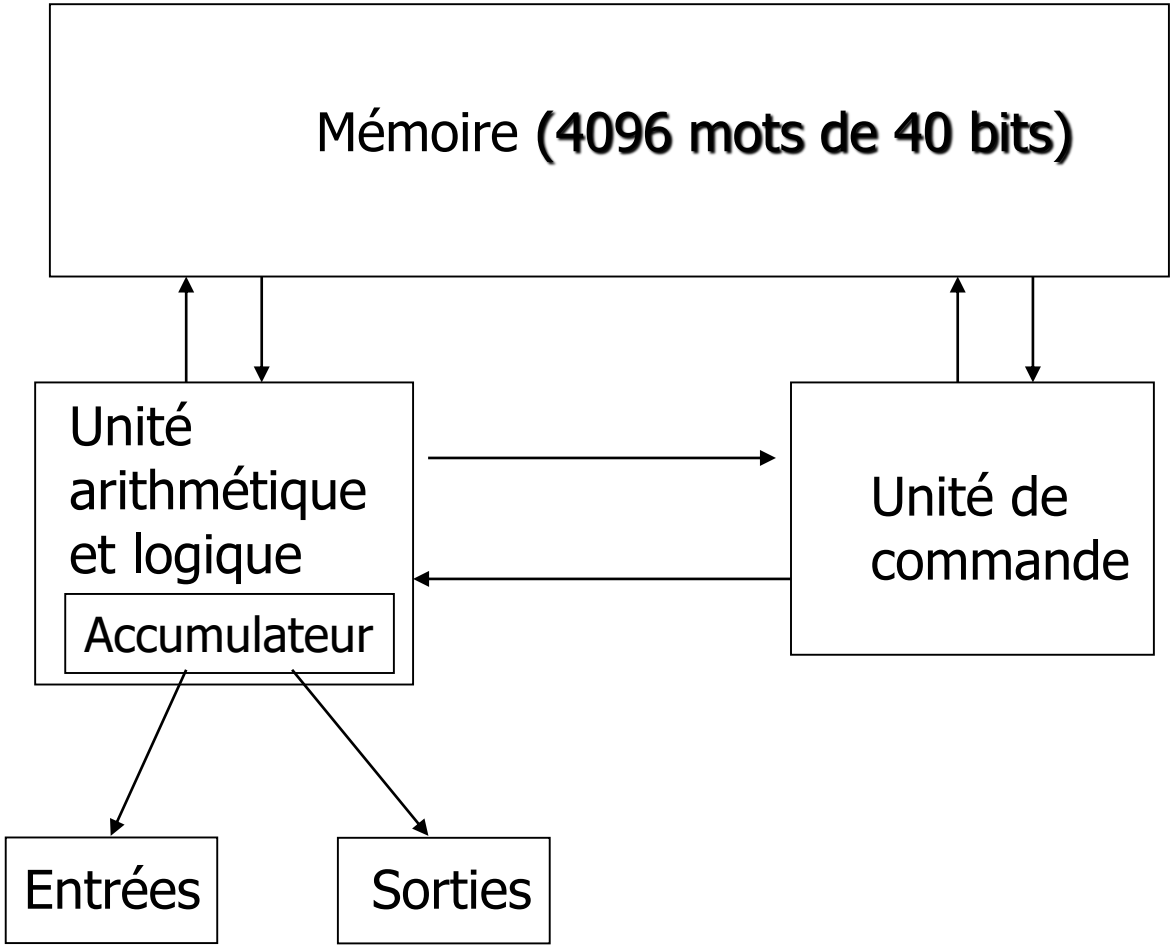
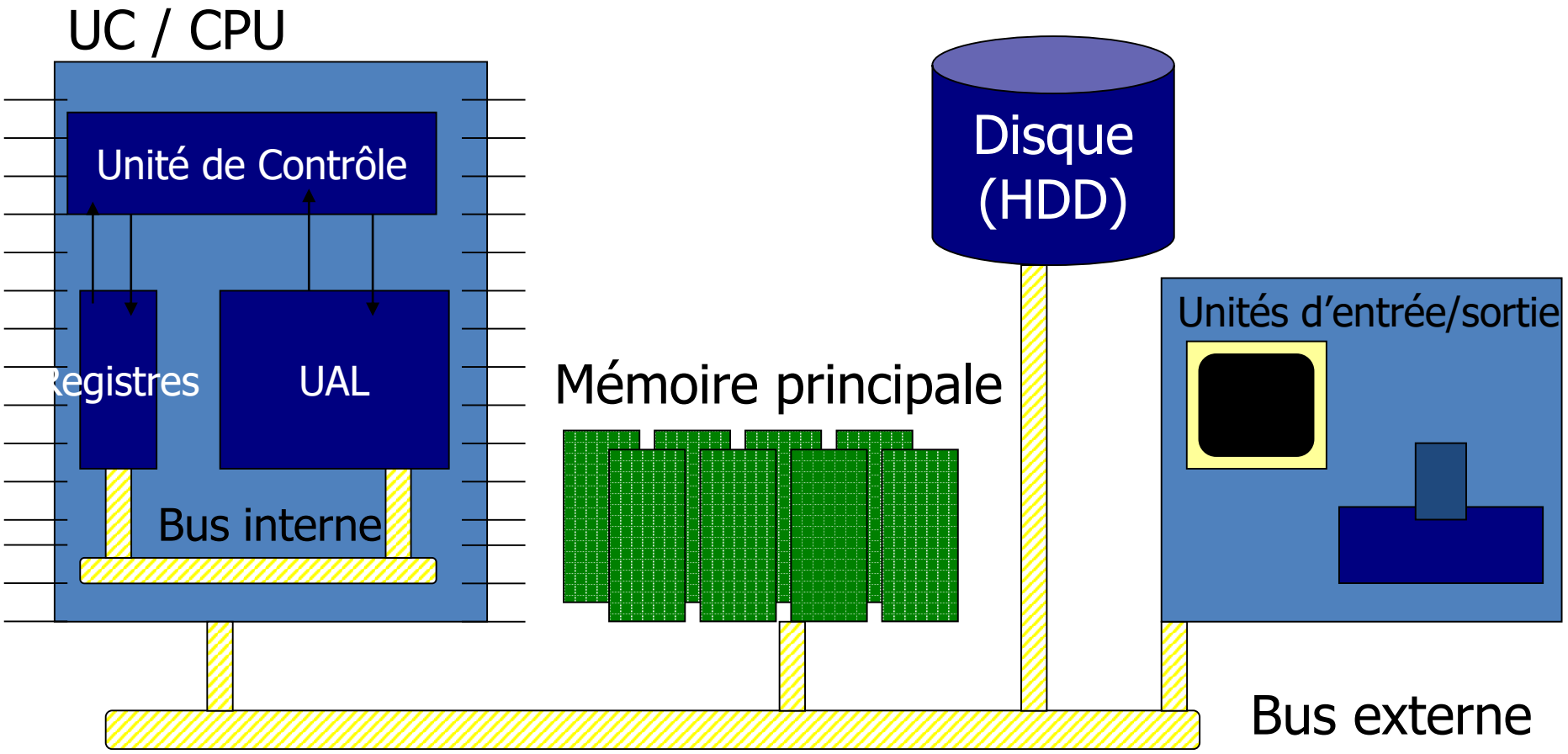
Le contrôleur et le cycle d'exécution machine

Architecture de Von Neumann (1952)



Architecture actuelle de l'ordinateur

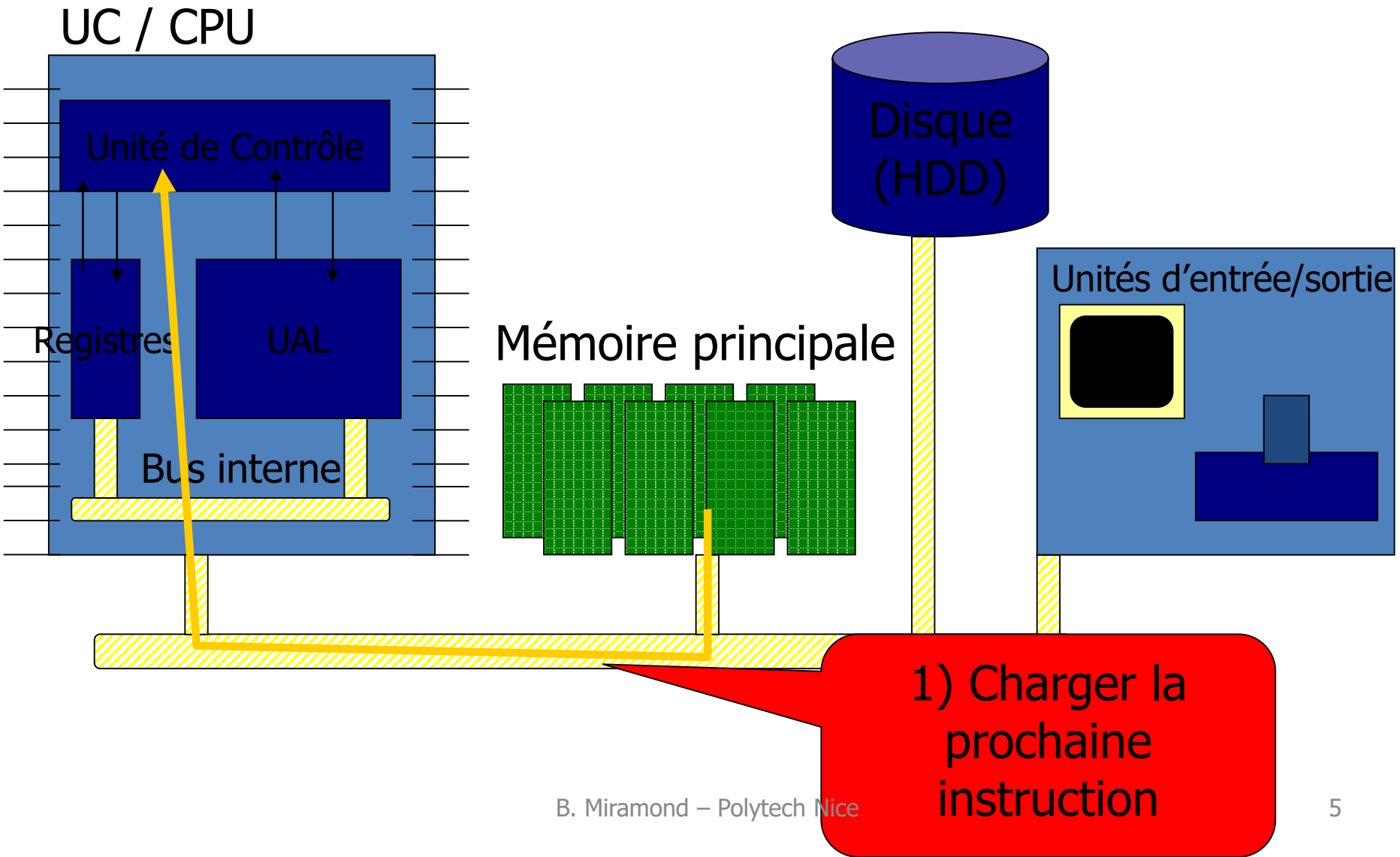


Bus externe
- Données
- Adresses
- Commandes

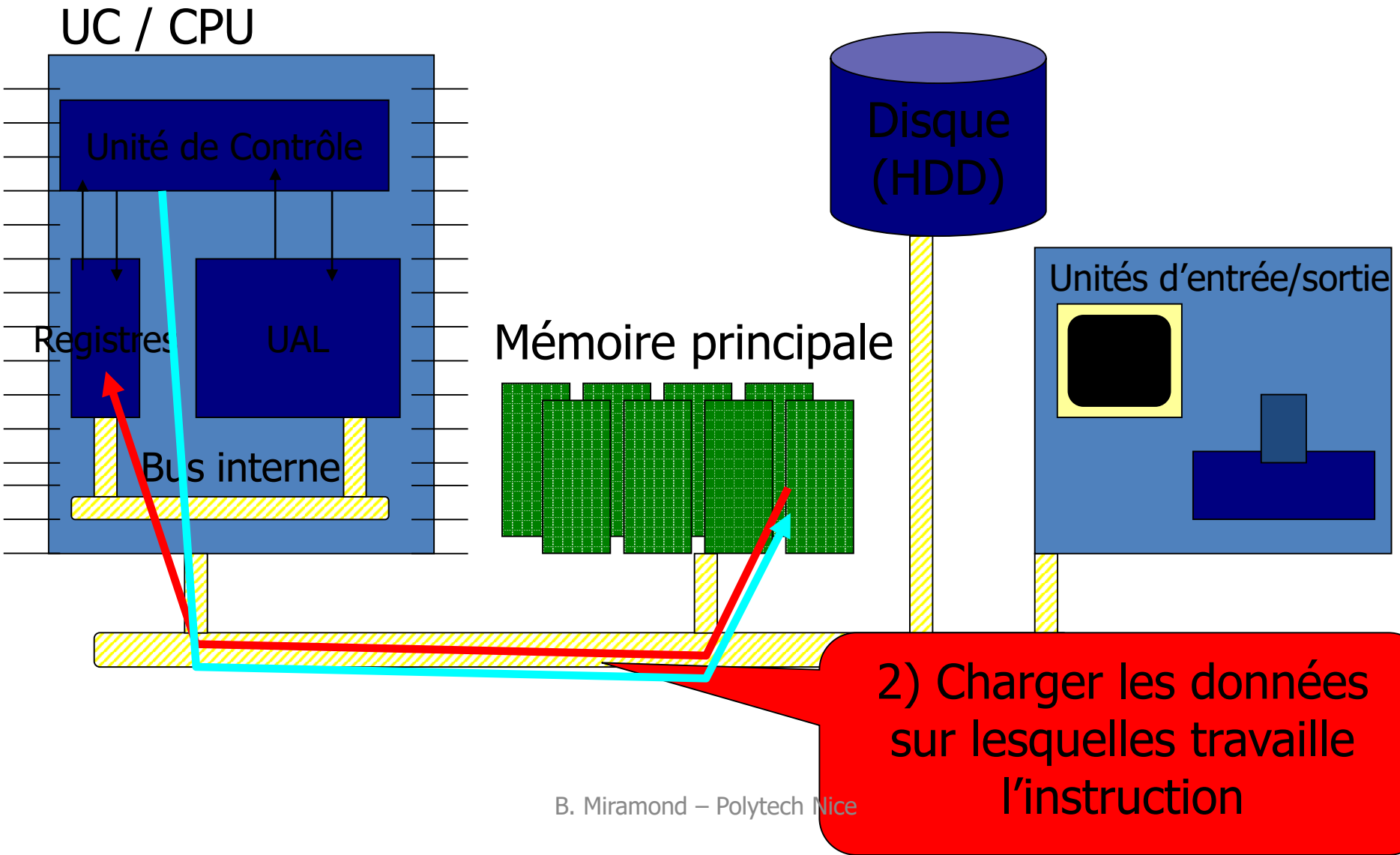
Echanges entre le processeur et la mémoire

