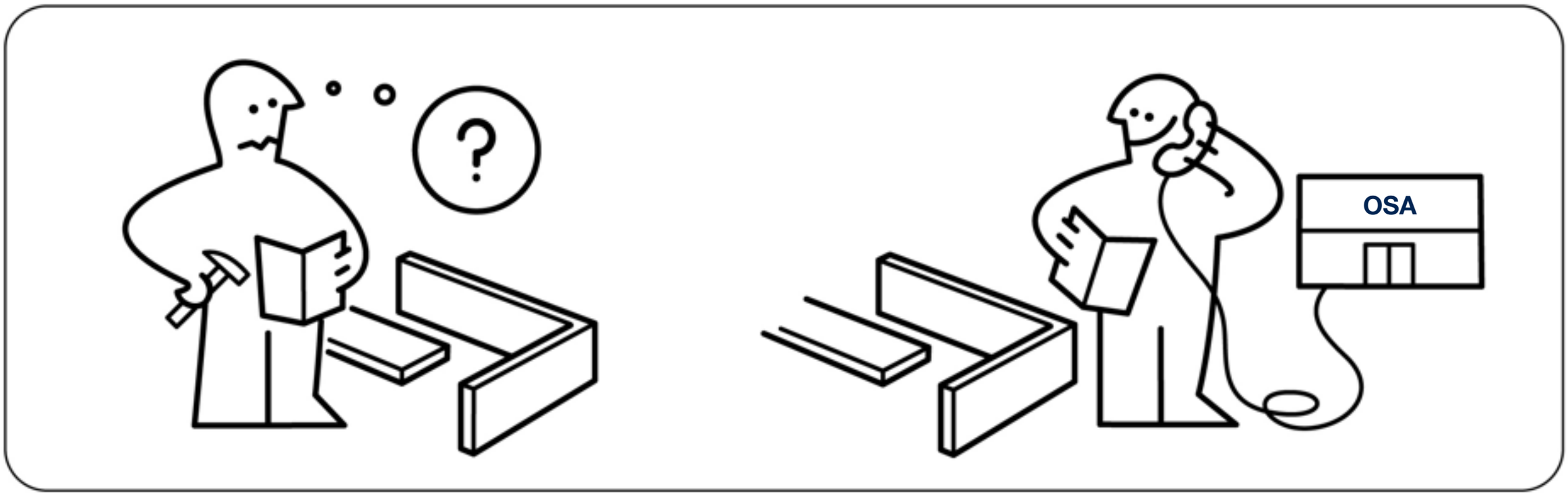
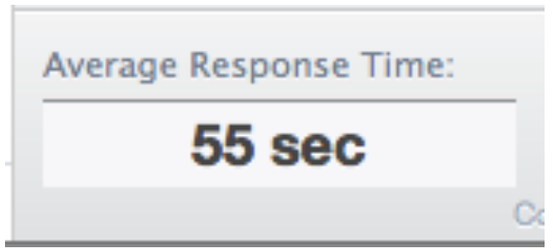
Instructions
et micro-instructions



Ordinateurs: Structure et Applications, Hiver 2015
Jean-François Lalonde

Aujourd'hui

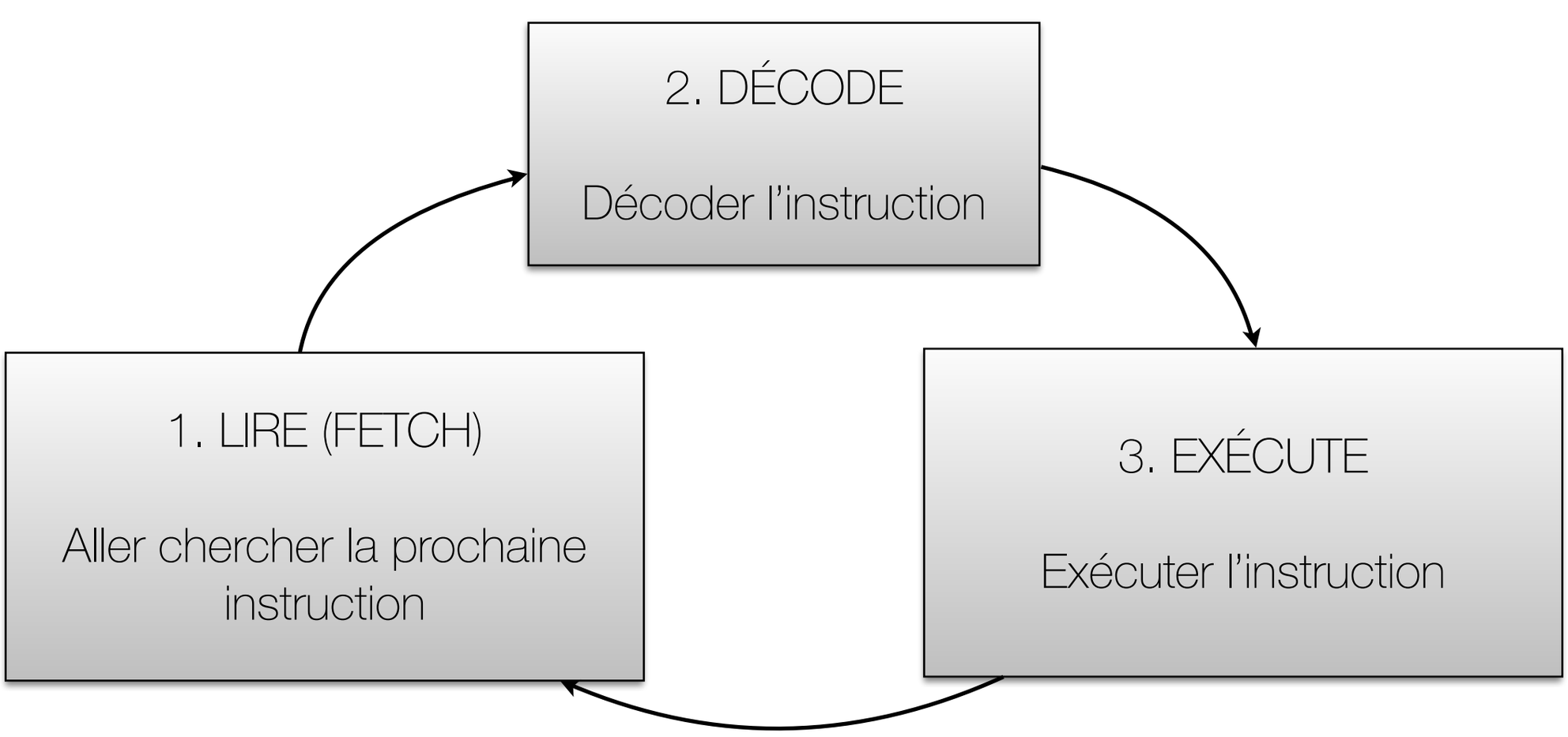
- Disponibilités
  - Lundi 15h30–16h30 (Marc-André, PLT-0103)
  - Mardi 9h30–12h00 (Samuel, PLT-0103)
  - Jeudi 13h30–14h30 (Jean-François, PLT-1138E)
  - Vendredi 15h30–16h30 (Jean-François, PLT-1138E)
    - En tout temps sur Piazza!
- Questions? TP1? Simulateur?
- Plan du cours:
  - Instructions
  - Micro-instructions



