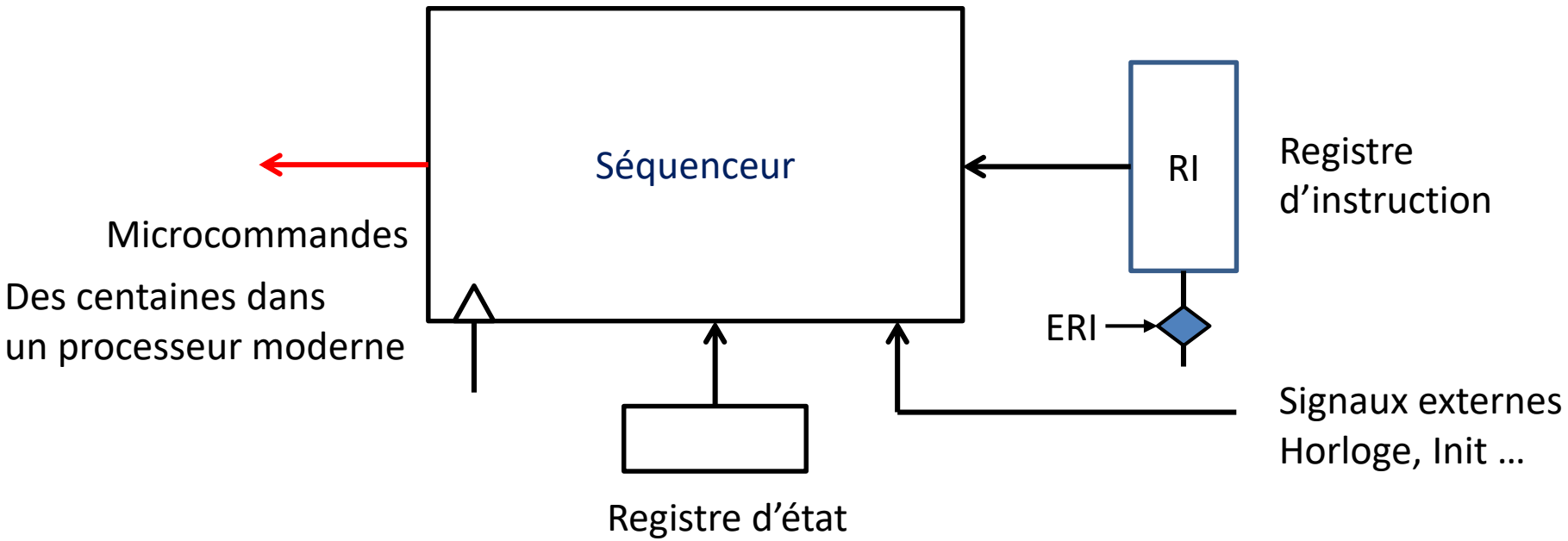
- Fonctionne au rythme de l'horloge interne
  (peut être différente de l'horloge externe)
- Définit la suite des microcommandes
  - Initialisation
  - Recherche d'instruction (fetch)
  - Interprétation de l'instruction

Microcommandes

- Entrées séquenceur
  - Instruction courante et opérande
  - Prise en compte du registre d'état
  - Prend en compte les signaux externes
- Sorties
  - Activation et séquencement des accès aux registres
  - Fabrication du code opération (ALU)
  - Calcul des adresses de saut
  - Signaux de gestion externe

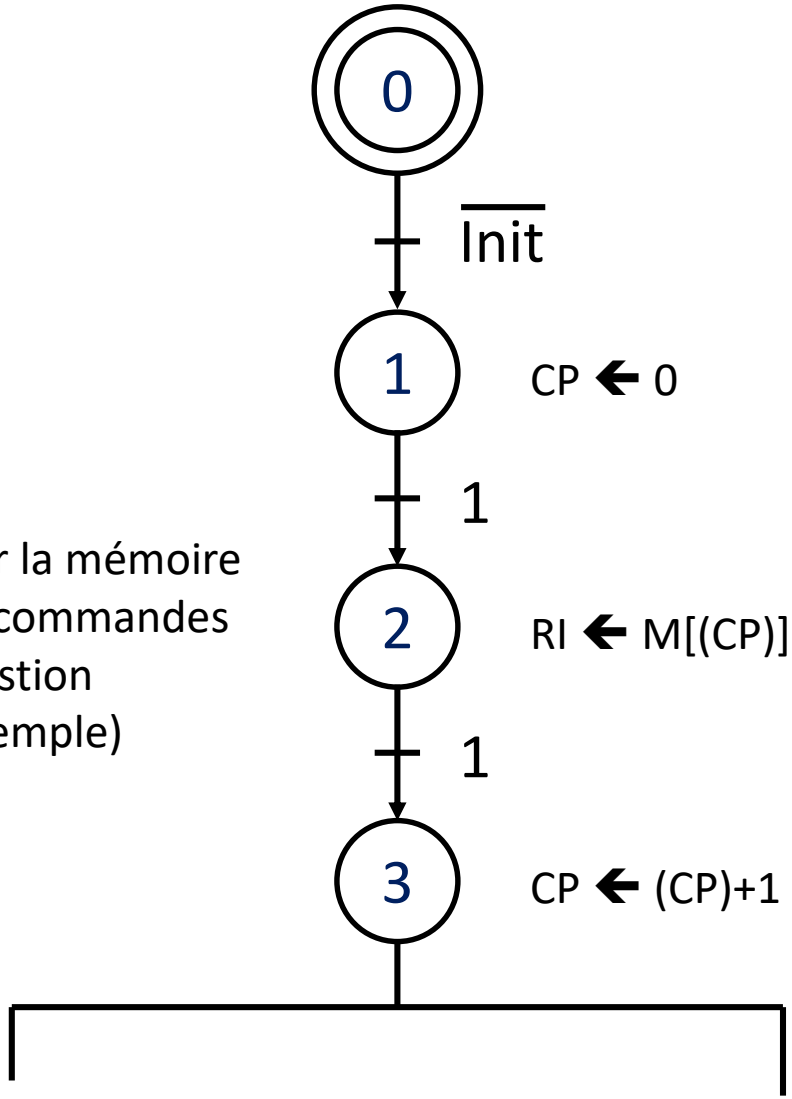
Séquenceur

Décoder instruction, générer microcommandes



Méthodes de réalisation :
- système câblé (complexe) par synthèse logique des graphes de fluence
- décodage conditions, mémoire de microcommande et compteur (simple)