- Le processeur exécute un programme
  - Programme écrit en mémoire
  - Transfert d'**instructions**
- Le programme manipule des variables
  - Transfert de **données**
- Toutes ces informations sont rangées à un certain emplacement
  - Transfert d'**adresses**

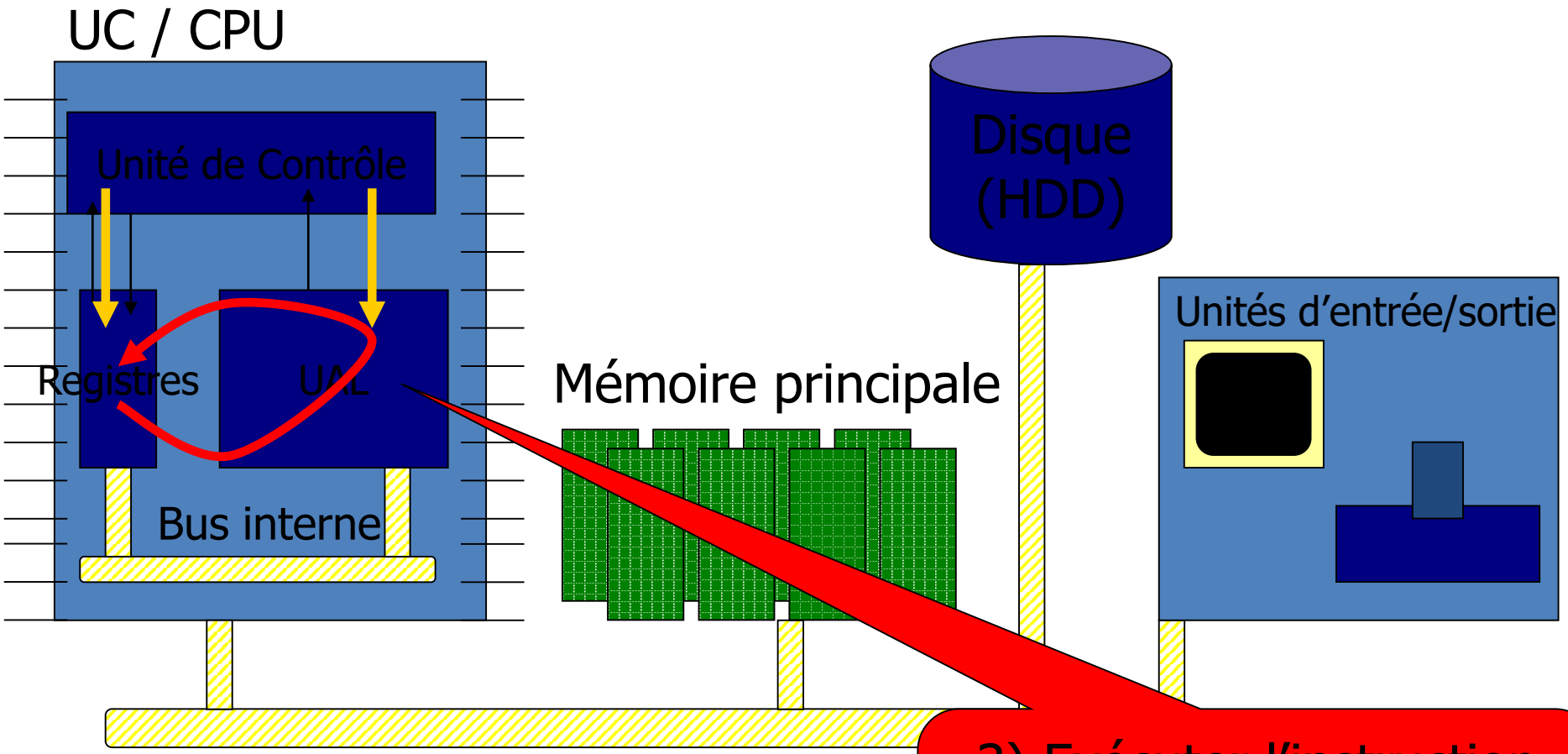
Principe général d'exécution



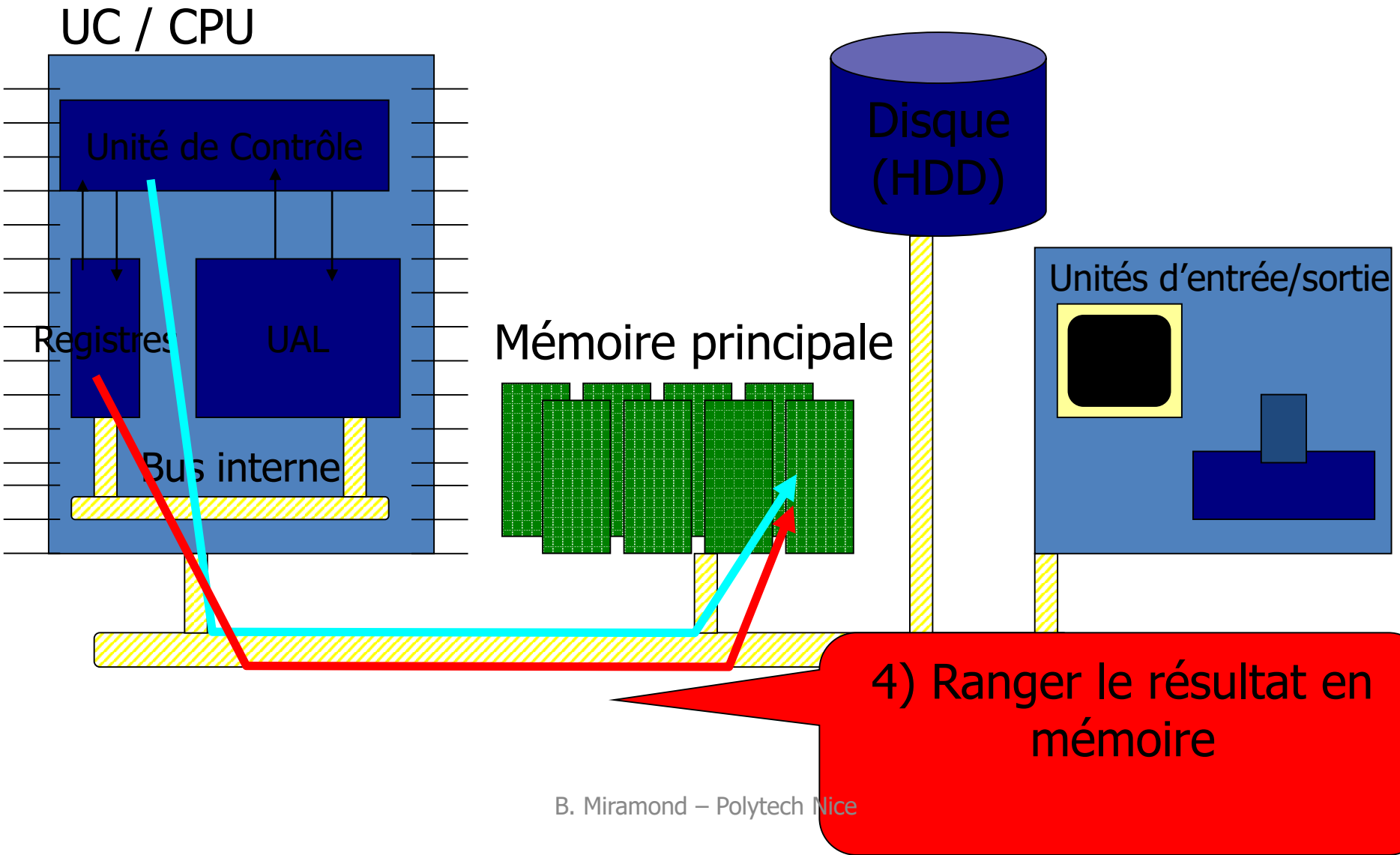
Principe général d'exécution



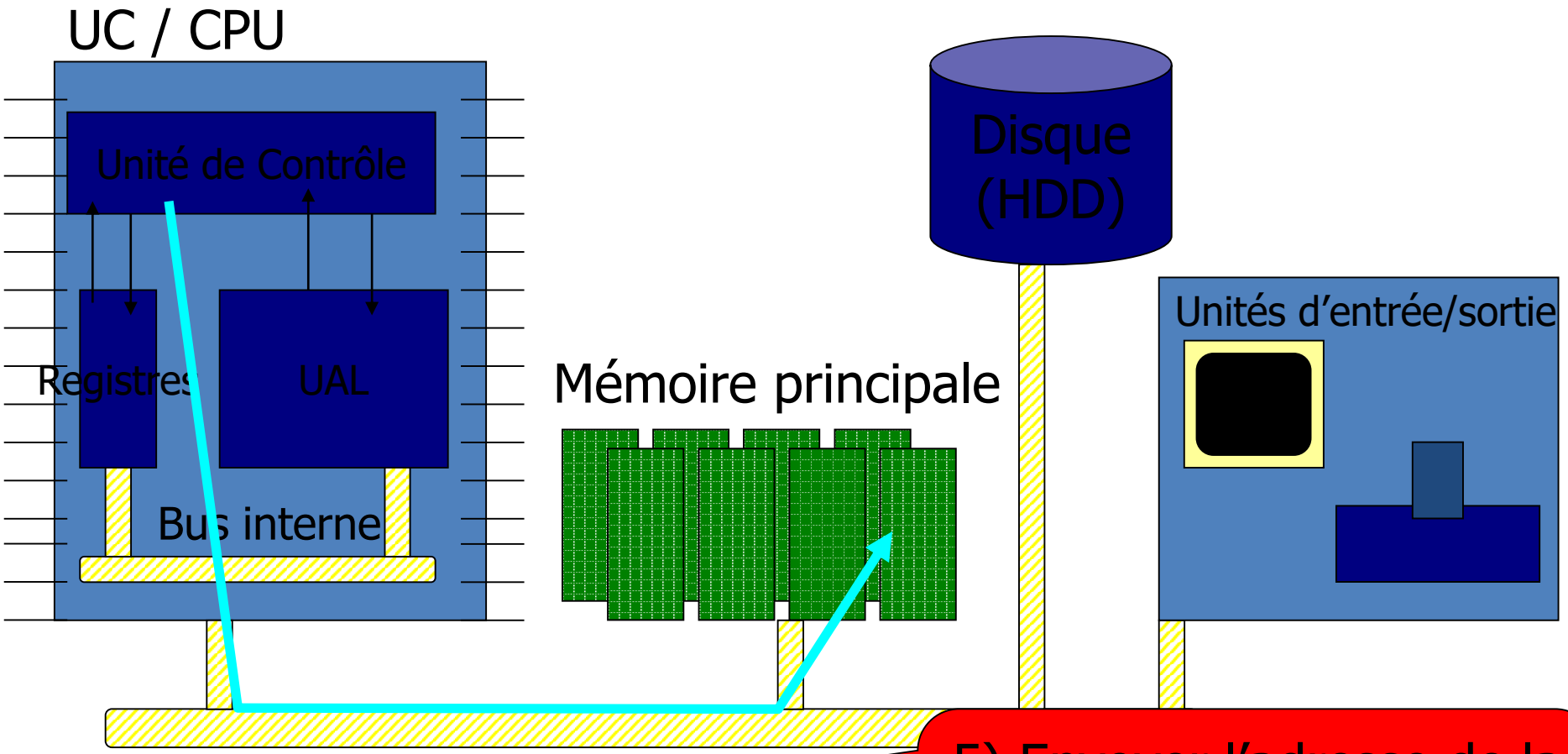
Principe général d'exécution



Principe général d'exécution



Principe général d'exécution



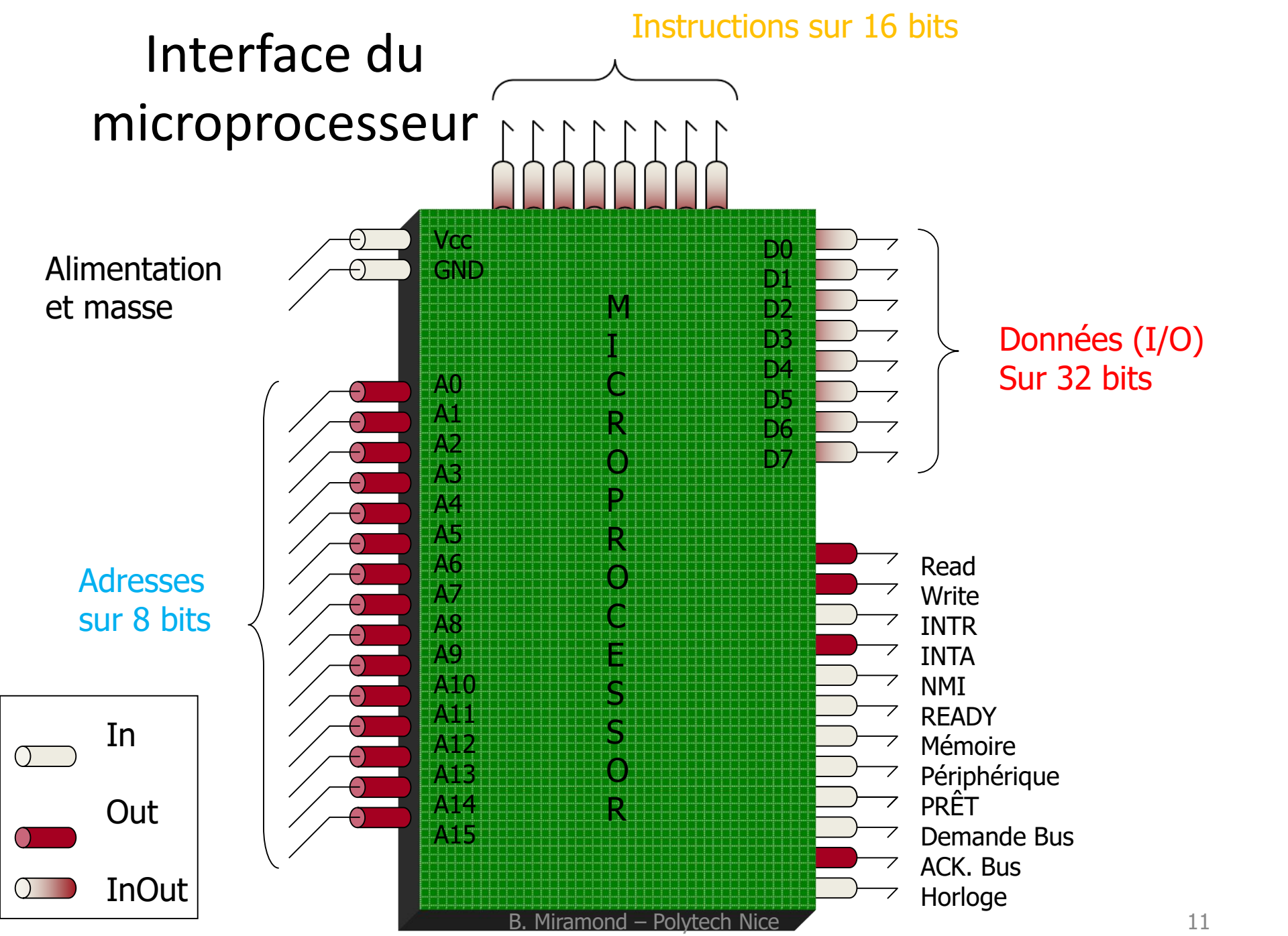
5) Envoyer l'adresse de la prochaine instruction
Revenir à l'étape 1)

Première vision du cycle d'exécution machine

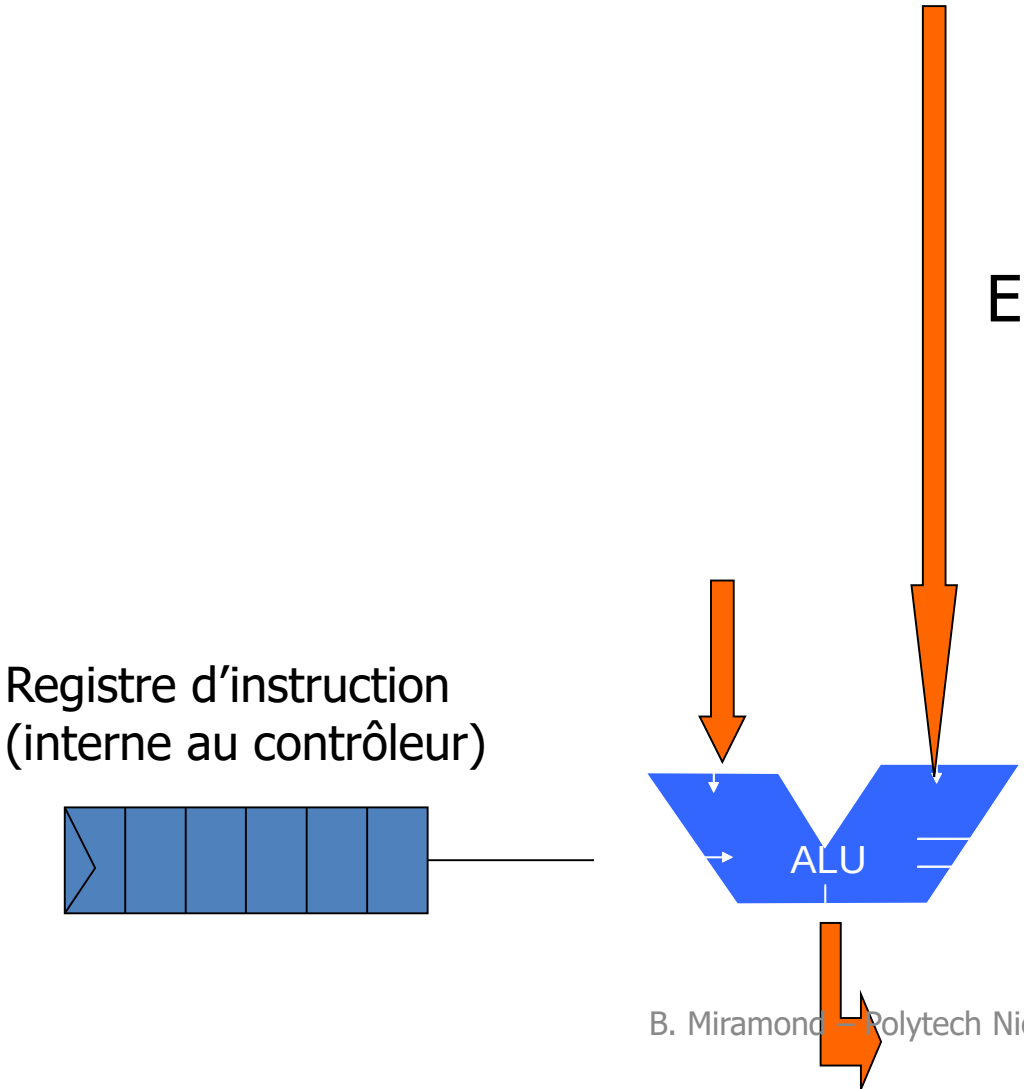
Un cycle d'exécution machine consiste à

1. Charger l'instruction
2. Charger ses données
3. Faire un traitement sur ces données
4. Ranger le résultat du traitement
5. Désigner la prochaine instruction