Cycle d'instructions

Que fait le microprocesseur?

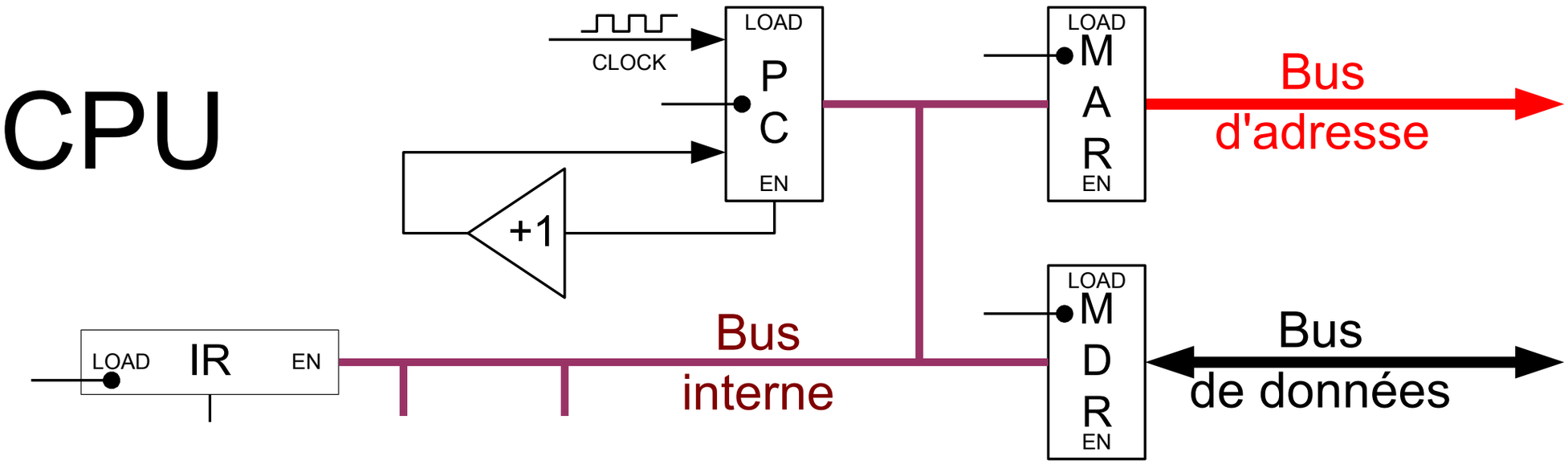
1. Lire: aller chercher la prochaine instruction
2. Décode: décode l'instruction (détermine ce qu'il y a à faire)
3. Exécute: exécuter l'instruction



“Fetch” en détails…

- Que se passe-t-il vraiment quand le micro-processeur veut lire la prochaine instruction?
  - On doit mettre l’instruction en question dans le registre IR (“Instruction Register”)
- Où est l’instruction?
  - En mémoire, à l’adresse contenue dans registre PC (“Program Counter”)

“Fetch” en détails…



“Fetch” en détails…

- PC -> MAR
  - On place le contenu du PC dans le MAR
    - PC contient l’adresse de la prochaine exécution à exécuter.
    - On active le bus de contrôle en lecture
    - Après cette opération, l’exécution comme telle est disponible… dans le MDR
- MDR -> IR
  - On place le contenu du MDR dans l’IR
    - Voilà! l’instruction est dans l’IR, prête à être décodée et ensuite exécutée

Types d’instructions

- Exemples:
  - Mouvements de données
  - Opérations arithmétiques et logiques
  - Rotations et décalages binaires
  - Contrôle de programme
  - Instructions pouvant traiter plusieurs données à la fois (“SIMD”)
- Nous en verrons d’autres dès la semaine prochaine!