Initialisation

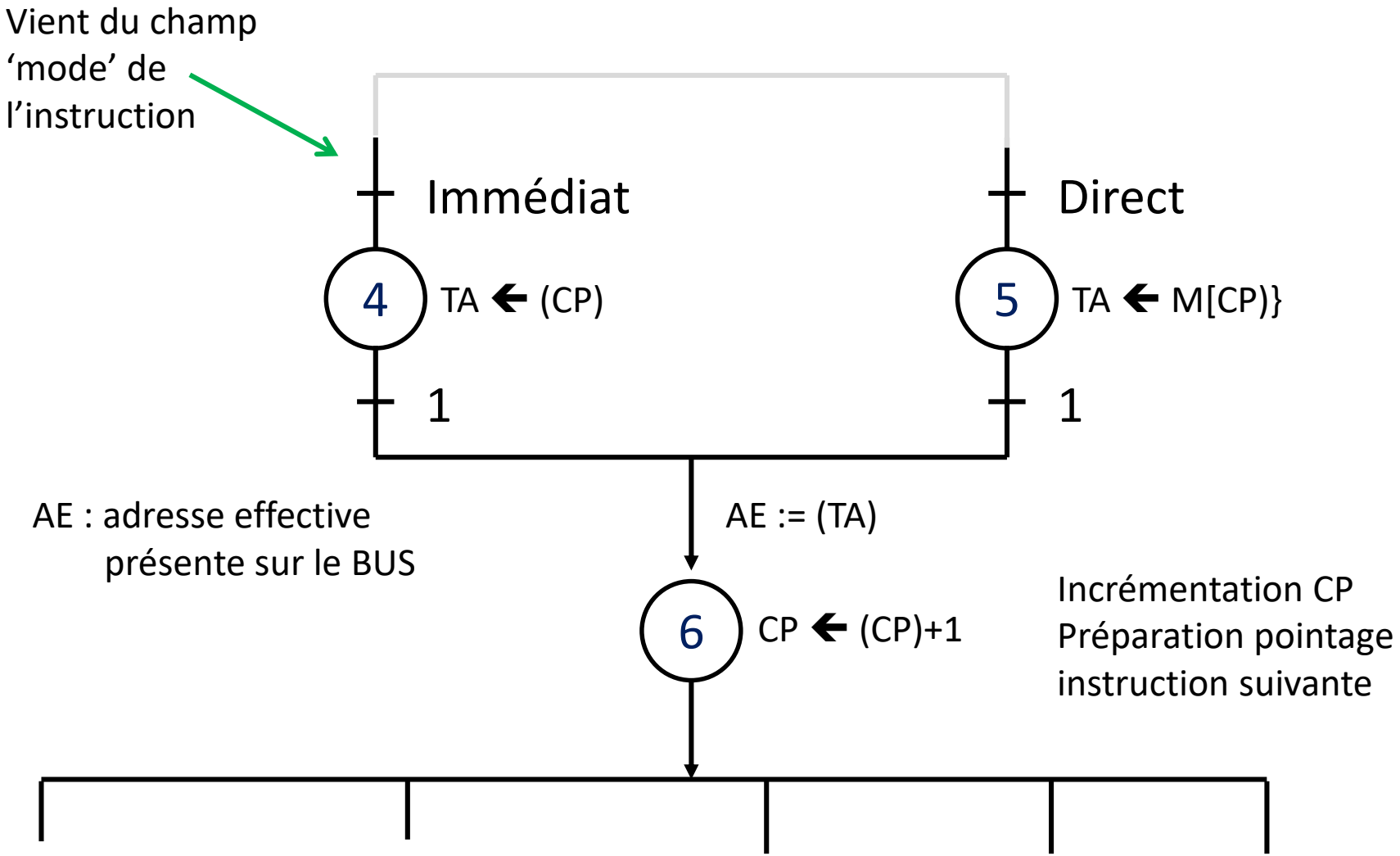


Note : les opérations sur la mémoire doivent positionner les commandes externe sur le bus de gestion d'interface (R/W par exemple)