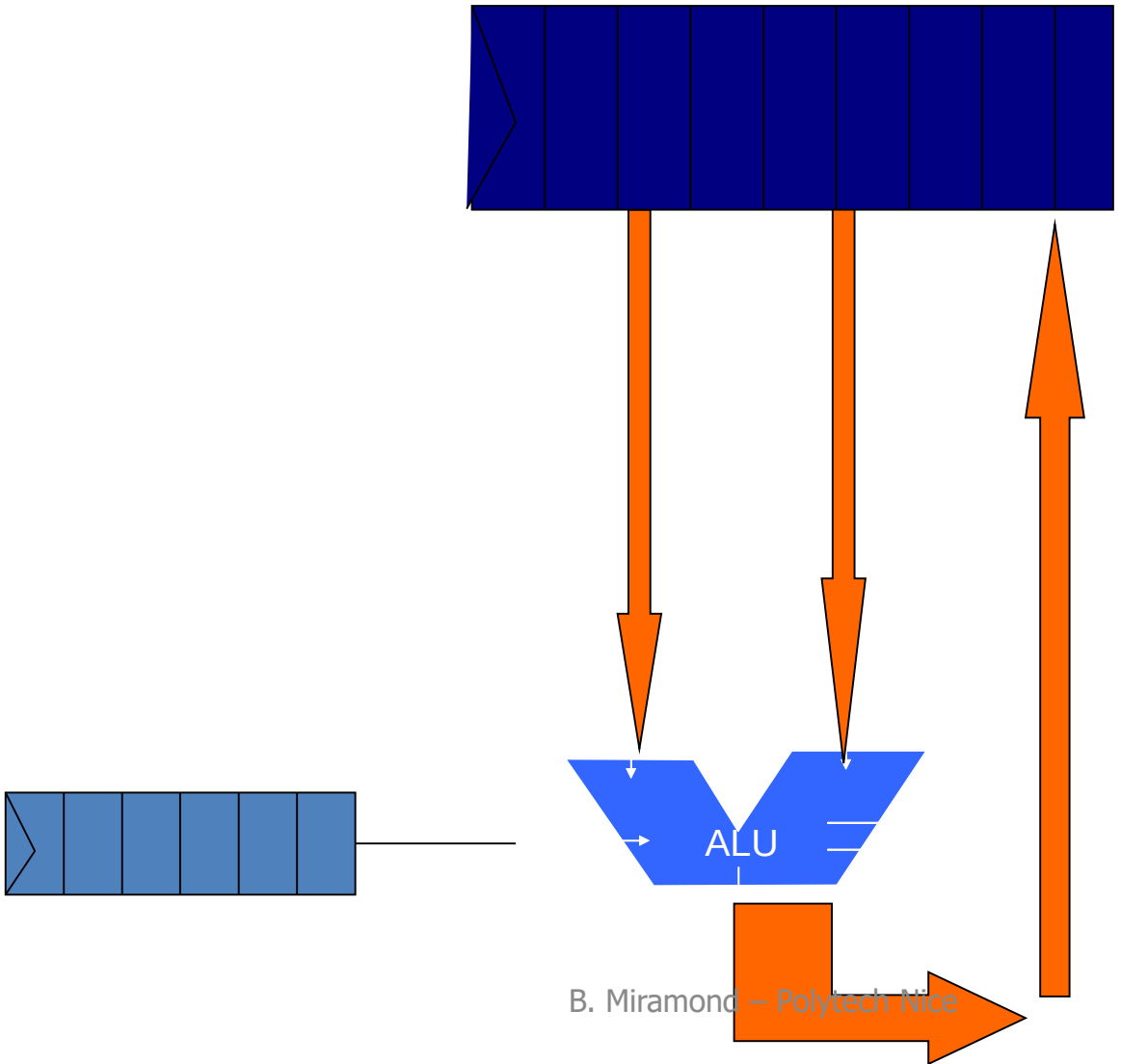
Interface du microprocesseur



Registre de contrôle de l'ALU



Architecture à chargement / rangement



Banc de registres

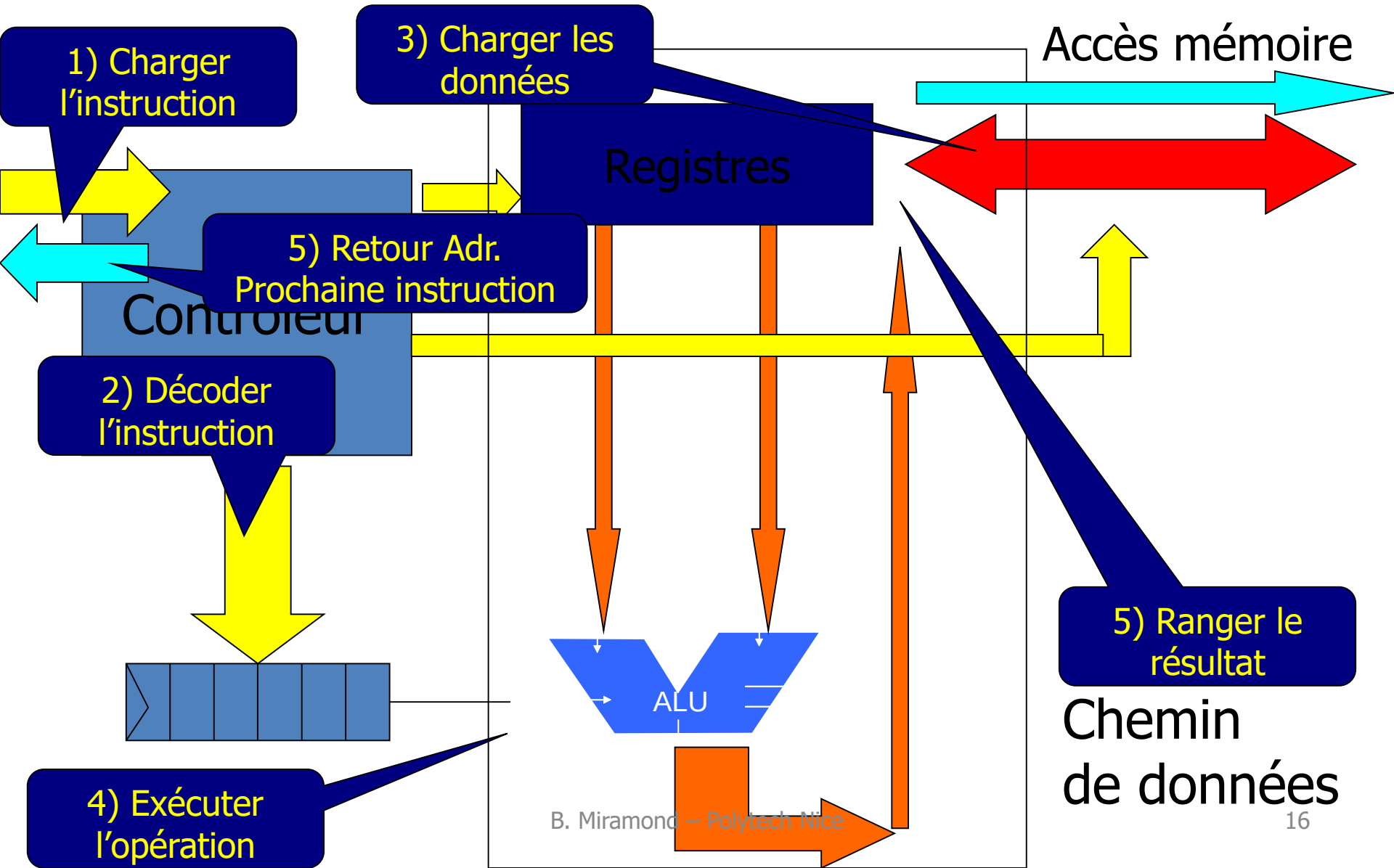
Contrôle du chemin de données

Rôle du contrôleur

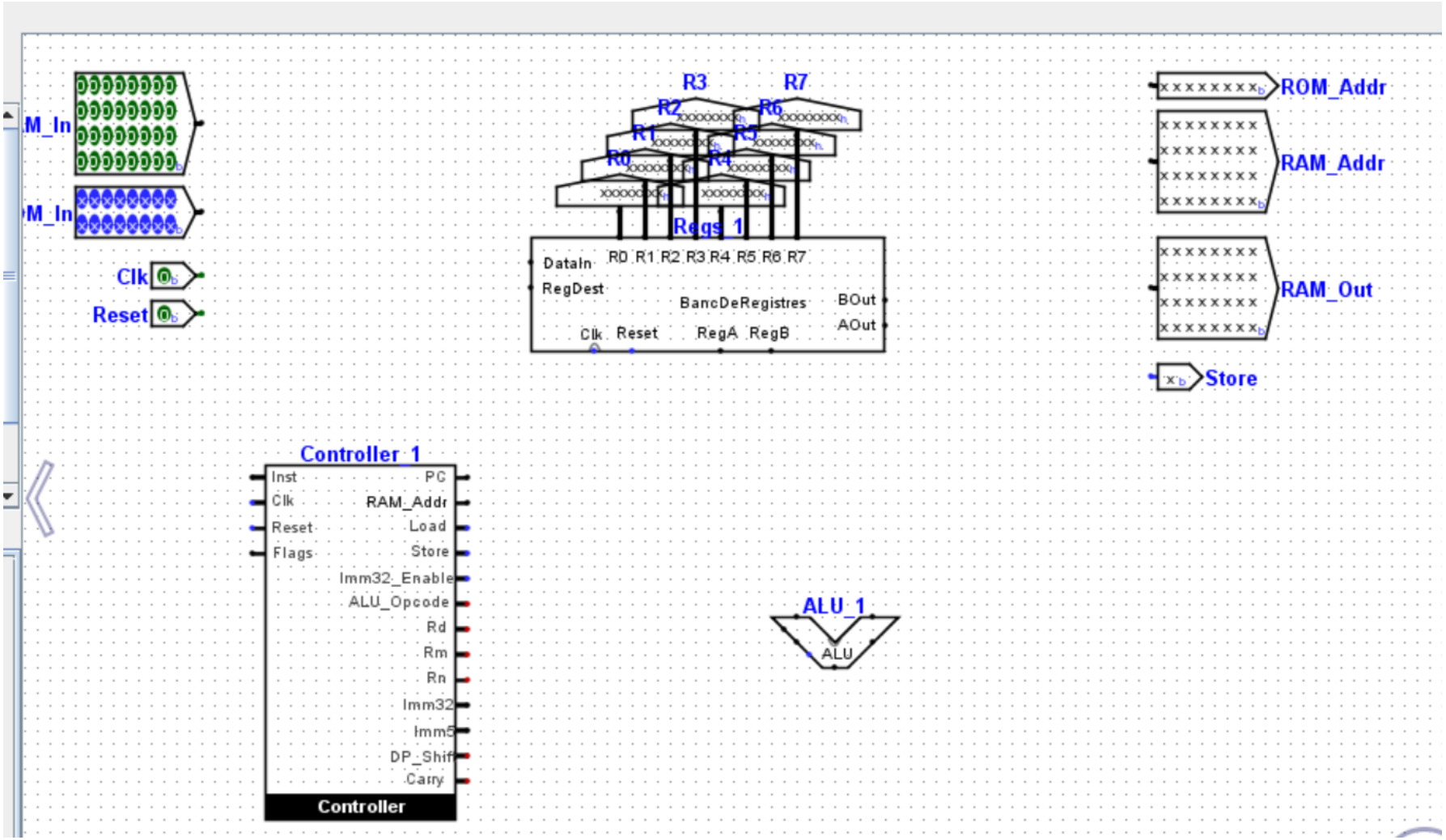
- Commander les opérations de l'ALU
- Placer les adresses mémoire de lecture de données (variables). Requête en lecture
- Désigner parmi les registres ceux qui alimenteront les 2 entrées de l'ALU *(RegA et RegB)*
- Désigner dans quel registre le résultat de l'ALU doit être rangé (RegDest)
- Placer l'adresse mémoire à laquelle doit être mémoriser un résultat (variable). Requête en écriture
- Charger la prochaine instruction

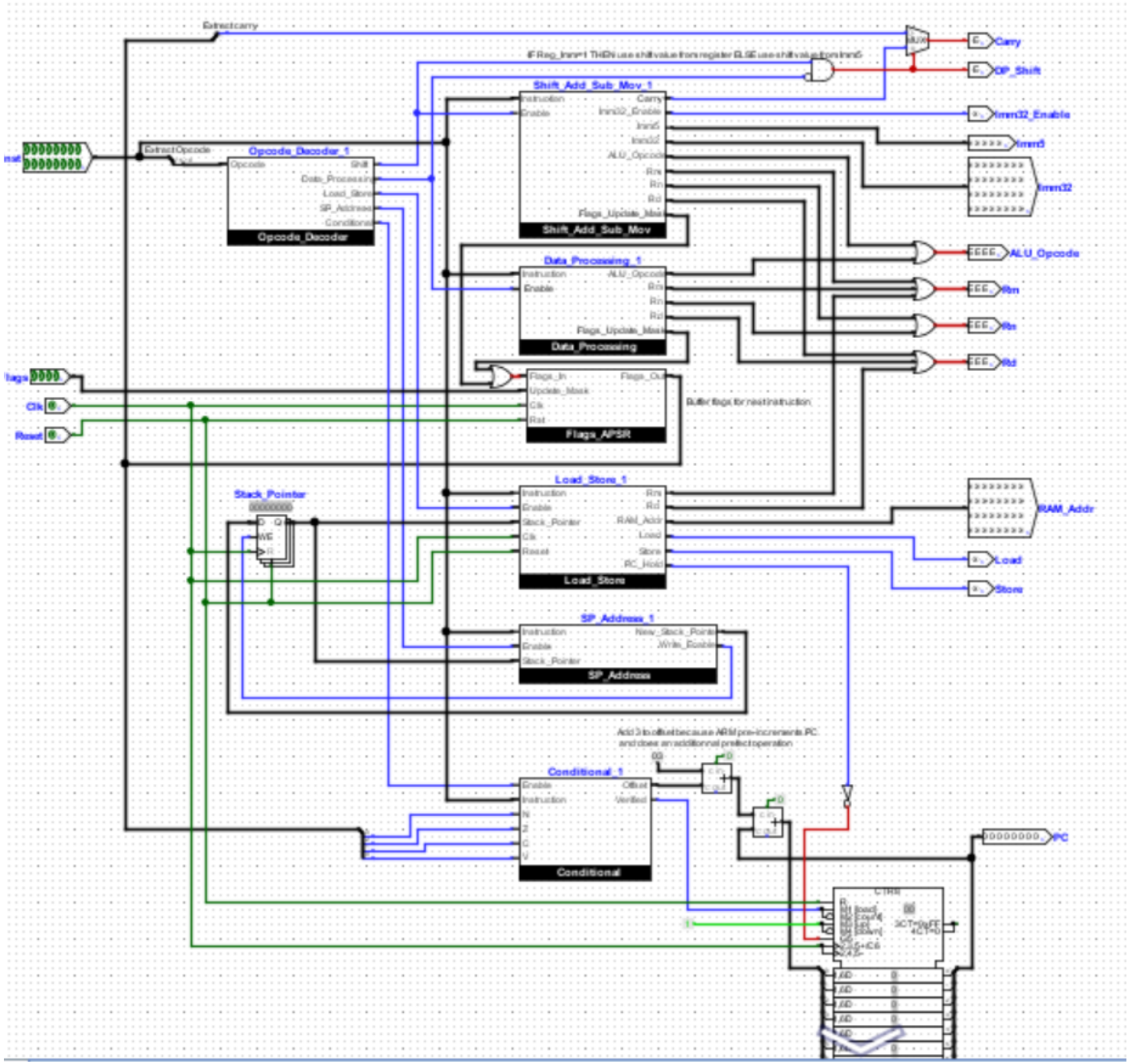
Le contrôleur dépend de l'architecture

Plus précisément



Le contrôleur dans Logisim




Extract carry
Opcode_Decoder_1
Opcode_Decoder
Shift_Add_Sub_Mov_1
Shift_Add_Sub_Mov
Data_Processing_1