Mouvement de données

- D'un registre à l'autre
  - ex. du TP1: MOV R1 R2
- De la mémoire vers un registre
  - ex. du TP1: LDR R1 [R2]
  - ou encore d'un emplacement mémoire directement: LDR R1 #0x22
- D'un registre vers la mémoire
  - ex. du TP1: STR R1 [R2]
  - ou encore d'un emplacement mémoire directement: STR R1 #0x22
- (Parfois—dépend de l'architecture) de la mémoire vers la mémoire directement, sans passer par un registre!
  - (pas dans le TP1): MOV #0x22 #0x32

Opérations arithmétiques

- Additions ou soustractions sur entiers
  - ex. du TP1: ADD R1 R2 ; R1 = R1 + R2
- Multiplication ou divisions
  - ex. ARM: MUL R0 R1 R2 ; R0 = R1 x R2
- Opérations sur entiers signés
  - ex. ARM: SMUL R0 R1 R2 ; R0 = R1 x R2
- Instructions pour valeurs à virgule flottante (IEEE 754)
  - ex. ARM: VADD R1 R0 ; R1 = R1 + R0

Rotations et décalages binaires

- Décalage
  - “tasser” tous les bits vers la droite ou la gauche
  - quelle est l’opération arithmétique correspondante?
  - que faire avec les nombres entiers en complément 2?
- Rotation
  - comme un décalage, sauf qu’on replace le bit à droite (ou à gauche) au lieu d’insérer un 0 (ou un 1).

Contrôle de programmes

- Sauter d'un emplacement mémoire à l'autre
  - ex: B #0xF1 ; #0xF1 -> PC
- Appeler une fonction
  - ex: CALL nomDeFonction
    - cela place l'adresse de nomDeFonction dans PC
    - plus de détails dans 2 semaines

Instructions SIMD

- SIMD: “Single Instruction, Multiple Data”
- Traiter plusieurs données en même temps, particulièrement utile pour des applications multimédias
  - ex: image = vecteur de pixels. On veut souvent appliquer la même opération (single instruction) à tous les pixels (multiple data)

Structure d'une instruction

- Instruction:
  - code d'opération ("opcode") en binaire
  - des paramètres: format et taille dépendent de l'opcode
- La taille et le format d'une instruction peuvent varier
- Par exemple (TP1):
  - instruction sur 16 bits
  - 4 premiers bits: opcode
    - combien d'opcodes peut-on définir au total?
  - 12 derniers bits: paramètres

Opcode	Argument 1	Argument 2
4 bits	4 bits	8 bits

Jeu d'instructions

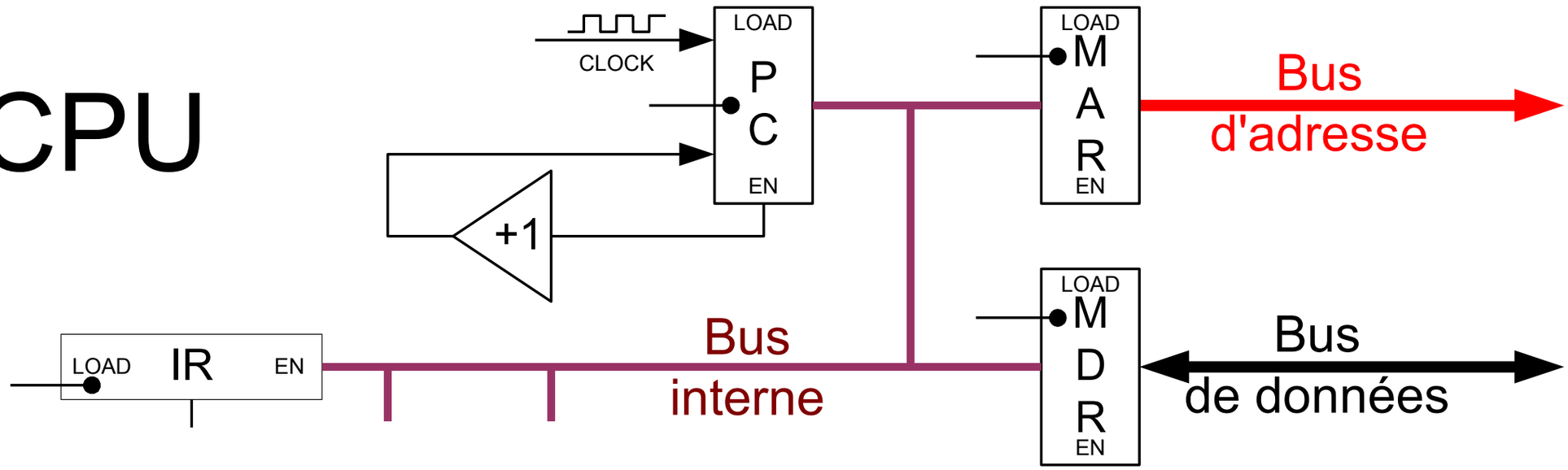
- La table ci-dessous est un exemple de jeu d'instructions.
- Chaque instruction possède un mnémonique en assembleur.
- Des microprocesseurs différents peuvent supporter le même jeu d'instruction ou plusieurs jeux d'instructions.