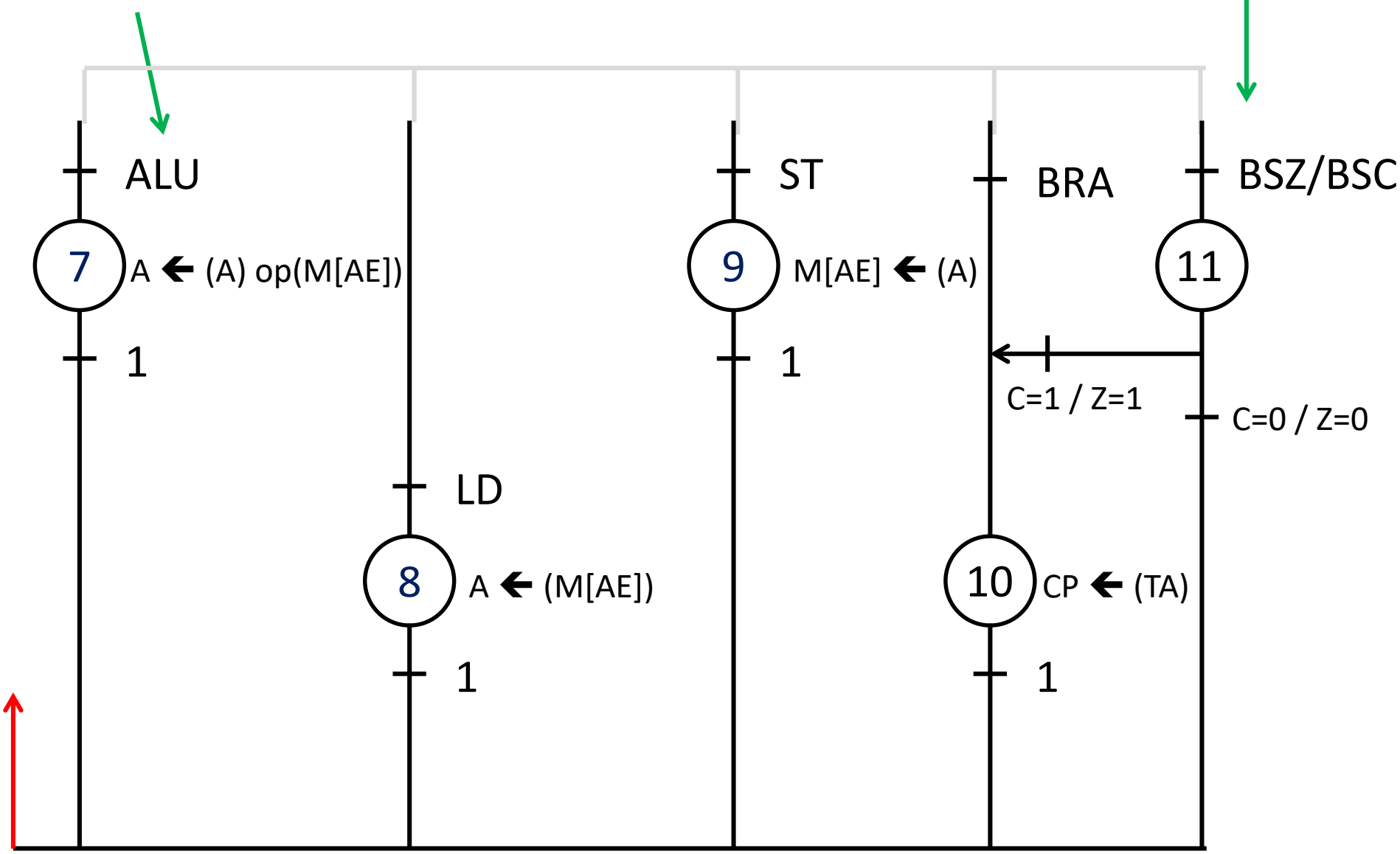
Recherche instruction

Incrémentation CP

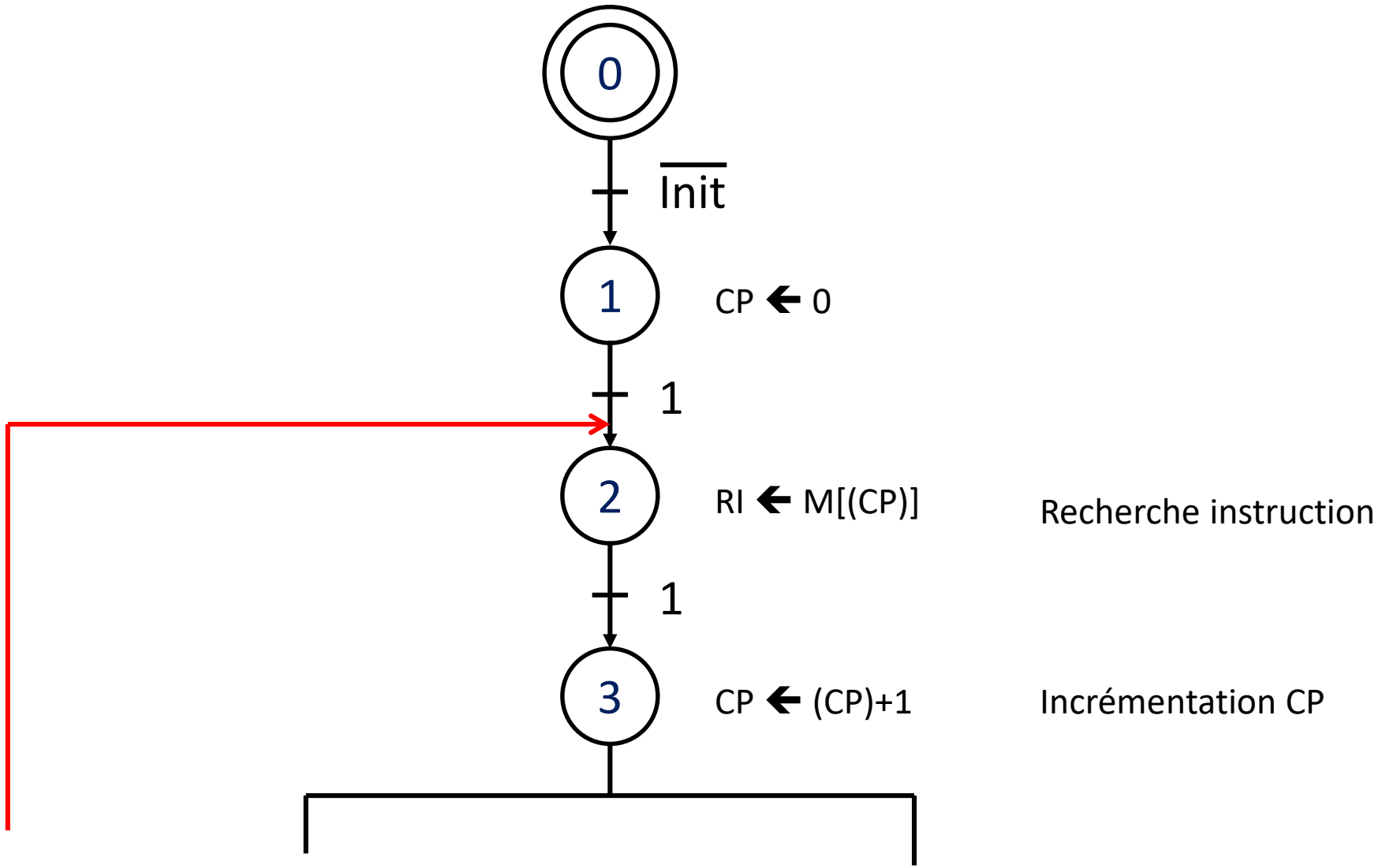
Mode d'adressage



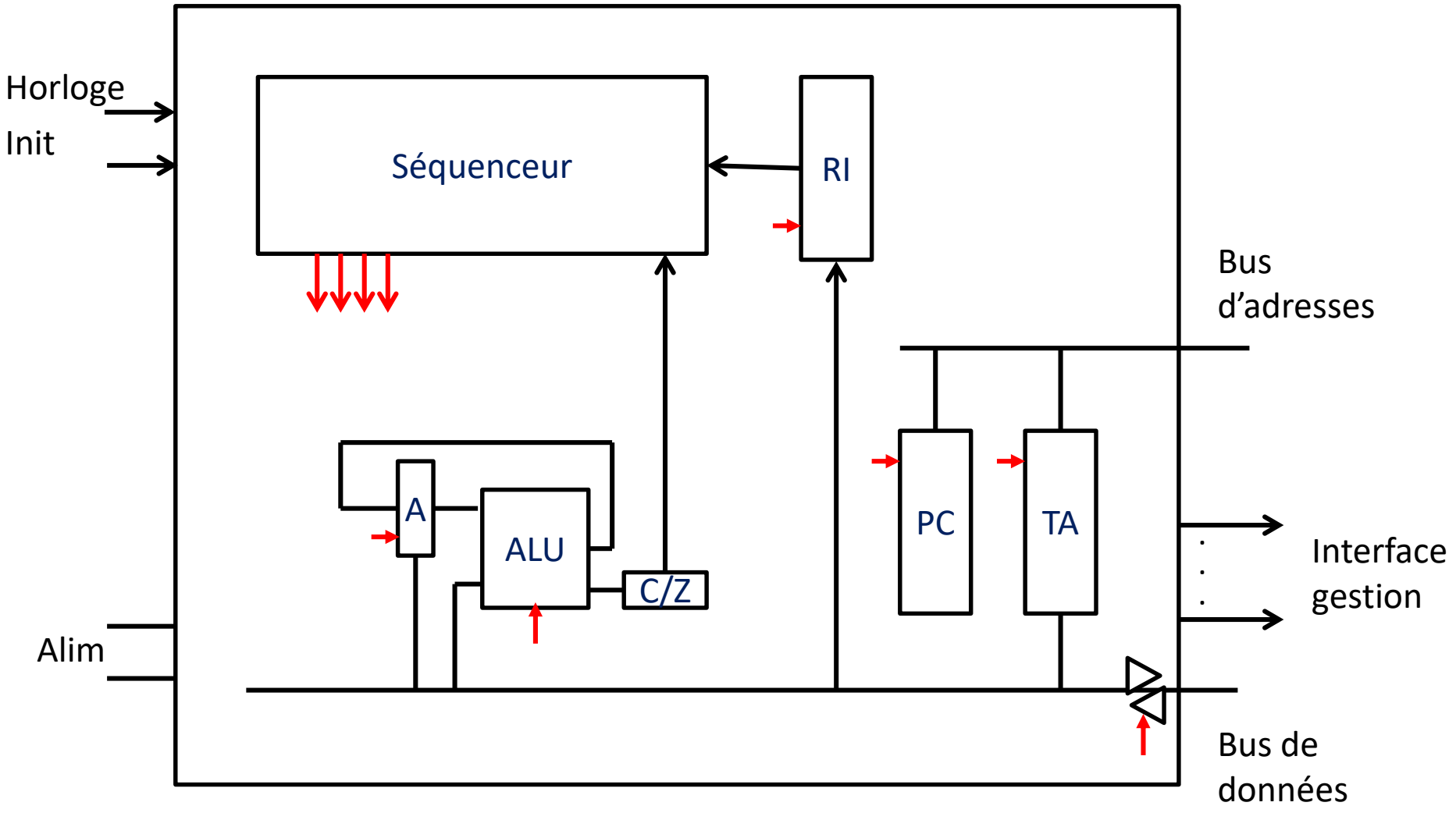
Fonctions


Vient du champ ‘fonction’ de l’instruction
Deux branches parallèles
ALU
7
A ← (A) op(M[AE])
1
LD
8
A ← (M[AE])
1
ST
9
M[AE] ← (A)
1
BRA
10
CP ← (TA)
1
BSZ/BSC
11
C=1 / Z=1
C=0 / Z=0

Rebouclage


0
$\overline{Init}$
1
CP ← 0
1
2
RI ← M[(CP)]
Recherche instruction
1
3
CP ← (CP)+1
Incrémentation CP

Assemblage


Horloge
Init
Séquenceur
RI
Bus
d'adresses
A
ALU
C/Z
PC
TA
Interface
gestion
Alim
Bus de
données

Archi de Von Neumann : définitions

- Une unité de traitement (UT)
- Une mémoire (données + instructions)
- Un canal d'échange (BUS) partagé

1- UT : opérations élémentaires câblées

Traitement déclenché par une commande appelée instruction. L'ensemble des instructions nommé le jeu d'instructions.

2- Exécution des tâches complexes : suite d'instructions exécutée séquentiellement nommée programme

3- Programme et données stockés dans une mémoire
= machine à programme enregistré.