Data_Processing
Flags_APSR
Load_Store_1
Load_Store
Stack_Pointer
SP_Address_1
SP_Address
Conditional_1
Conditional
Carry
OP_Shift
Imm32_Enable
Imm5
Imm32
ALU_Opcode
Rm
Rn
Rd
RAM_Addr
Load
Store
PC
Clk
Reset

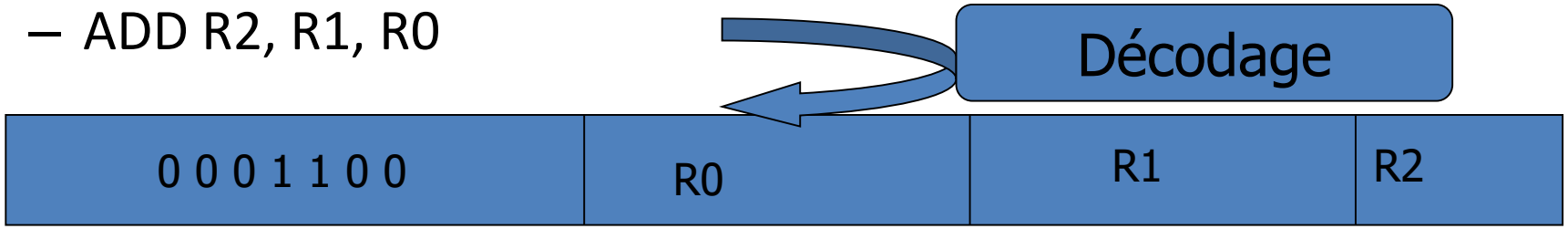
Commandes de l'architecture

- Tous les signaux de commande de l'architecture sont mémorisées dans un seul registre appelé le **Registre d'Instruction**
- Il est composé de plusieurs champs contrôlant chacun une partie de l'architecture
  - L'ALU
  - Les opérandes de l'ALU
  - La/les sorties de l'ALU
  - Les accès mémoires (Lecture, écriture, fetch)
  - Des données complémentaires (immédiat, adresse de saut…)

Cycle d'exécution des instructions

- Un **cycle d'exécution** du processeur correspond à l'exécution d'une instruction :

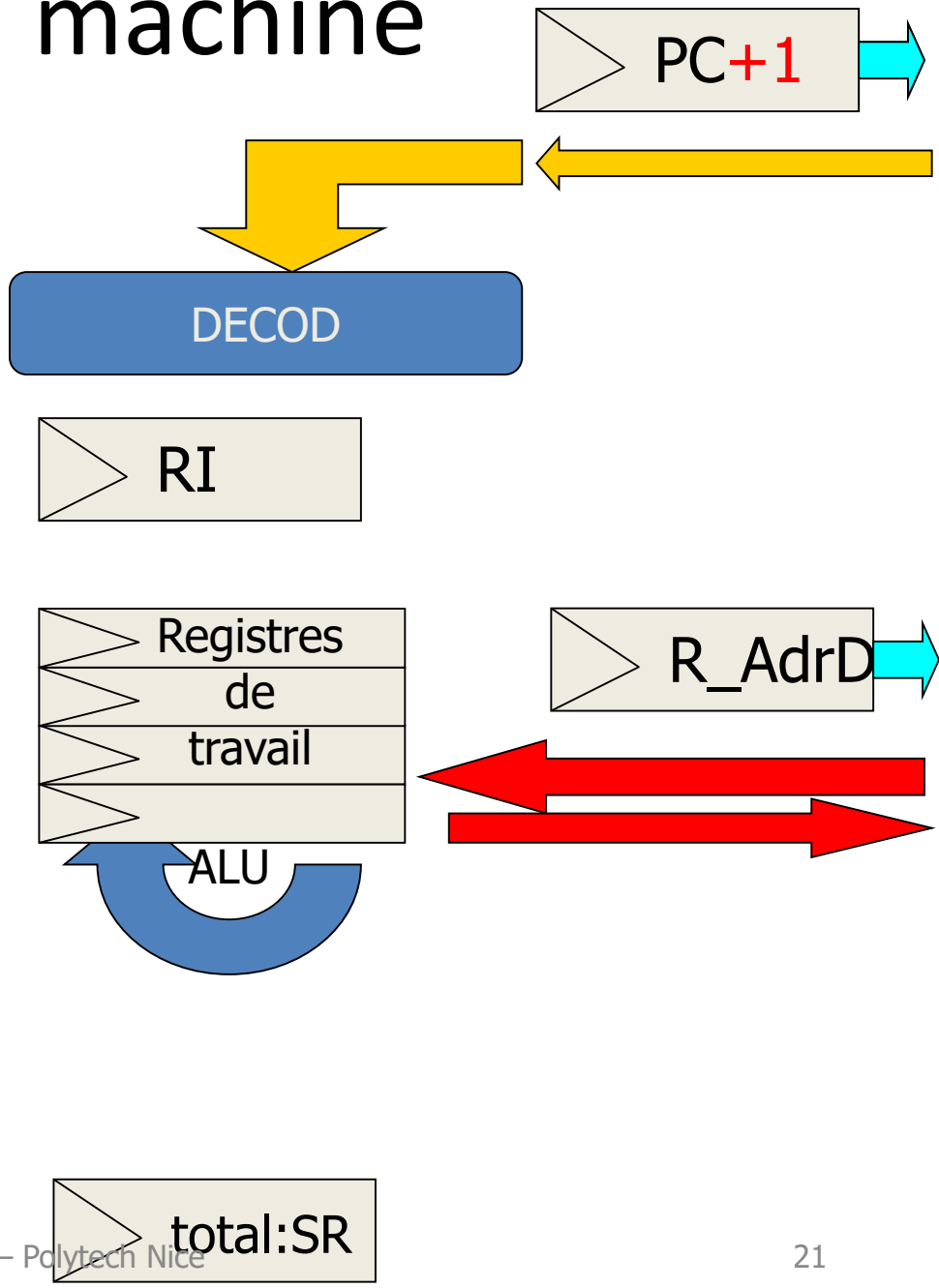
- Ces **instructions** sont des suites de bits (commandes) que l'on peut coder par des *mnémoniques* en assembleur :
  - ADD R2, R1, R0