Bits 15 à 12 : Opcode de l'instruction
Bits 11 à 8 : Registre utilisé comme premier paramètre.
Bits 7 à 0 : Registre ou constante utilisés comme deuxième paramètre
Le microprocesseur possède quatre registres généraux nommés R0,R1,R2 et R3.

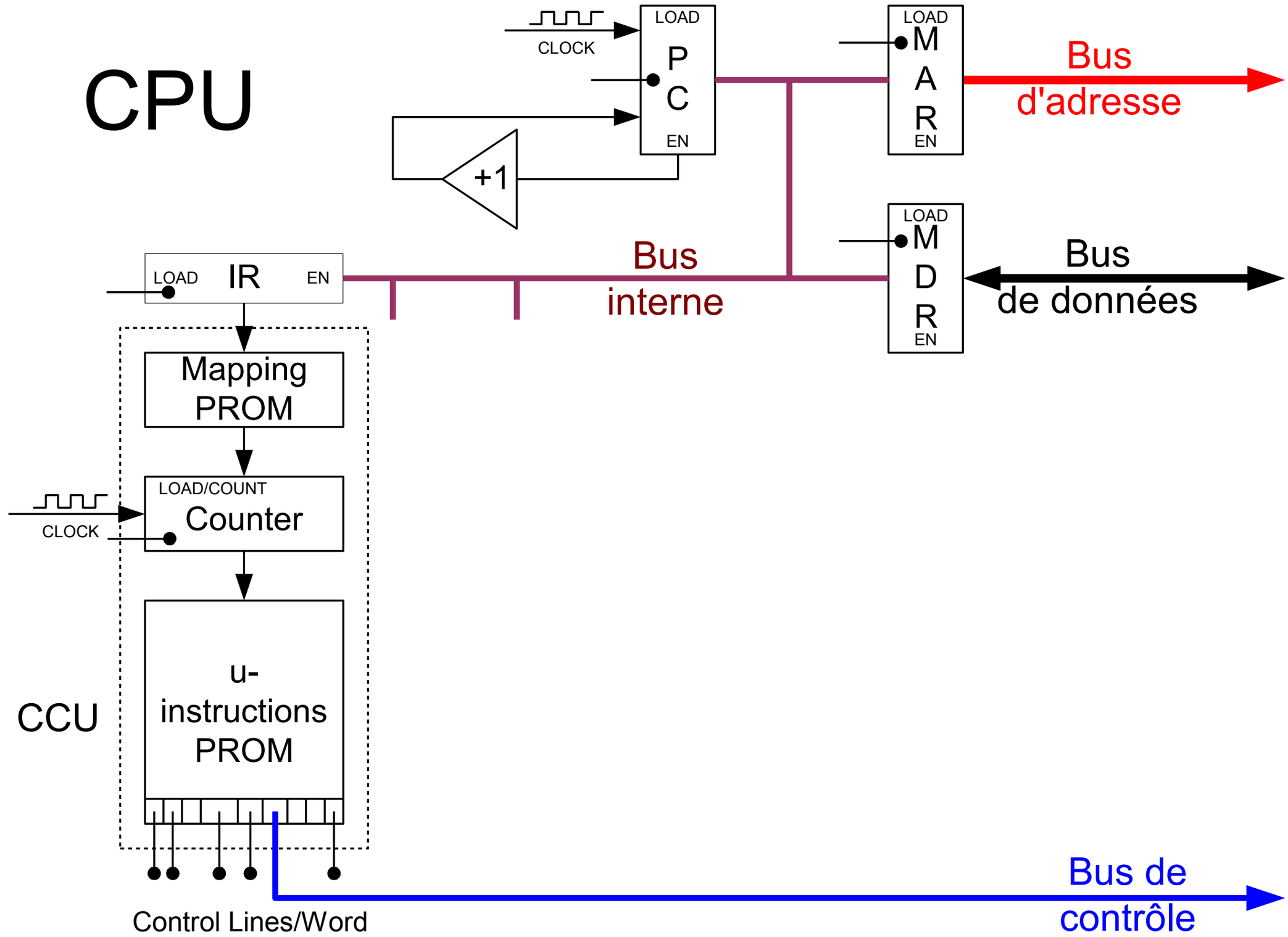
Le jeu d'instruction supporte les quatre instructions suivantes où Rd est le registre destination et Rs le registre source :

Mnémonique	**Opcode**	**Description**
MOV Rd Rs	0000	Écriture de la valeur du registre Rs dans le registre Rd
MOV Rd Const	0100	Écriture d'une constante dans le registre Rd
ADD Rd Rs	0001	Addition des valeurs des registres Rd et Rs et insertion du résultat dans le registre Rd
ADD Rd Const	0101	Addition de la valeur du registre Rd avec une constante et insertion du résultat dans Rd
SUB Rd Rs	0010	Soustraction de la valeur Rs à l'intérieur de registre Rd.
SUB Rd Const	0110	Soustraction d'une constante à l'intérieur du registre Rd
LDR Rd [Rs]	1000	Chargement d'une valeur se trouvant à l'adresse Rs de l'ordinateur dans un registre.
STR Rd [Rs]	1001	Écriture de la valeur d'un registre à l'adresse Rs de l'ordinateur.