4- Certaines instructions : ruptures de séquences conditionnelles

Ce point fondamental en fait la nouveauté en 1945

Fonctionnement (1)

- On reprend le besoin du début : Z = X + Y
- Programme connaît 3 adresses : AdX, AdY,AdZ
- Spécification

  Programme « addtion8bits »

  Variable X,Y,Z : octet

  Après initialisation

  Début

  Z := X +Y

  Attendre ici

  Fin

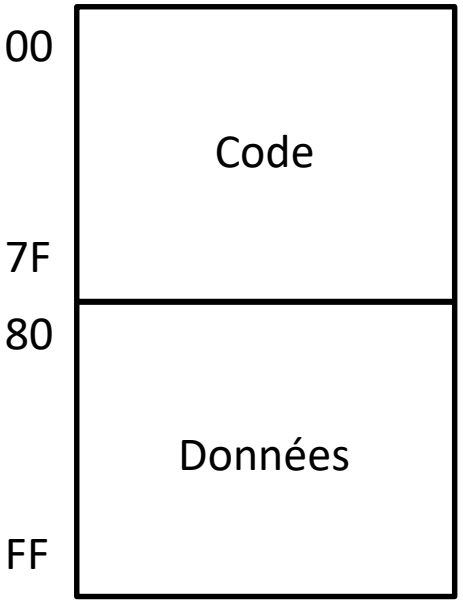
Fonctionnement (2)

- Compte tenu des capacités « hardware »
  - Chargement de A avec X
  - Addition avec Y
  - Stockage du résultat
  - Attente

- Après initialisation PC = 0
  - Le programme commence à cette adresse

Fonctionnement (3)

- Utilisation de la mémoire : cartographie d'usage



En Hexa :

De 00 à 7F : programme (128 mots)
De 80 à FF : données (128 mots)

Choix arbitraire :
X ➔ AdX = 80
Y ➔ AdY = 81
Z ➔ AdZ =82

Fonctionnement (4)

- Code programme en langage assembleur (mnémoniques) Ecrit sur un autre ordinateur

Directives d'assemblage

```
        ORG   00          Adresse début programme
AdX     EQU   80          Alias pour définir localisation
AdY     EQU   81
AdZ     EQU   82

Deb:    LD    AdX         Charge l'accu par X=M(AdX)
        ADD   AdY         Addition de l'accu avec Y=M(AdY)
        ST    AdZ         Stocke résultat en M(AdZ)
Wait:   BRA   Wait        Attente infinie
```

Fonctionnement (5)

- Utilisation d'un cross assembleur
  (outil sur machine 1 pour fabriquer un code pour machine 2)
  Remplace les alias
  Remplace les mnémoniques par les codes machine
  Calcule les adresses

			Ad	**Inst**
Deb:	LD	AdX	00	0D 80
	ADD	AdY	02	0B 81
	ST	AdZ	04	15 82
Wait:	BRA	Wait	06	0F 06

Programme

OD
80
0B
81
15
82
0F
06

Fonctionnement (6)

- Implantation du programme et des données
  - Si prévu : téléchargement depuis l'extérieur
  - Préparation mémoire à l'extérieur puis report

    Type de mémoire ?

Problème réglé plus tard (par les gens du hard)

- Fonctionnement interne ?
  - Comment le programme fait fonctionner le processeur