  - LDR R3, [12] ou LDR R3, [SP, 12]

Cycle d'exécution machine

1. Charger l'instruction
2. Incrémenter PC
3. Décoder l'instruction
4. Charger les données
5. Exécuter l'opération
6. Ranger le résultat
7. Retour



Résumé

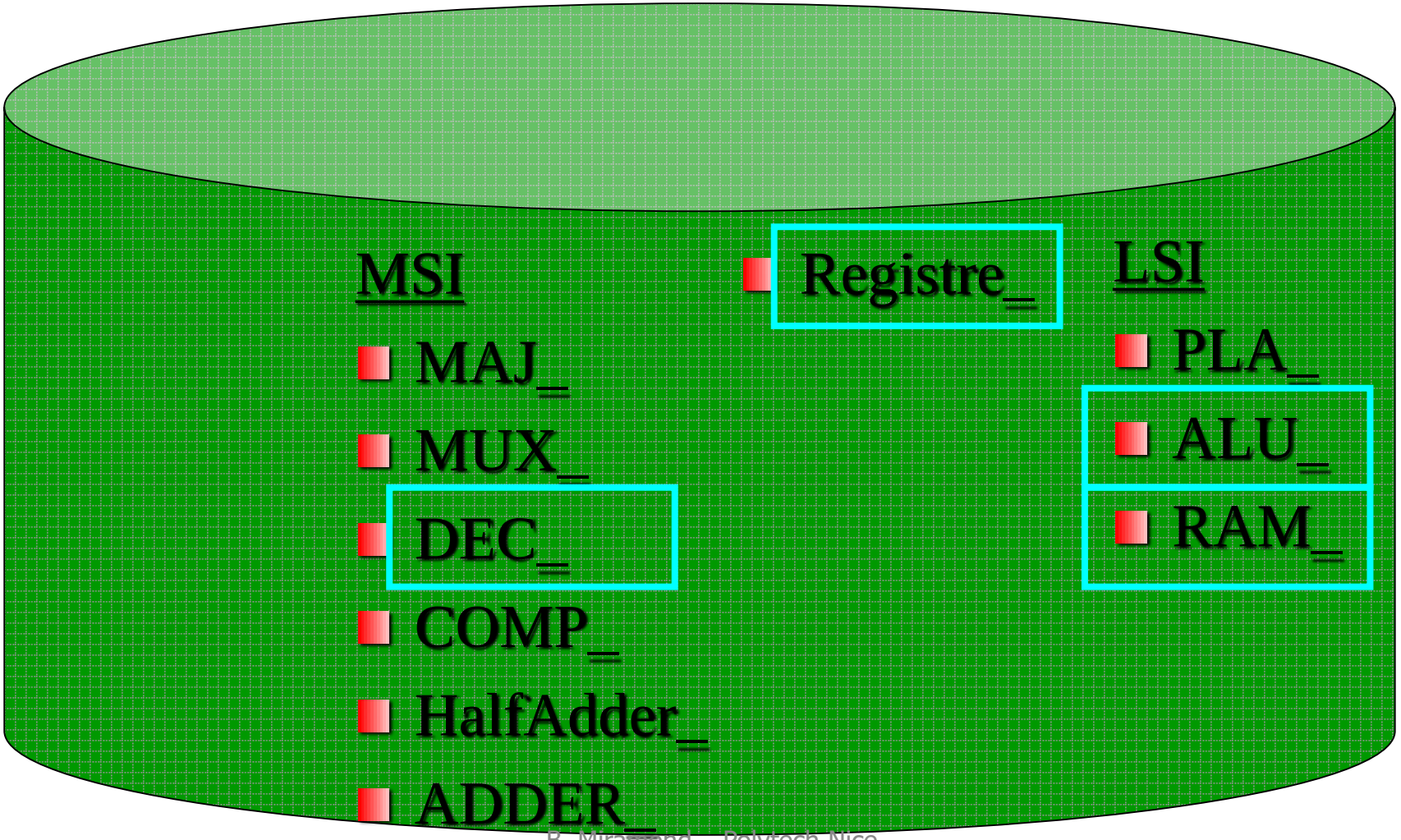
Architecture composée de 2 parties

- Le **chemin de donnée** réalise les traitements
  1. **Unité de transfert mémoire**
     - Registre d'adresse de données
     - Registre d'adresse d'instructions ou Compteur Ordinal (CO) ou PC
  2. **Registres de travail** (architecture à Accumulateur, à Pile, à banc de registres)
  3. **L'ALU**
- Le **contrôleur**
  4. **Séquenceur**
     - Registre d'état APSR
  5. **Décodeur**
     - Registre d'instruction

Posez vous quelques questions

- A quoi servent chacun des registres du contrôleur ?
- Pourquoi y a-t-il 7 sous-composants dans le contrôleur ?
- Le codage des instructions est-il toujours le même ?
- Quelle est la différence entre les ports de sortie imm5 et imm32 ?
- A quoi sert le signal PC_Hold dans le contrôleur ?

Notre bibliothèque de portes



JEU D'INSTRUCTIONS P-ARM

Instructions P-ARM

- Il s'agit d'un sous-ensemble du jeu d'instructions ARMv7-M
- Nous considérons uniquement les instructions codées sur 16 bits : codage Thumb 1
- Parmi les 12 types d'instructions, nous nous limitons à 4 types
  - La documentation du projet présente les 29 instructions
  - Pour les détails, se référer à la documentation ARM (chapitre 5, page 127)

4 Types d'instructions

a) **Shift, add, sub, mov,**
   - 8 instructions

b) **Data Processing,**
   - 16 instructions

c) **Load/Store,**
   - 2 instructions

d) **Stack pointer,**
   - 2 instructions

e) **Branch**
   - 1 instruction

Table A5-1 16-bit Thumb instruction encoding

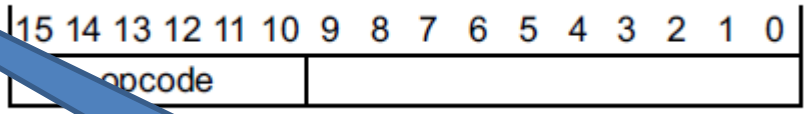
opcode	Instruction or instruction class
00xxxx	*Shift (immediate), add, subtract, move, and compare* on page A5-128
010000	*Data processing* on page A5-129
010001	*Special data instructions and branch and exchange* on page A5-130
01001x	Load from Literal Pool, see *LDR (literal)* on page A7-254
0101xx 011xxx 100xxx	*Load/store single data item* on page A5-131
10100x	Generate PC-relative address, see *ADR* on page A7-197
10101x	Generate SP-relative address, see *ADD (SP plus immediate)* on page A7-193
1011xx	*Miscellaneous 16-bit instructions* on page A5-132
11000x	Store multiple registers, see *STM, STMIA, STMEA* on page A7-422
11001x	Load multiple registers, see *LDM, LDMIA, LDMFD* on page A7-248
1101xx	*Conditional branch, and supervisor call* on page A5-134
11100x	Unconditional Branch, see *B* on page A7-207

Code d'instruction		Catégorie A	Catégorie B	Catégorie C	Catégorie D	Catégorie E
00 XX XX	Shift, add, sub…	1				
01 00 00	Data processing		1			
10 01 XX	Load/Store			1		
10 11 XX	Stack pointer				1	
11 01 XX	Branch					1

Codage Thumb2 16 bits

Instructions pour le projet

1. **00xxxx** **Shift (immediate), add, subtract, move, and compare**
2. **010000** ***Data processing***
- 010001 *Special data instructions and branch and exchange*
- 01001x Load from Literal Pool
3. **0101xx**
   011xxx ***Load/store single data***
   100xxx
- 10100x Generate PC-relative address
- 10101x Generate SP-relative address
- 1011xx *Miscellaneous 16-bit instructions*
- 11000x Store multiple registers
- 11001x Load multiple registers
4. **1101xx** ***Conditional branch, and supervisor call***
- 11100x Unconditional Branch

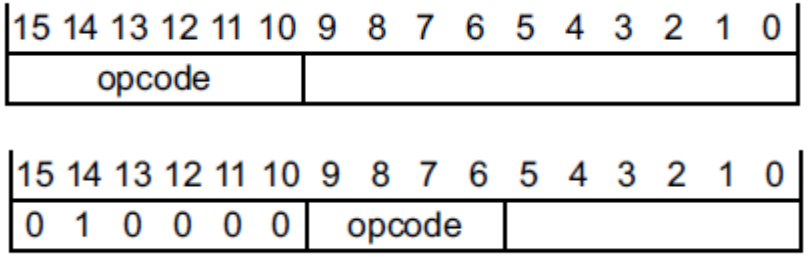


Commençons par le type le plus simple

2. Data processing instructions

Opcode 010000 puis :

- AND, logical And
- EOR, Exclusive Or
- LSL, Logical Shift Left Register
- LSR, Logical Shift Right Register
- ASR, Arithmetic Shift Right Register
- ADC, Add with Carry
- SBC, SuBstrate with Carry
- ROR, ROtate Right
- TST, Set flags
- RSB, Reverse substract from 0
- CMP, Compare Register
- CMN, Compare Negative Register
- LOR, Logical Or
- MUL, Multiply
- BIC, Bit Clear
- MVN, Bitwise Or



opcode	Instruction	See
0000	Bitwise AND	*AND (register)* on page A7-201
0001	Exclusive OR	*EOR (register)* on page A7-239
0010	Logical Shift Left	*LSL (register)* on page A7-300
0011	Logical Shift Right	*LSR (register)* on page A7-304
0100	Arithmetic Shift Right	*ASR (register)* on page A7-205
0101	Add with Carry	*ADC (register)* on page A7-187
0110	Subtract with Carry	*SBC (register)* on page A7-380
0111	Rotate Right	*ROR (register)* on page A7-368
1000	Set flags on bitwise AND	*TST (register)* on page A7-466
1001	Reverse Subtract from 0	*RSB (immediate)* on page A7-372
1010	Compare Registers	*CMP (register)* on page A7-231
1011	Compare Negative	*CMN (register)* on page A7-227
1100	Logical OR	*ORR (register)* on page A7-336
1101	Multiply Two Registers	*MUL* on page A7-324
1110	Bit Clear	*BIC (register)* on page A7-213
1111	Bitwise NOT	*MVN (register)* on page A7-328

Codage des instructions ALU

- Il s'agit d'effectuer une opération dans l'ALU sur 2 registres provenant du banc de registres
  - Rn et Rm
- Le résultat est stocké dans le même registre qu'une des 2 opérandes
  - Rd = Rn

ANDS <Rdn>,<Rm>

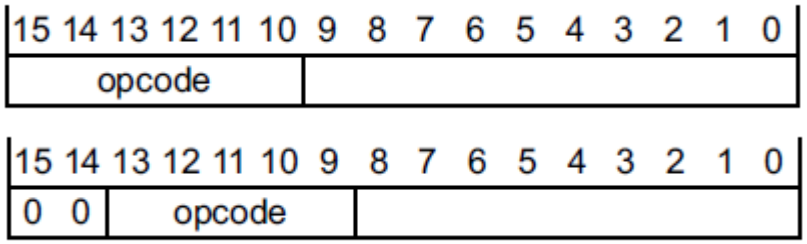
Binaire :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	Rm			Rdn		

1. Shift, add, substract, move and compare

Opcod 00xxxx

- LSL, Logical Shift Left Immediate
- LSR, Logical Shift Right Immediate
- ASR, Arithmetic Shift Right Immediate
- ADD, Add register / Immediate (3 or 8 bits)
- SUB, Sub register / Immediate (3 or 8 bits)
- MOV, Move
- CMP, Compare



Codage des instructions shift…

- Il s'agit d'effectuer des opérations avec un codage étendu par rapport au type b
  - 3 registres différents Rm, Rn, Rd
  - Des calculs sur constantes codées sur 3 ou 5 bits
  - Des déplacement de données de registre à registre
- Exemple avec le déalage Arithmétique :
  - Imm5 code le nombre de décalage de 0 à 31

ASRS <Rd>,<Rm>,#<imm5>

Binaire :

15	14	13	12	11	10 9 8 7 6	5 4 3	2 1 0
0	0	0	1	0	imm5	Rm	Rd

3. Load/Store instructions

- STR, Store Register (from register or imm)
  - **STR**, STRH, STRB
- LDR, Load Register (from register or imm)
  - **LDR**, LDRH, LDRB,
- LDR, Load Register (from register)
  - LDRSH, LDRSB

Data type	Load	Store
32-bit word	LDR	STR
16-bit halfword	-	STRH
16-bit unsigned halfword	LDRH	-
16-bit signed halfword	LDRSH	-
8-bit byte	-	STRB
8-bit unsigned byte	LDRB	-
8-bit signed byte	LDRSB	-
Two 32-bit words	LDRD	STRD