“Execute” en détails…



“Execute” en détails…



ALU

- Le microprocesseur en exemple a un ALU très simple. Cet ALU a:
  - deux entrées (A et B) et une sortie
  - deux multiplexeurs (MUX_A et MUX_B) contrôlés par le CCU qui déterminent quelles seront les entrées.
- Le CCU détermine également quelle sera l'opération effectuée par l'ALU (+, -, &, |).
- Le résultat de l'opération se retrouve toujours dans l'accumulateur (ACC).
- À partir de là, il peut être propagé dans les registres de travail de l'ALU, sur le bus interne, ou à l'entrée (B) de l'ALU.
- Quelques détails supplémentaires sur l'ALU:
  - Si on suppose 6 registres de travail interne (R0, R1, R2, R3, R4 et R5), il faut 3 lignes de contrôle provenant du CCU pour MUX_A et 3 lignes pour MUX_B (MUX_A Select et MUX_B Select). En effet, chaque MUX a 8 entrées possibles et 2^3 = 8 (6 Regs + 0 + Acc ou Bus interne).
  - Si on suppose que l'ALU peut effectuer 16 opérations, il faut 4 lignes de contrôles pour déterminer ce que fera l'ALU (Function Select).
  - Les lignes de contrôles branchées sur les registres de travail et l'accumulateur détermineront où se retrouvera le résultat d'une opération de l'ALU.

MOV R0 #0x71

- (IR[#0x71] -> R0)
  - 0 + IR[#0x71] -> ACC
    - On place la portion de l'instruction qui correspond à #0x71 dans l'accumulateur (ACC)
  - ACC -> R0
    - On place le contenu de l'accumulateur dans R0. Voilà! L'instruction MOV R0 #0x71 est complétée.
- PC+1 -> PC
  - On incrémente PC pour la prochaine instruction.

Lisons la prochaine instruction (“fetch”)

- PC -> MAR
  - On place le contenu du PC dans le MAR
    - PC contient l’adresse de la prochaine exécution à exécuter.
    - On active le bus de contrôle en lecture
    - Après cette opération, l’exécution comme telle est disponible… dans le MDR
- MDR -> IR
  - On place le contenu du MDR dans l’IR
    - Voilà! l’instruction est dans l’IR, prête à être décodée et ensuite exécutée

LDR R1 [R0]

- (R0 -> MAR)
  - R0 + 0 -> ACC
    - On place la valeur de R0 dans l'accumulateur. Comme on passe par l'ALU, on additionne 0
  - ACC -> MAR
    - On place la valeur de l'accumulateur sur le bus d'adresse (MAR), et le bus de contrôle en lecture.
- (MDR -> R1)
  - 0 + MDR -> ACC
    - La valeur du bus de données est disponible sur le MDR, donc on la place dans l'accumulateur (encore une fois en l'additionnant avec 0)
  - ACC -> R1
    - On transfert la valeur de l'accumulateur vers le registre R1
- PC+1 -> PC
  - On incrémente PC

Lisons la prochaine instruction (“fetch”)

- PC -> MAR
  - On place le contenu du PC dans le MAR
    - PC contient l’adresse de la prochaine exécution à exécuter.
    - On active le bus de contrôle en lecture
    - Après cette opération, l’exécution comme telle est disponible… dans le MDR
- MDR -> IR
  - On place le contenu du MDR dans l’IR
    - Voilà! l’instruction est dans l’IR, prête à être décodée et ensuite exécutée