Conventions pour la suite

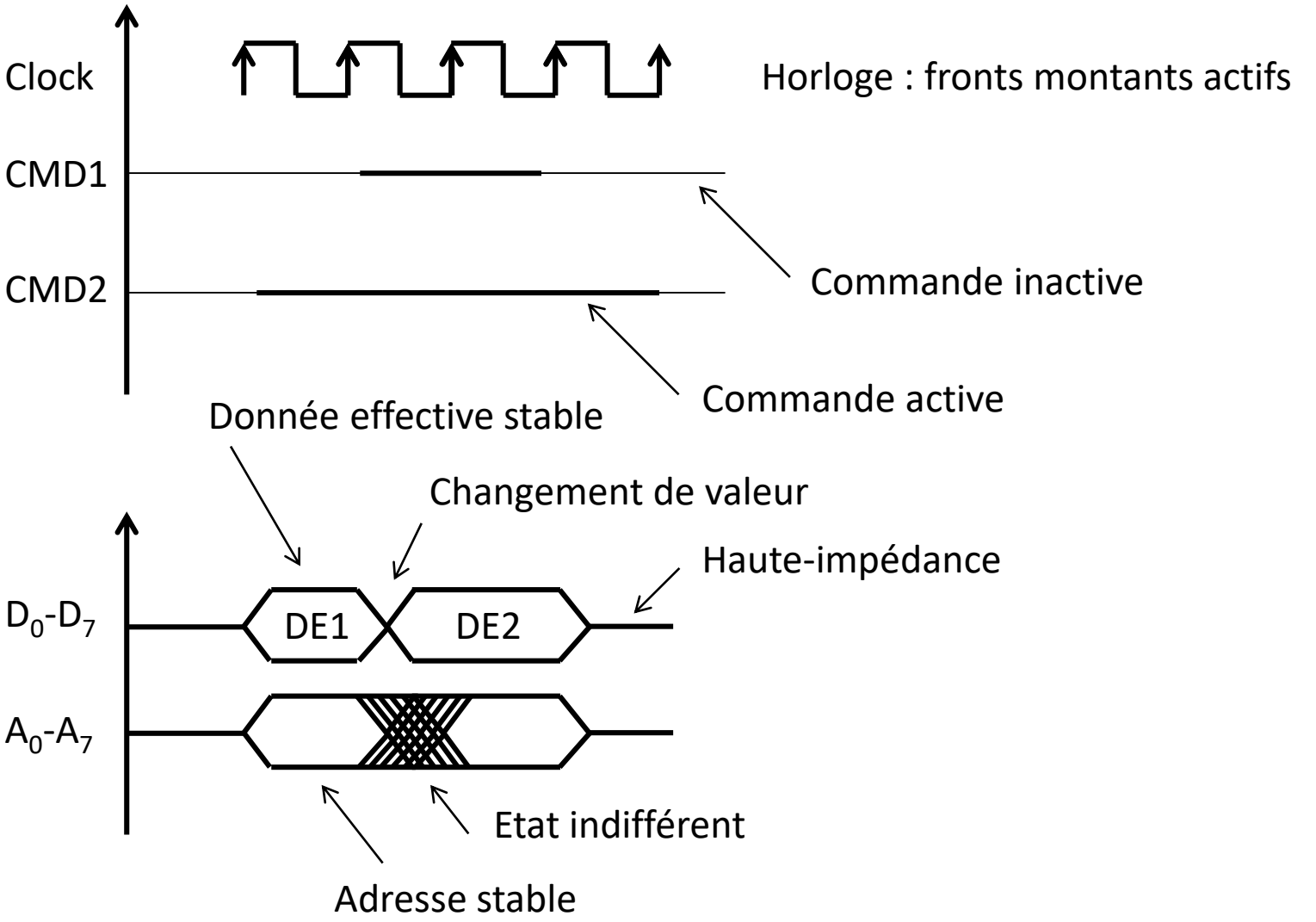
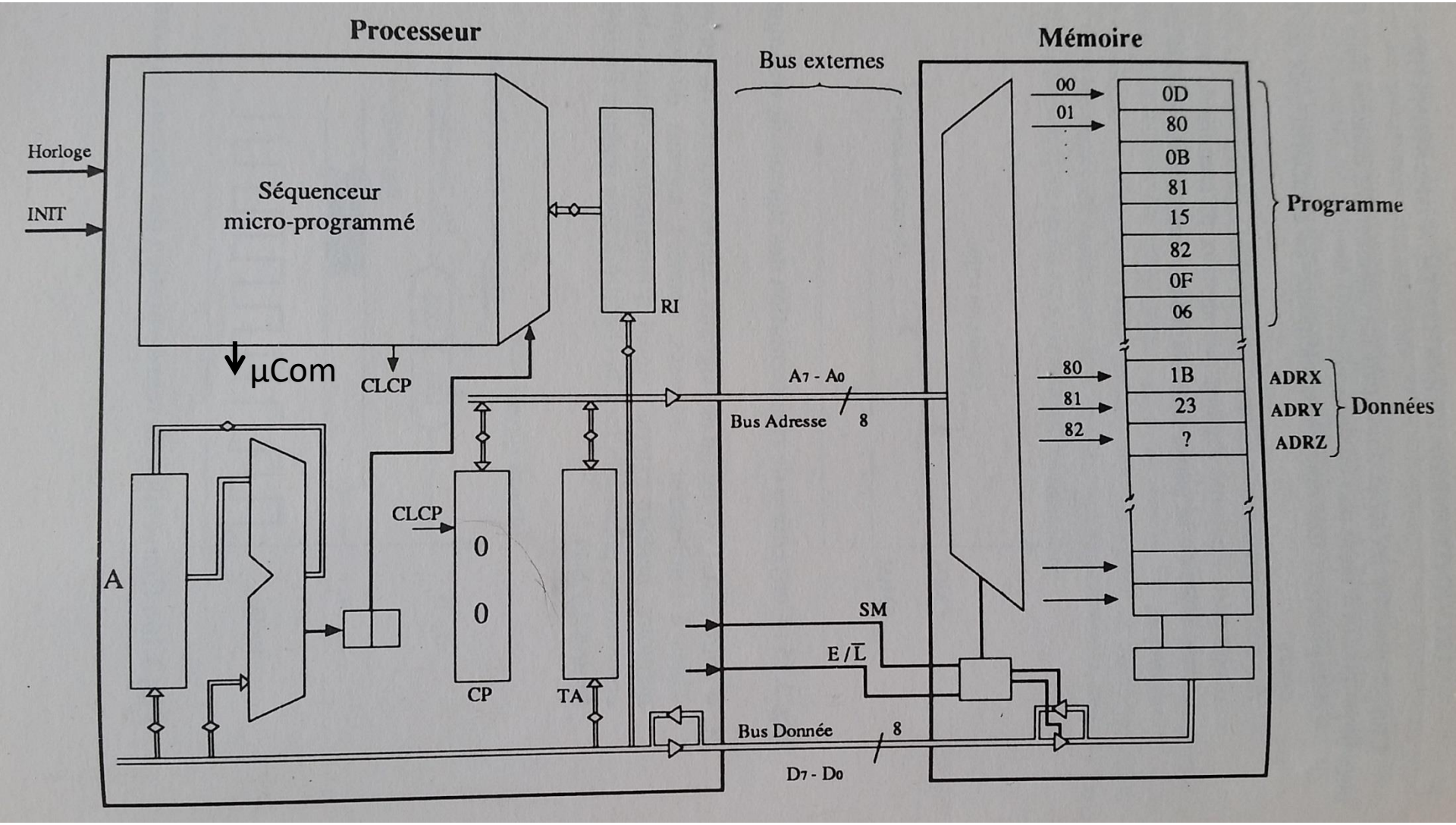