Codage des instructions LDR et STR

- Il s'agit d'effectuer une lecture/load (LDR) ou une écriture/store STR) en mémoire pour accéder aux variables du programme
- Les instructions ont donc besoin des arguments suivants
  - L'adresse de la variable en mémoire (imm sur 8 bits)
  - Le numéro du registre de stockage temporaire (Rt)

STR <Rt>,[SP,#<imm8>]

Binaire :

15	14	13	12	11	10 9 8	7 6 5 4 3 2 1 0
1	0	0	1	0	Rt	imm8

LDR <Rt>,[SP{,#<imm8>}]

Binaire :

15	14	13	12	11	10 9 8	7 6 5 4 3 2 1 0
1	0	0	1	1	Rt	imm8

4. Stack pointer

- Il s'agit de modifier le registre de pointeur de pile SP
- Les variables locales sont accédées de manière relative par rapport à ce pointeur

ADD [SP,] SP,#<imm7>

Binaire :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	0	imm7						

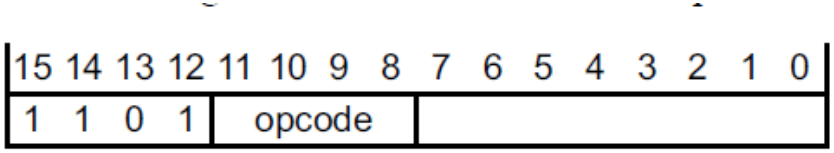
SUB [SP,] SP,#<imm7>

Binaire :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	1	imm7						

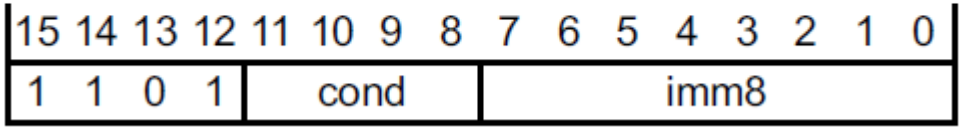
5. Branchements

- Branchement conditionnel avec immédiat sur 8 bits:



opcode	Instruction	See
not 111x	Conditional branch	*B* on page A7-207

B<c> <label>



Si la condition *cond* est vérifiée

– alors PC = PC + Imm32 (extension de Imm8)

cond	Mnemonic extension	Meaning, integer arithmetic	Meaning, floating-point arithmetic[a]	Condition flags
0000	EQ	Equal	Equal	Z == 1
0001	NE	Not equal	Not equal, or unordered	Z == 0
0010	CS [b]	Carry set	Greater than, equal, or unordered	C == 1
0011	CC [c]	Carry clear	Less than	C == 0
0100	MI	Minus, negative	Less than	N == 1
0101	PL	Plus, positive or zero	Greater than, equal, or unordered	N == 0
0110	VS	Overflow	Unordered	V == 1
0111	VC	No overflow	Not unordered	V == 0
1000	HI	Unsigned higher	Greater than, or unordered	C == 1 and Z == 0
1001	LS	Unsigned lower or same	Less than or equal	C == 0 or Z == 1
1010	GE	Signed greater than or equal	Greater than or equal	N == V
1011	LT	Signed less than	Less than, or unordered	N != V
1100	GT	Signed greater than	Greater than	Z == 0 and N == V
1101	LE	Signed less than or equal	Less than, equal, or unordered	Z == 1 or N != V
1110	None (AL) [d]	Always (unconditional)	Always (unconditional)	Any

Pour aller plus loin :
Un exemple d'exécution du processeur

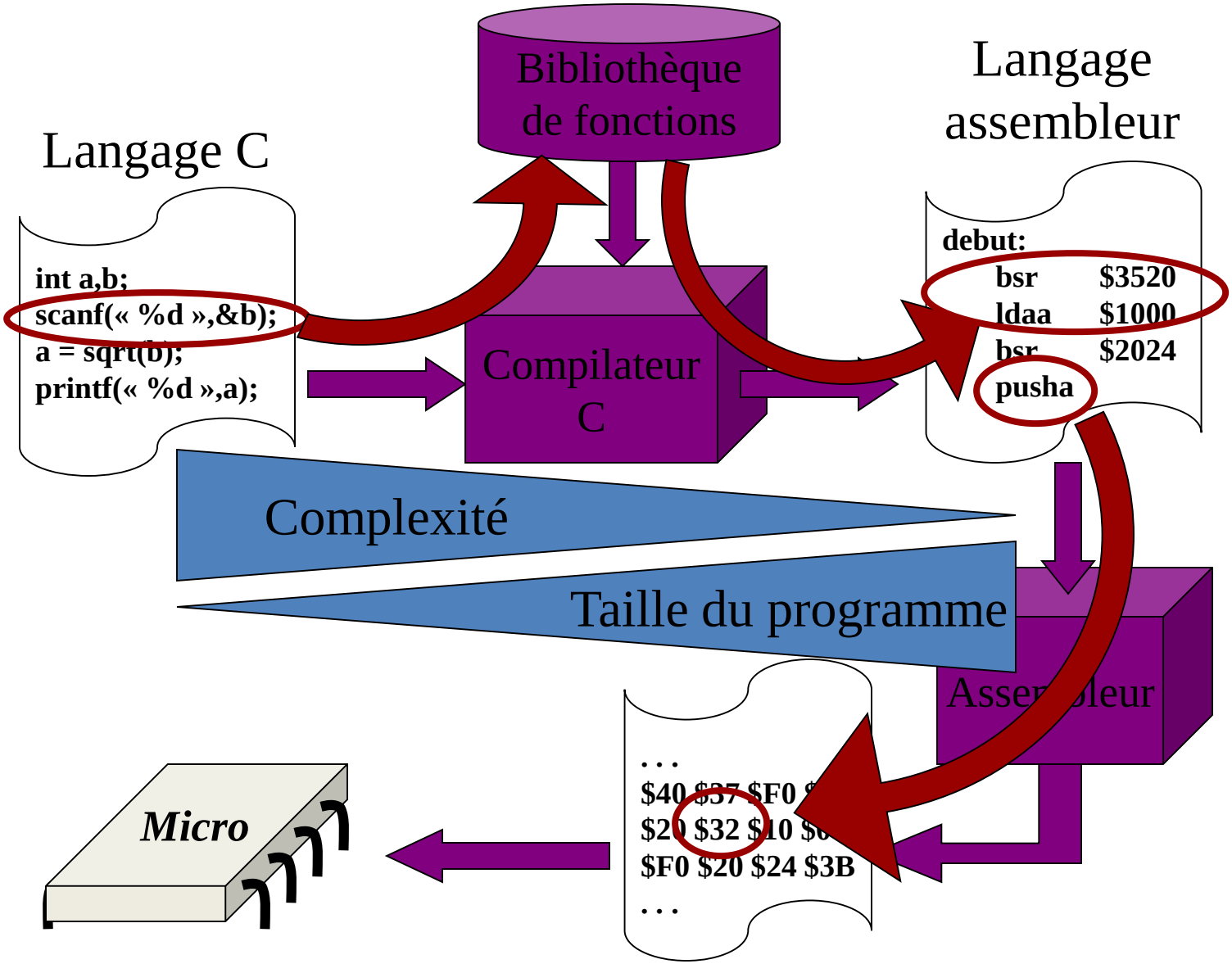
Architecture différente du P-ARM

Exécution du processeur

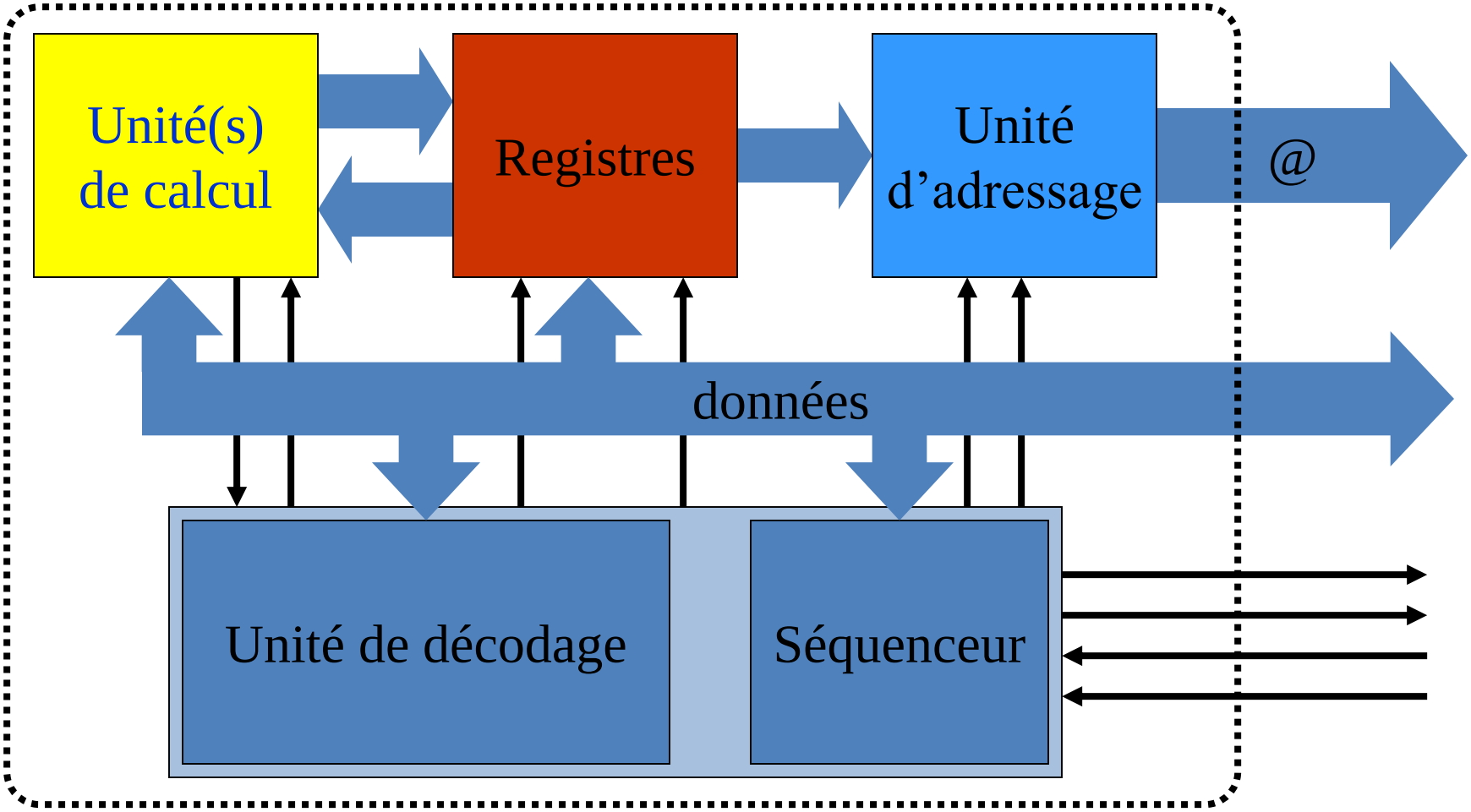
- Exemple sur une architecture à banc de registres

- Composée des 5 parties définies précédemment
  1. Calcul (ALU)
  2. Registres de travail
  3. Transfert mémoire

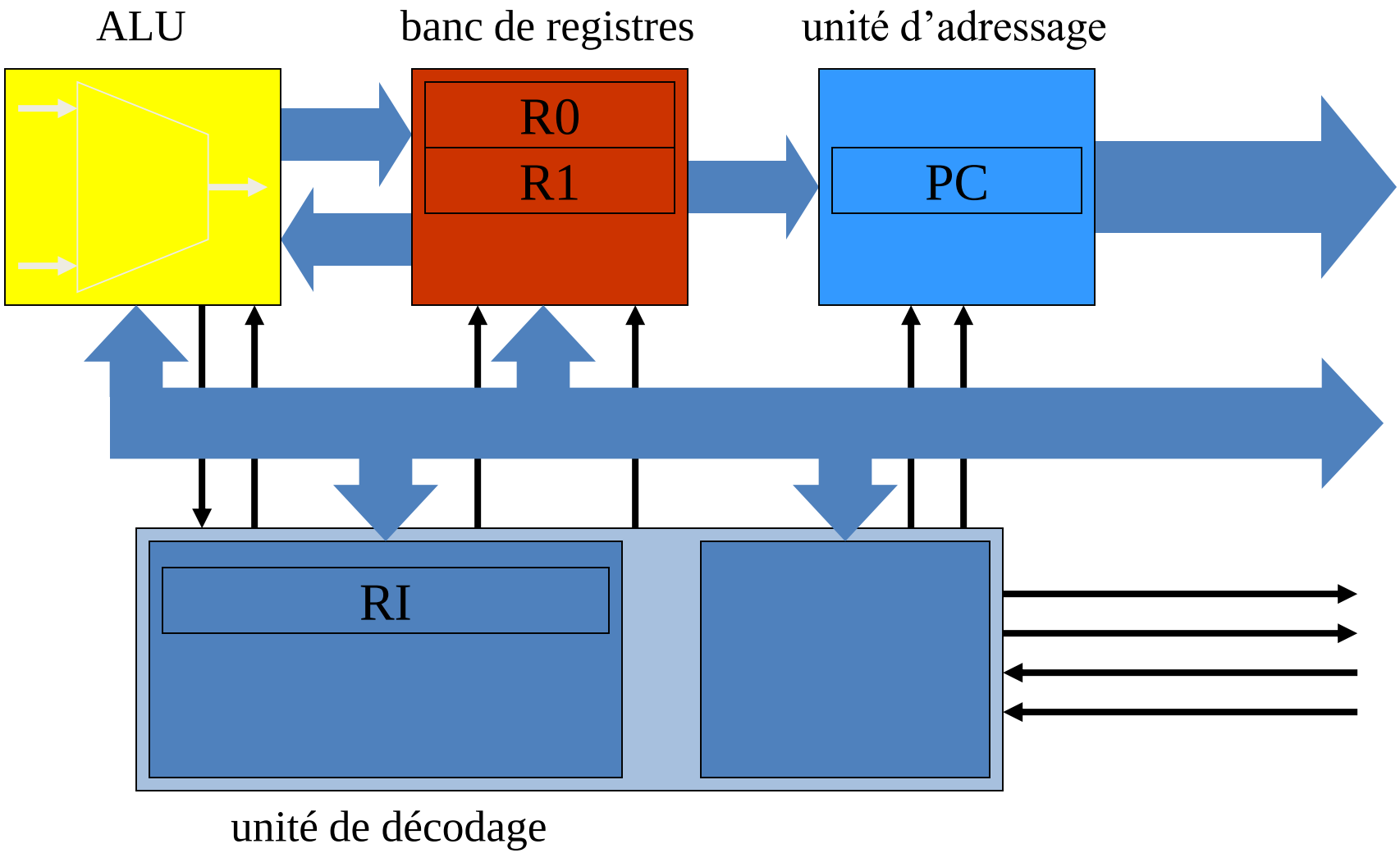
  4. Séquenceur
  5. Décodeur


Bibliothèque de fonctions
Langage C
Langage assembleur
int a,b;
scanf(« %d »,&b);
a = sqrt(b);
printf(« %d »,a);
Compilateur C
debut:
bsr $3520
ldaa $1000
bsr $2024
pusha
Complexité
Taille du programme
Micro
$40 $37 $F0
$20 $32 $10
$F0 $20 $24 $3B