ADD R1 R3

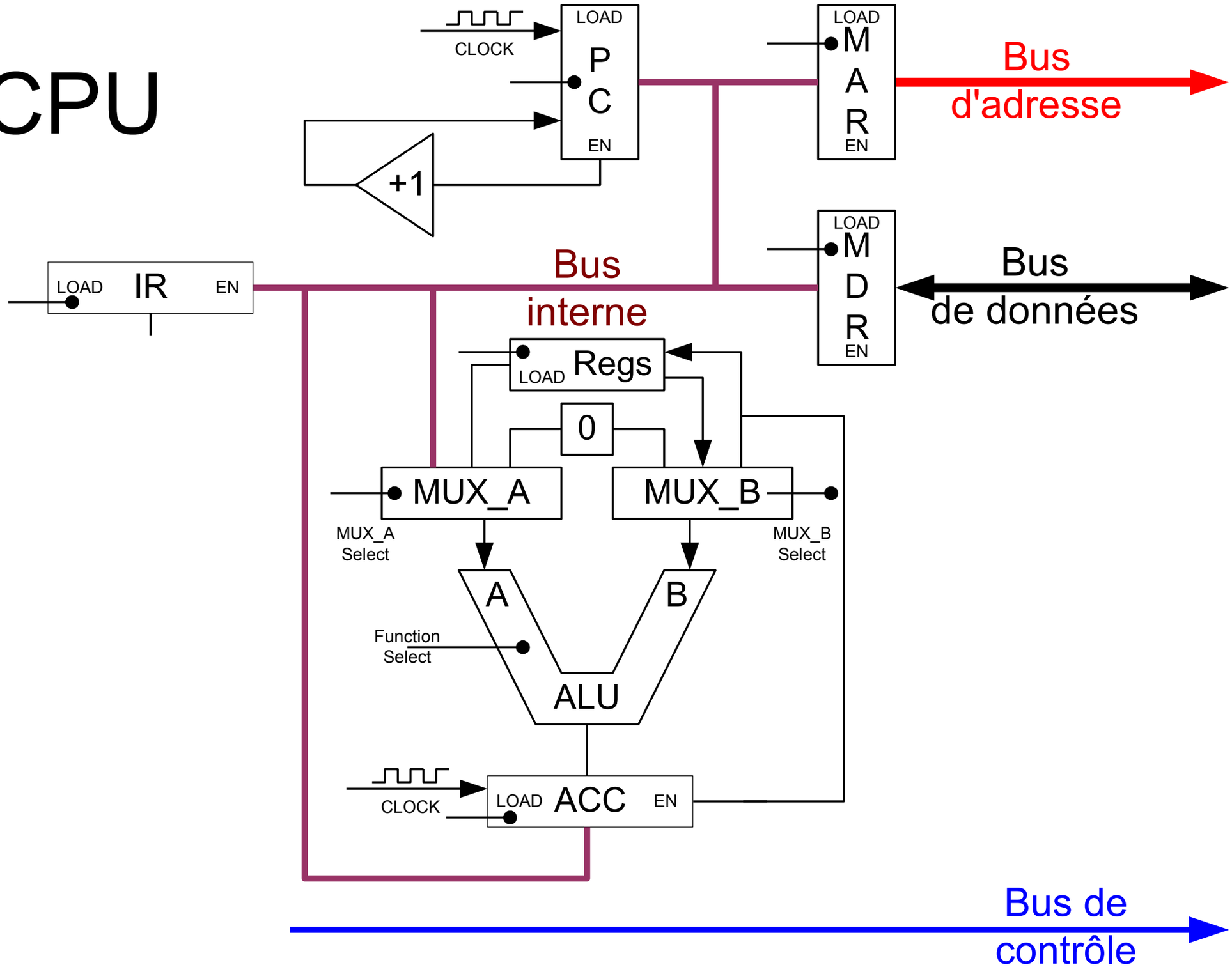
- (R1 + R3) -> R1
    - R1 + R3 -> ACC
        - On fait l'addition de R1 et R3, le résultat est placé dans l'accumulateur
    - ACC -> R1
        - On transfert le contenu de l'accumulateur dans le registre R1
- PC+1 -> PC
    - On incrémente PC

Décodage

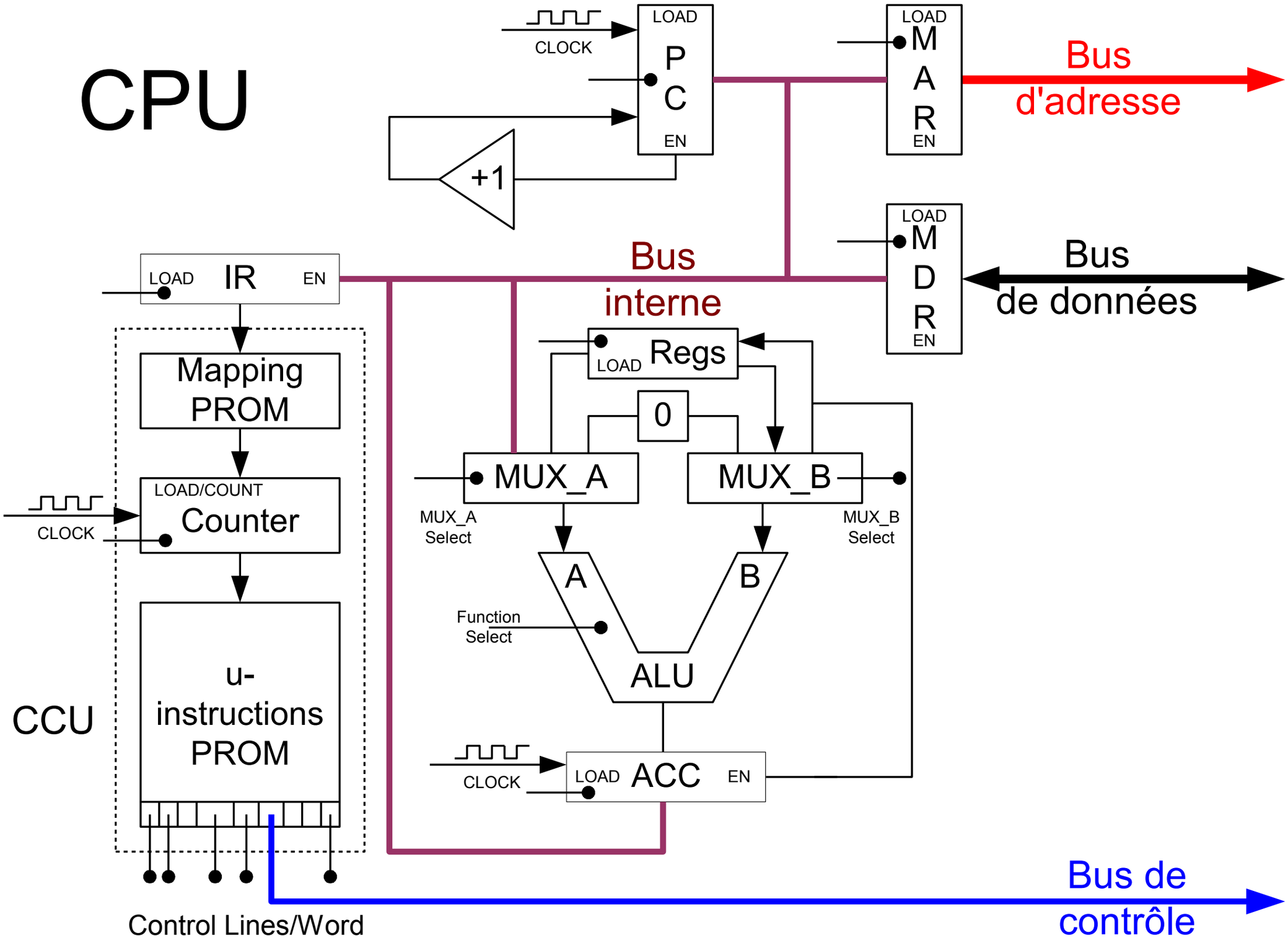
- Nous avons vu comment la prochaine instruction est lue (“fetch”) et exécutée (“execute”), mais comment est-elle décodée?
    - En d’autres mots, comment traduire instruction dans l’IR en une séquence de micro-instructions?