Schéma global



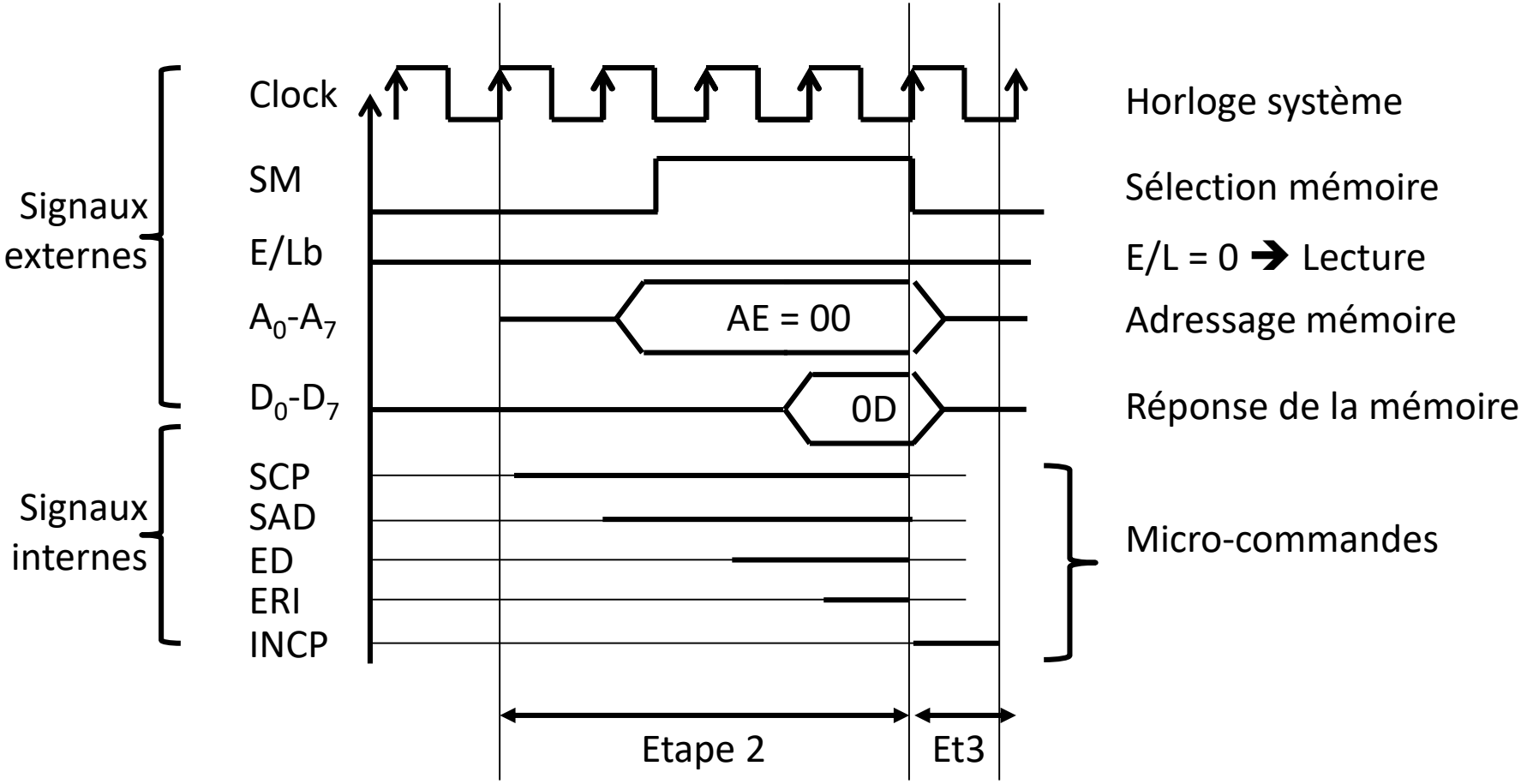
SM : Sélection mémoire ($\overline{CE}$: chip enable) E/$\overline{L}$: 1=Ecriture/0=Lecture (R/$\overline{W}$)

Initialisation et fetch

- Etape 1
  - CP ←0 : microcommande CLCP actif (CLCP)
- Etape 2 (6 phases)
  - CP sur le bus interne : SCP
  - Activation buffer sortie adresse : SAD
  - Accès mémoire : SM et L
  - Activation buffer mémoire en entrée : ED
  - Activation écriture RI : ERI
  - Désactivation toutes commandes

CP ←CP + 1

- Etape 3 : Incrémentation PC : INCP



La suite

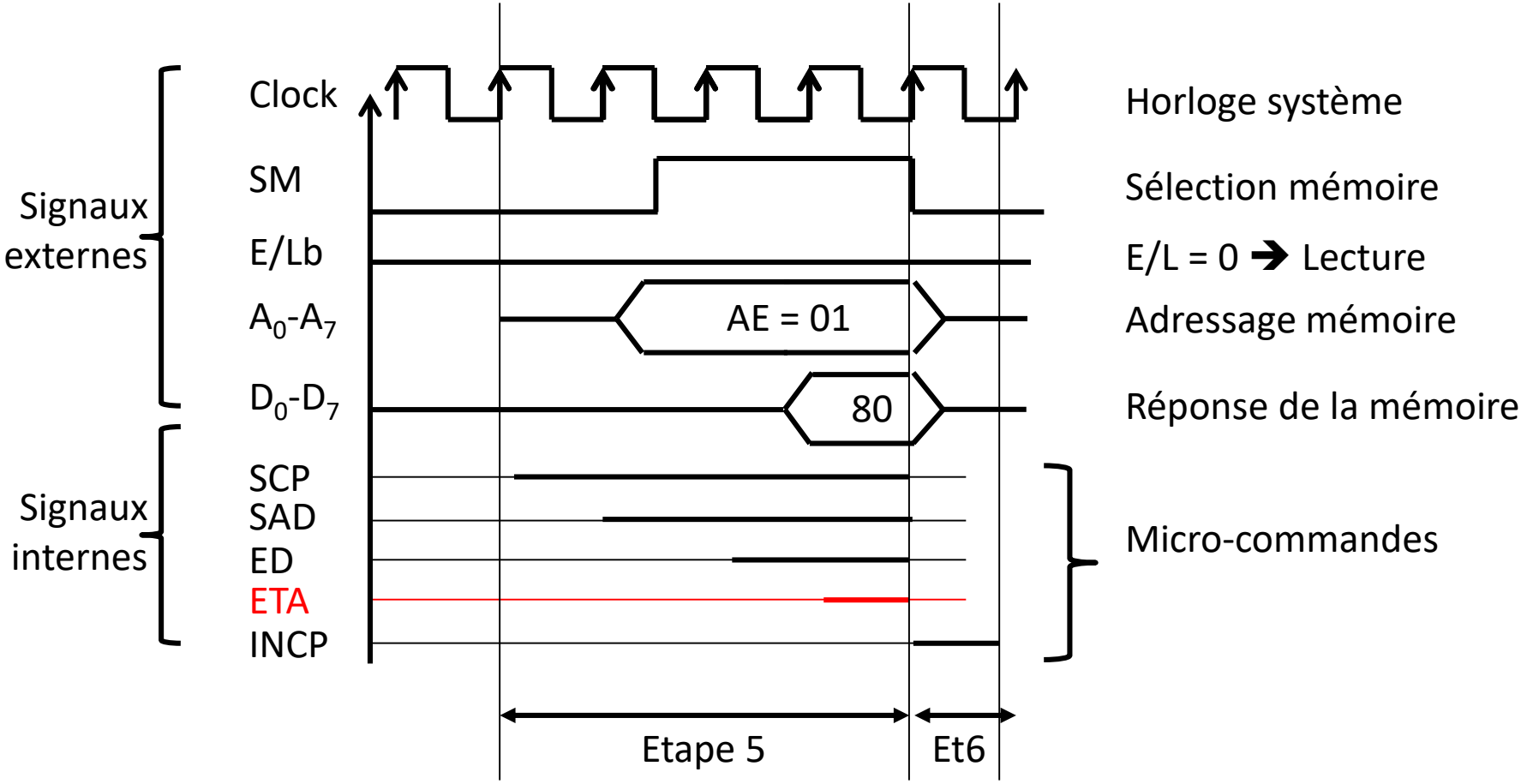
- A la fin de l'étape 3 : (CP) = 01 et (RI) = 0D