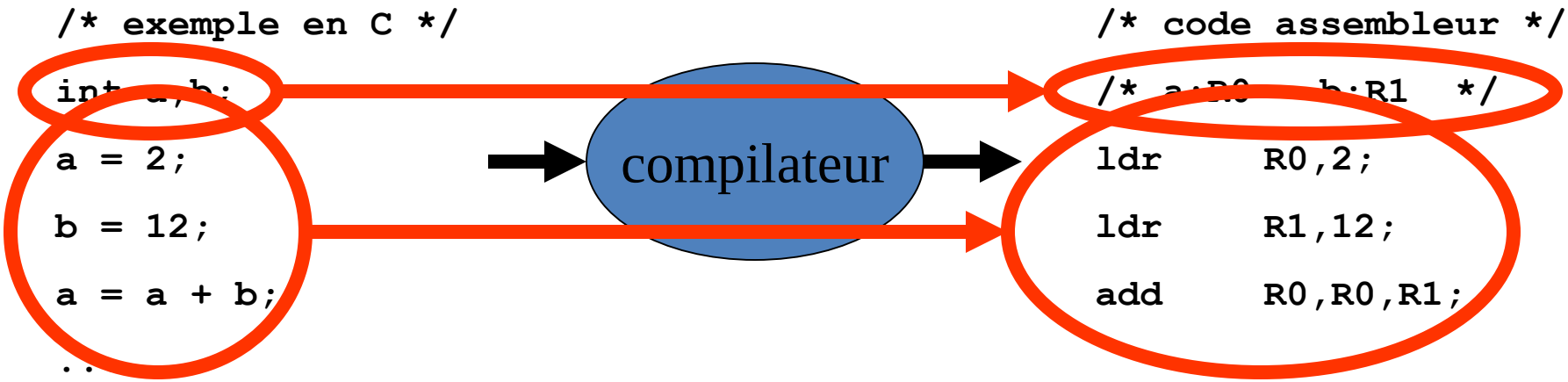
Architecture de base d'un processeur (architecture Von Neumann)



Exécution du programme



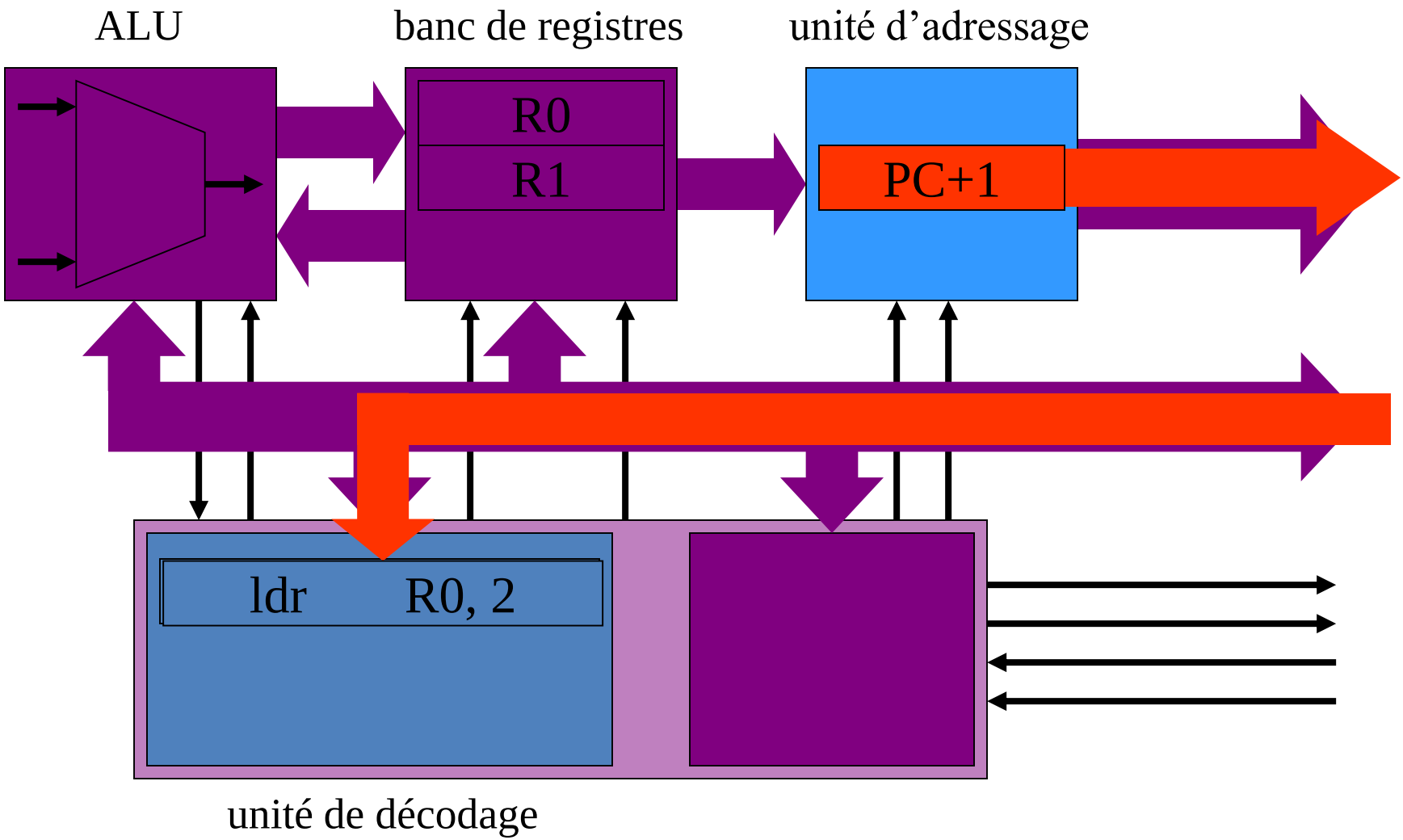
Exemple (simple)



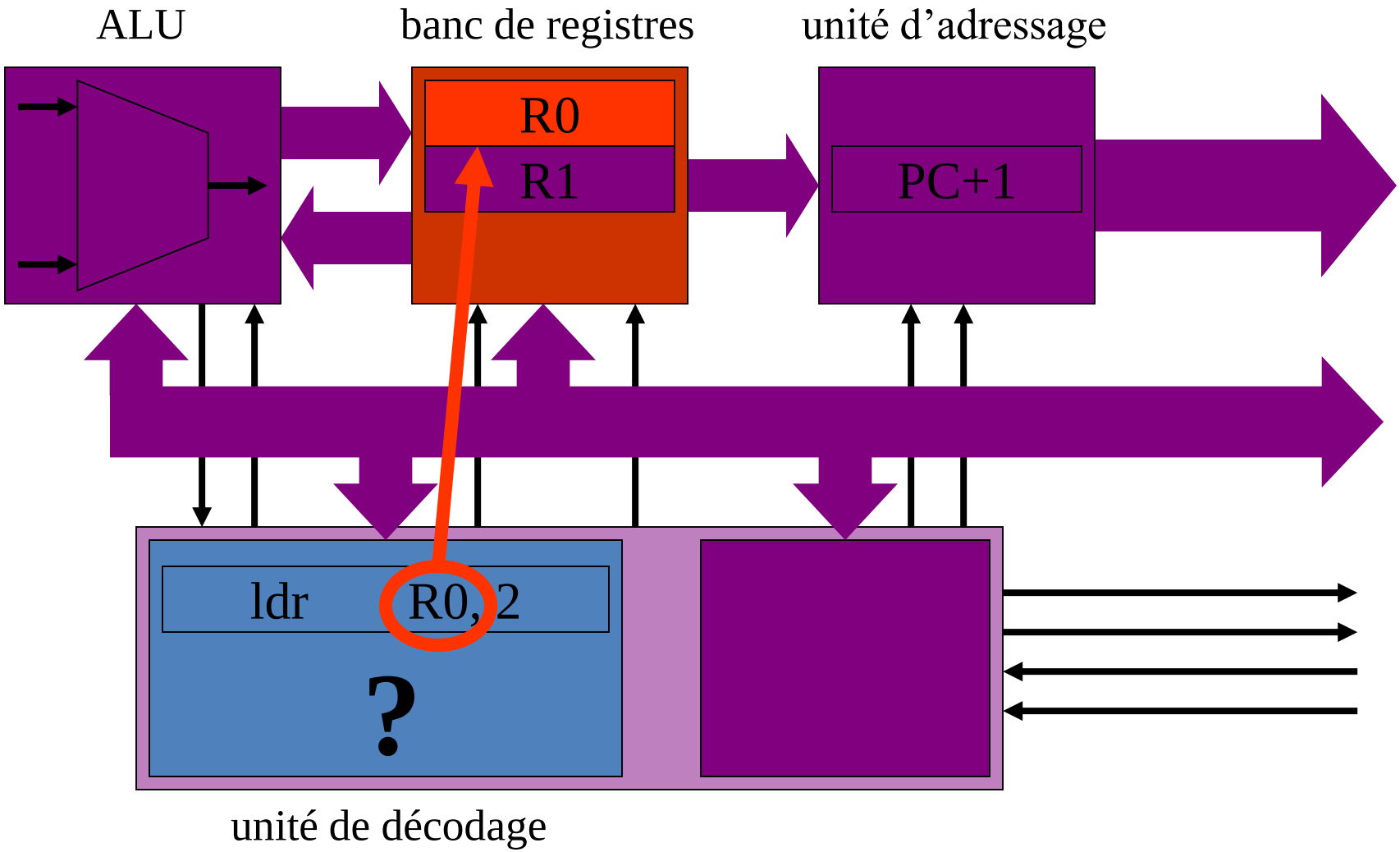
Le rôle du compilateur consiste principalement, ici, à effectuer une allocation des variables du programme (a et b) aux registres disponibles dans le processeur et à traduire les opérations arithmétiques en instructions assembleur.

```
ldr R0,2
```

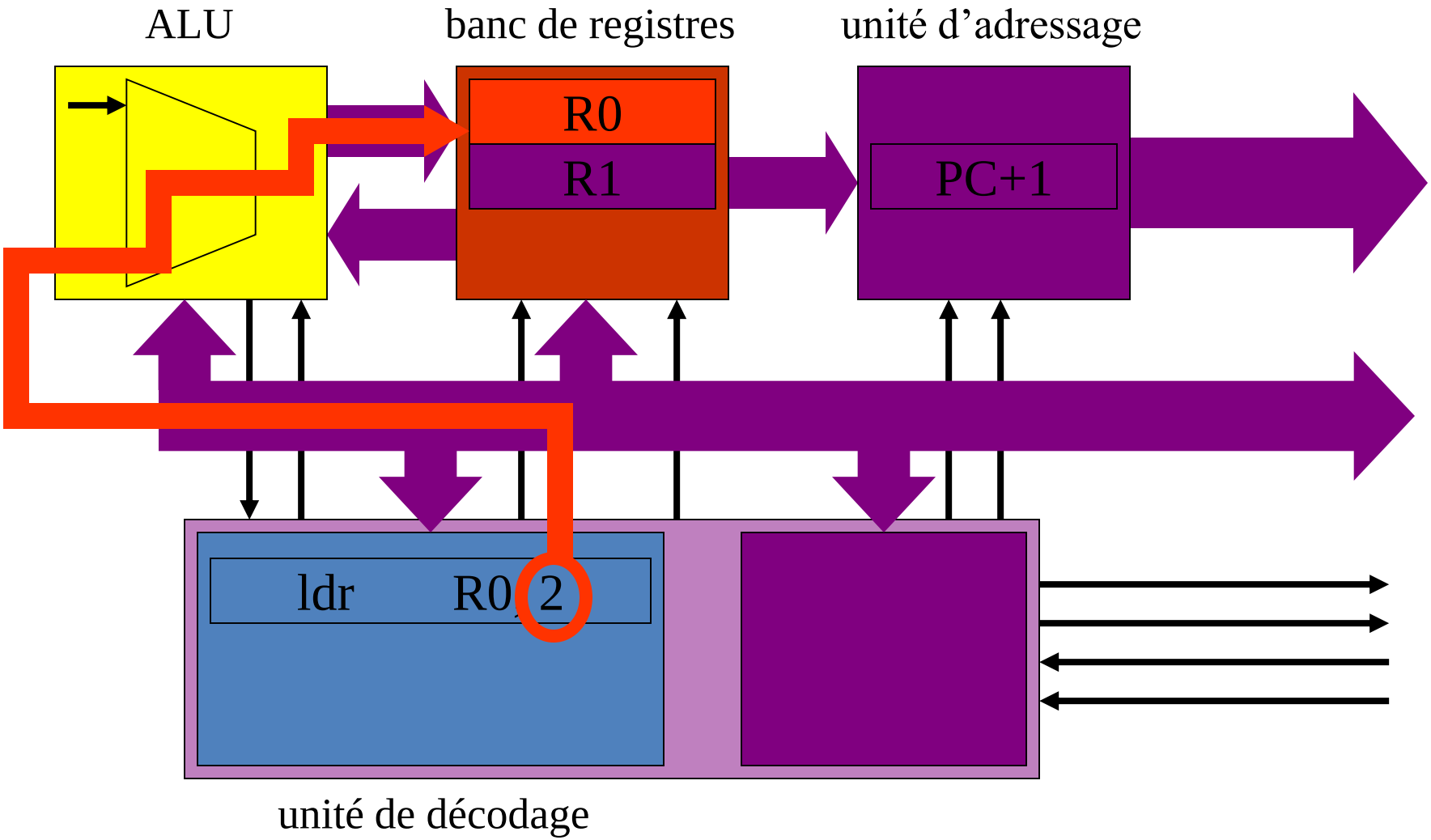
Phase de recherche de l'instruction (I-Fetch)