- 3 composantes:
    - “Mapping PROM”
    - “Counter” (compteur)
    - “micro-instruction PROM”

Décodage


CPU
CLOCK
LOAD
P
C
EN
+1
LOAD
M
A
R
EN
Bus
d'adresse
LOAD
IR
EN
Bus
interne
LOAD
M
D
R
EN
Bus
de données
LOAD
Regs
0
MUX_A
MUX_B
MUX_A
Select
MUX_B
Select
A
B
Function
Select
ALU
CLOCK
LOAD
ACC
EN
Bus de
contrôle

Décodage


CPU
CLOCK
LOAD
P
C
EN
+1
LOAD
M
A
R
EN
Bus
d'adresse
LOAD
M
D
R
EN
Bus
de données
Bus
interne
LOAD
IR
EN
Mapping
PROM
LOAD/COUNT
Counter
CLOCK
u-
instructions
PROM
CCU
LOAD
Regs
0
MUX_A
MUX_B
MUX_A
Select
MUX_B
Select
A
B
Function
Select
ALU
CLOCK
LOAD
ACC
EN
Bus de
contrôle
Control Lines/Word

Décodage

- “Mapping PROM”:
  - détermine l’emplacement de la première u-instruction à effectuer pour effectuer l’instruction.
    - Entrée: l’opcode de l’instruction
    - Sortie: Numéro de micro-instruction à effectuer.
- “Counter” (compteur):
  - détermine quelle sera la prochaine u-instruction à faire.
    - La valeur initiale de ce compteur provient de l’opcode de l’instruction via le mapping prom.
    - Par la suite, le compteur est incrémenté à chaque coup d’horloge, changeant la u-instruction en cours.

Décodage (suite)

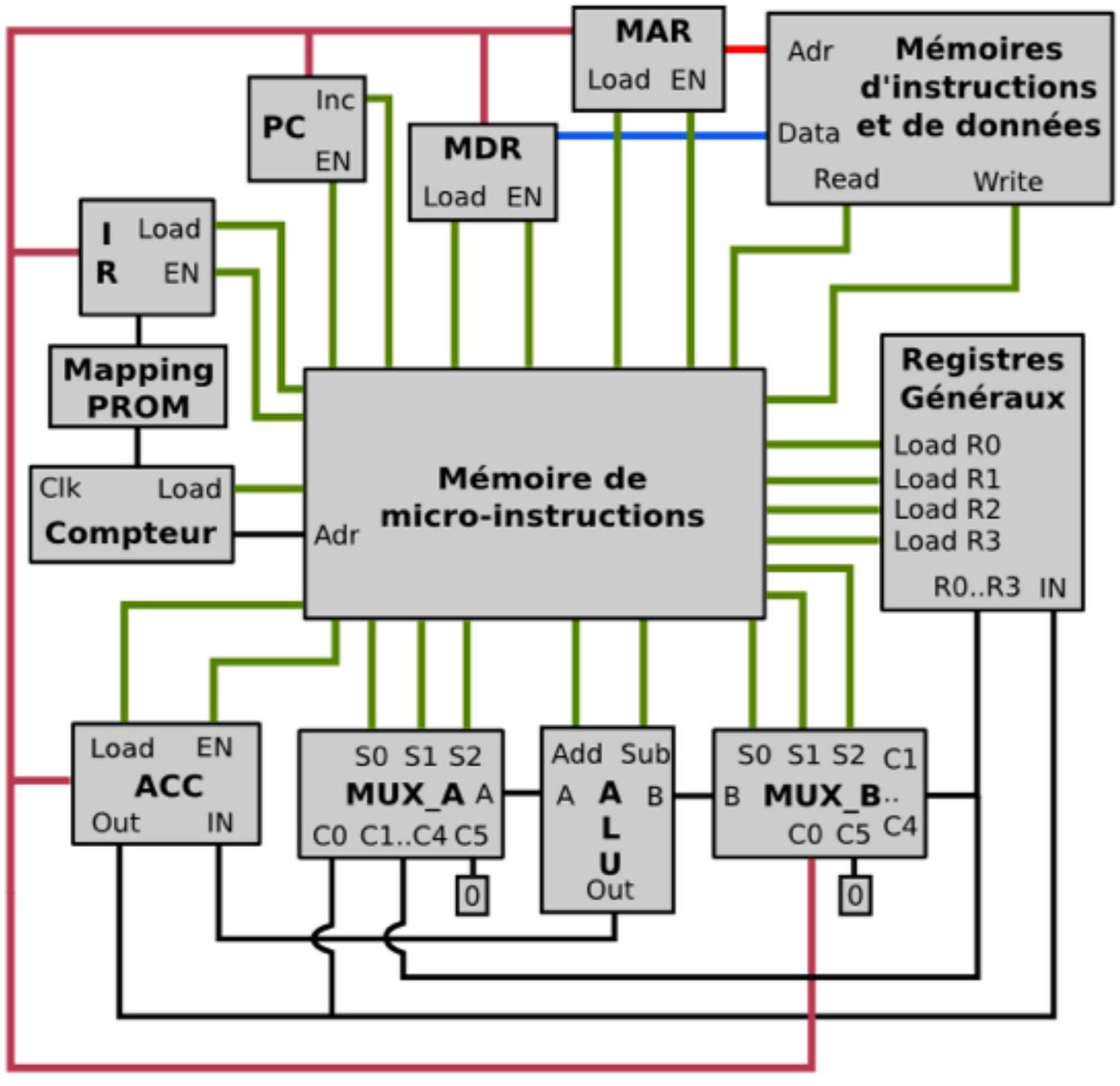
- “u-instruction PROM”:
    - mémoire contrôlant l’ensemble du microprocesseur.
        - Entrée: numéro de micro-instruction
        - Sortie: valeurs (prédéterminées) pour toutes les lignes de contrôle du microprocesseur
    - autrement dit, ce PROM détermine:
        - si les registres sont chargés ou lus
        - quelle sera l’opération effectuée par l’ALU;
        - quelle sera la valeur de certaines lignes du bus de contrôle. Les lignes de contrôles (ou mot de contrôle) sont représentées par des –o dans l’exemple. Elles partent toutes du CCU et elles se rendent au diverses composantes du microprocesseur.