  ➔ Etape suivante : 5 (adressage direct)

- Etape 5 : TA ← (M[(CP)] (6 phases)
  - CP sur le bus interne : SCP
  - Activation buffer sortie adresse : SAD
  - Accès mémoire : SM et L
  - Activation buffer mémoire en entrée : ED
  - Activation registre temporaire adresse : ETA
  - Désactivation toutes commandes

CP ←CP + 1

- Etape 6 : Incrémentation PC : INCP



La suite

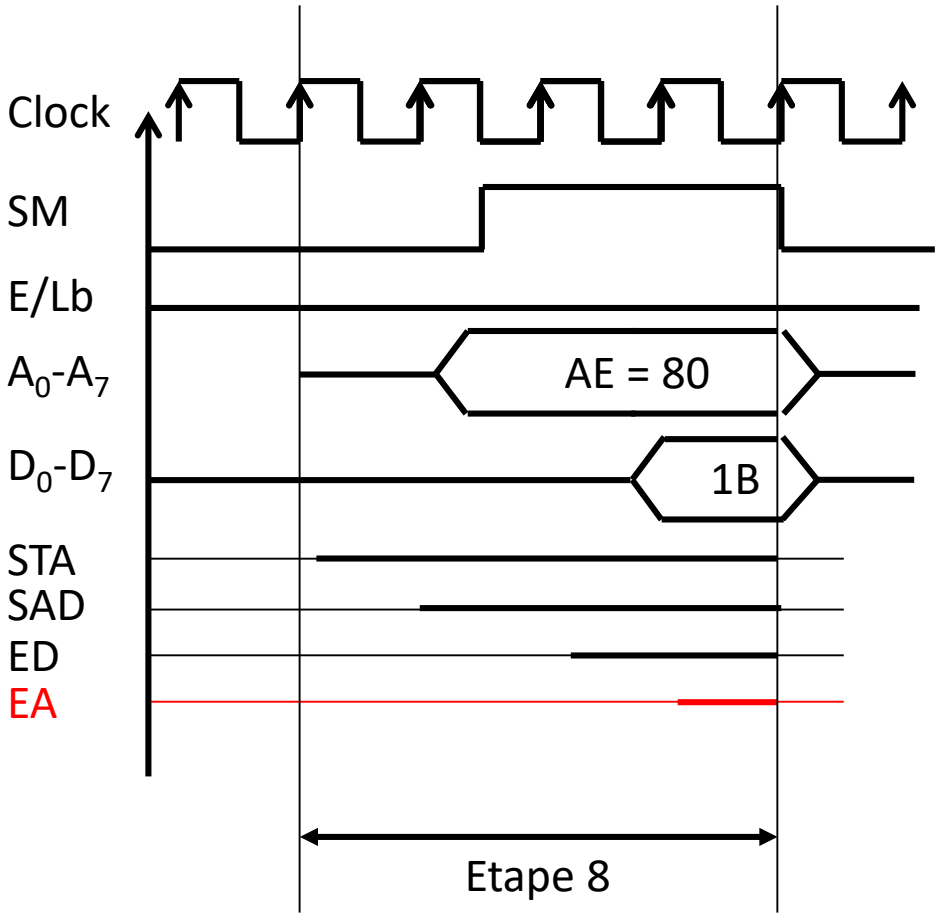
- A la fin de l'étape 6 : transfert/chargement

  ➔ Etape suivante : 8

- Etape 8

  A ← (M[(AE)]

  – Rangement dans l'accumulateur : Activation EA



Rebouclage

- On repasse étape 2, 3, 5, 6
- Etape 7 : A ← (A) + (M[(AE)])

Et ainsi de suite …

Remarques

- Durée des instructions
  - Chaque instruction : 5 étapes
  - BSZ/BSC : loupé = 5 étapes / réussi = 6 étapes
- Chaque étape gère une séquence de signaux
  - Au rythme de l'horloge
- Une seule étape active : optimalité ?
- Le programme commence à l'adresse 0
- Ne prend pas en compte les signaux extérieurs
  - Sauf Init
  - Pas d'entrée-sorties (clavier, souris, écran …)