Exercice

- Avec les instructions ci-bas, écrivez un programme qui additionne deux nombres:
  - Le premier est à l'adresse mémoire #0x71
  - Le deuxième est à l'adresse mémoire #0x72
  - Le résultat est sauvegardé en mémoire, à l'adresse #0x73

Mnémonique	**Opcode**	**Description**
MOV Rd Rs	0000	Écriture de la valeur du registre Rs dans le registre Rd
MOV Rd Const	0100	Écriture d'une constante dans le registre Rd
ADD Rd Rs	0001	Addition des valeurs des registres Rd et Rs et insertion du résultat dans le registre Rd
ADD Rd Const	0101	Addition de la valeur du registre Rd avec une constante et insertion du résultat dans Rd
SUB Rd Rs	0010	Soustraction de la valeur Rs à l'intérieur de registre Rd.
SUB Rd Const	0110	Soustraction d'une constante à l'intérieur du registre Rd
LDR Rd [Rs]	1000	Chargement d'une valeur se trouvant à l'adresse Rs de l'ordinateur dans un registre.
STR Rd [Rs]	1001	Écriture de la valeur d'un registre à l'adresse Rs de l'ordinateur.

Démonstration sur simulateur



Programmes, instructions, et micro-instructions

Programme
(additioner)

Instructions

```
MOV R0 #0x71
LDR R1 [R0]
MOV R0 #0x72
LDR R1 [R0]
ADD R1 R2
…
```

Micro-instructions

```
0 + IR[#0x71]
ACC -> R0
PC+1 -> PC
PC -> MAR
MDR -> IR
```

Micro-instructions

```
R0 + 0 -> ACC
ACC -> MAR
0 + MDR -> ACC
ACC -> R1
PC+1 -> PC
PC -> MAR
MDR -> IR
```

RISC & CISC

- Il existe plusieurs approches pour la conception d'un microprocesseur et de son jeu d'instructions. Ces approches influencent chaque aspect du design de l'architecture d'un microprocesseur. Les principales approches utilisées à ce jour sont CISC et RISC.

- CISC (Complex Instruction Set Computer)

    - jeu d'instructions complexe dont la longueur (des instructions) varie. Comme les instructions peuvent être longues et complexes, peu de registres sont requis.

    - Exemple: x86: (8086, Pentium)

- RISC (Reduced Instruction Set Computer)

    - jeu d'instructions simple dont la longueur est fixe (ex: 4 octets). Plusieurs registres sont requis pour exécuter des tâches complexes.

    - Exemple: PowerPC, ARM

RISC vs. CISC

Avantages RISC (**R**educed **I**nstruction **S**et **C**omputer)	**Avantages CISC** (**C**omplex **I**nstruction **S**et **C**omputer)
10 instructions sont utilisées 70% du temps	Plus de flexibilité au programmeur (par exemple, transferts mémoire-mémoire)
Avoir plusieurs registres permet d'éviter les accès mémoires (qui sont plus lents)	Programmes plus courts, plus petits
Opérations "fetch-decode-execute" sont simplifiées, car toutes les instructions ont la même taille	
Opérations simplifiées = architecture simplifiée = consommation réduite	

- Les microprocesseurs modernes sont des microprocesseurs RISCs ou hybrides (un microprocesseur supportant des instructions ayant deux longueurs seulement par exemple).
- Les microprocesseurs CISCs disponibles découpent habituellement les instructions complexes en instructions simples (comme du RISC) avant de les exécuter.