Phase de décodage de l'instruction (Decode)

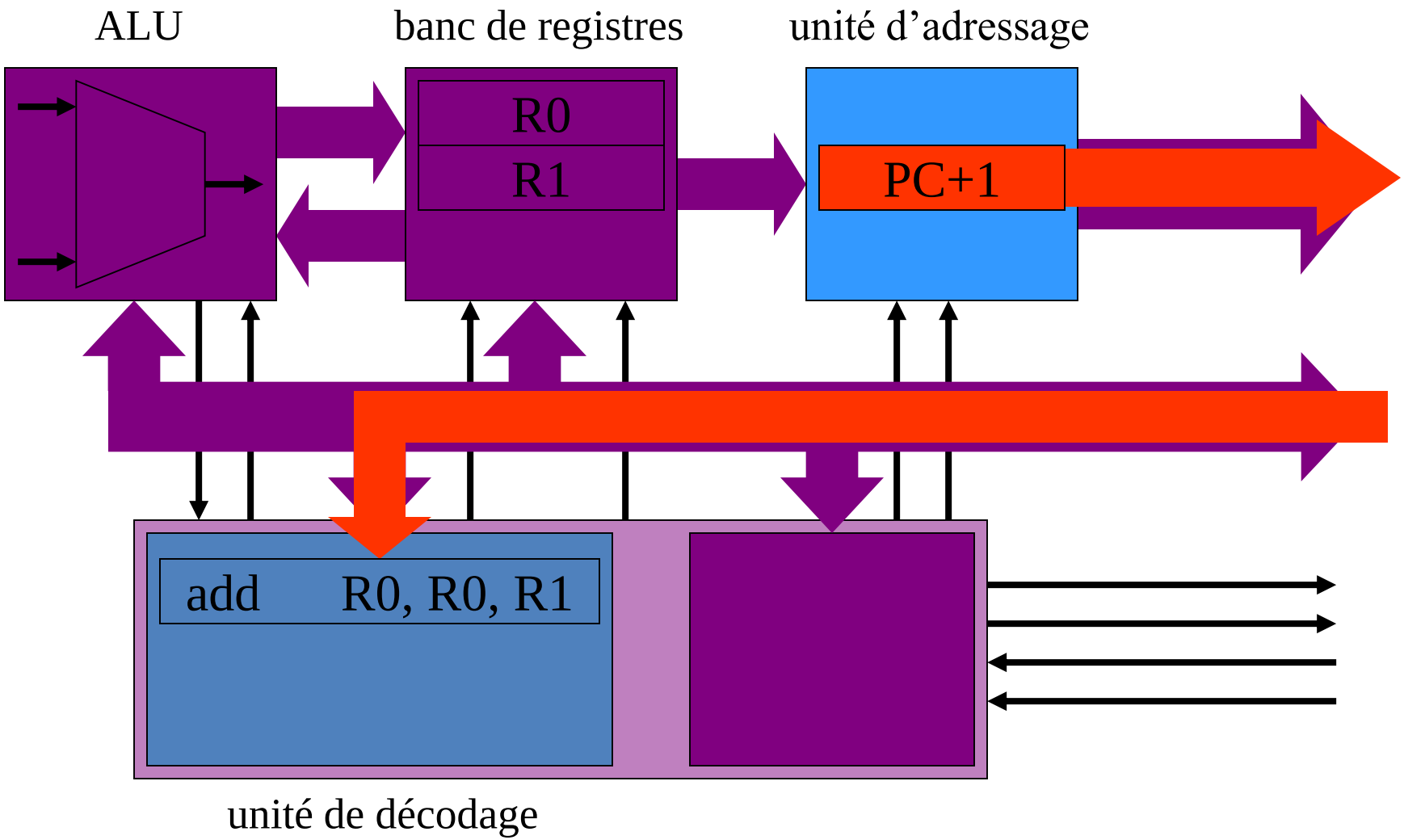


Phase d'exécution de l'instruction (Execute)

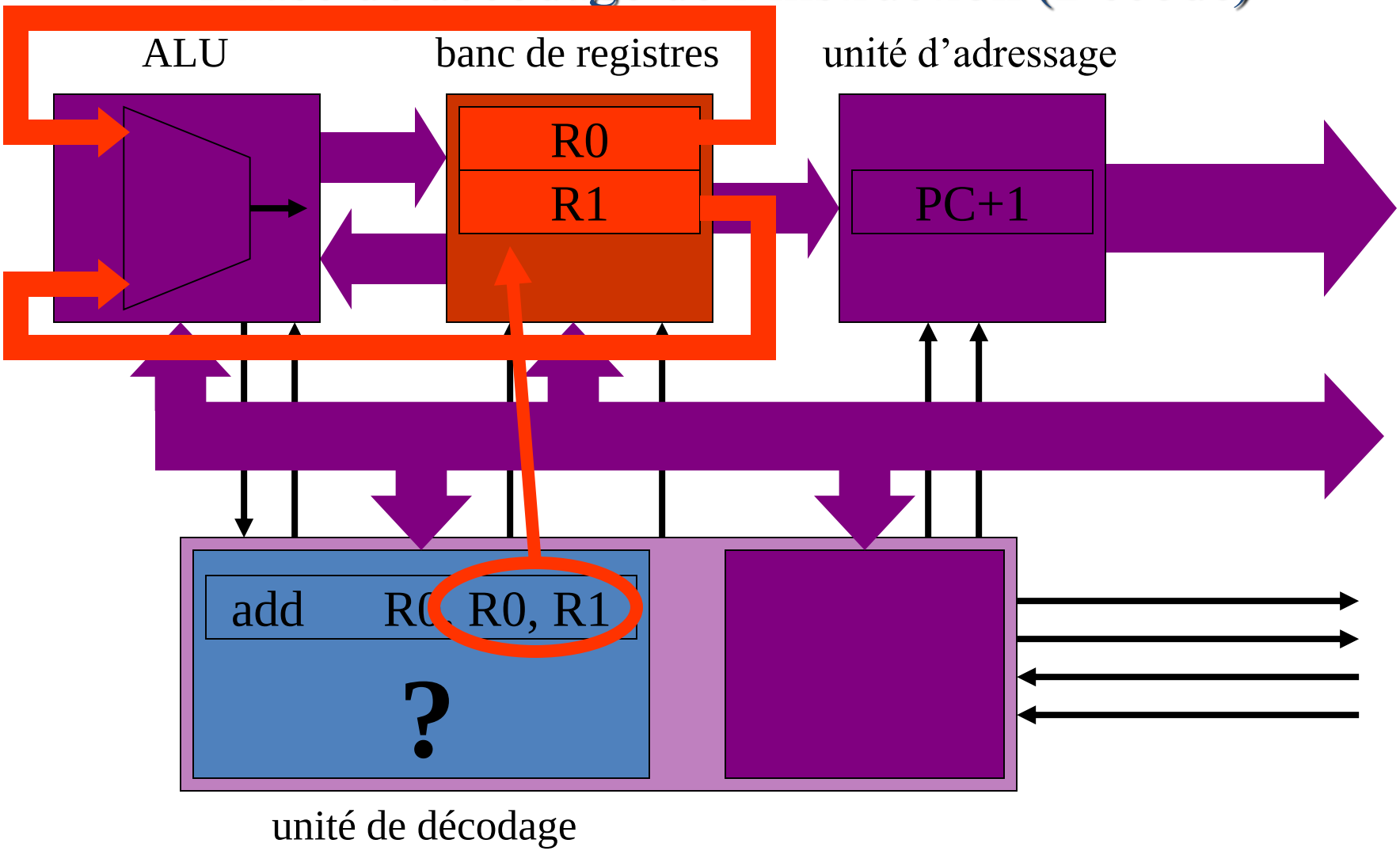


```
add     R0,R0,R1
```

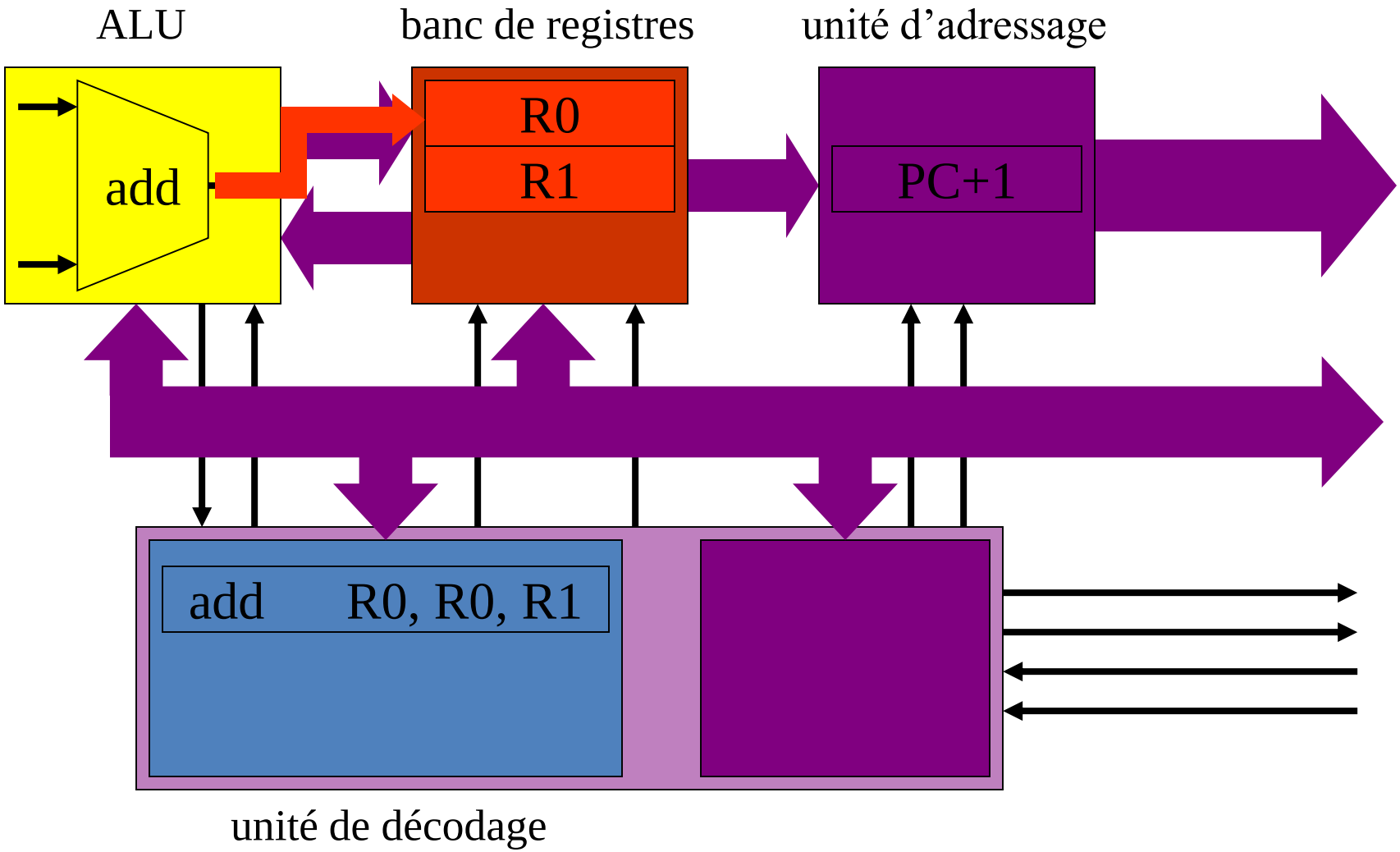
Phase de recherche de l'instruction (I-Fetch)



Phase de décodage de l'instruction (Decode)

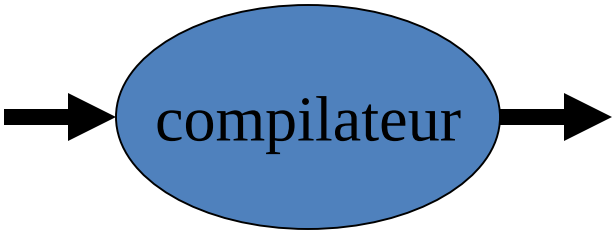


Phase d'exécution de l'instruction (Execute)



Cas des accès en mémoire

```
/* exemple en C */
int Tab[100];
a = Tab[0];
b = 12;
Tab[12] = a + b;
```

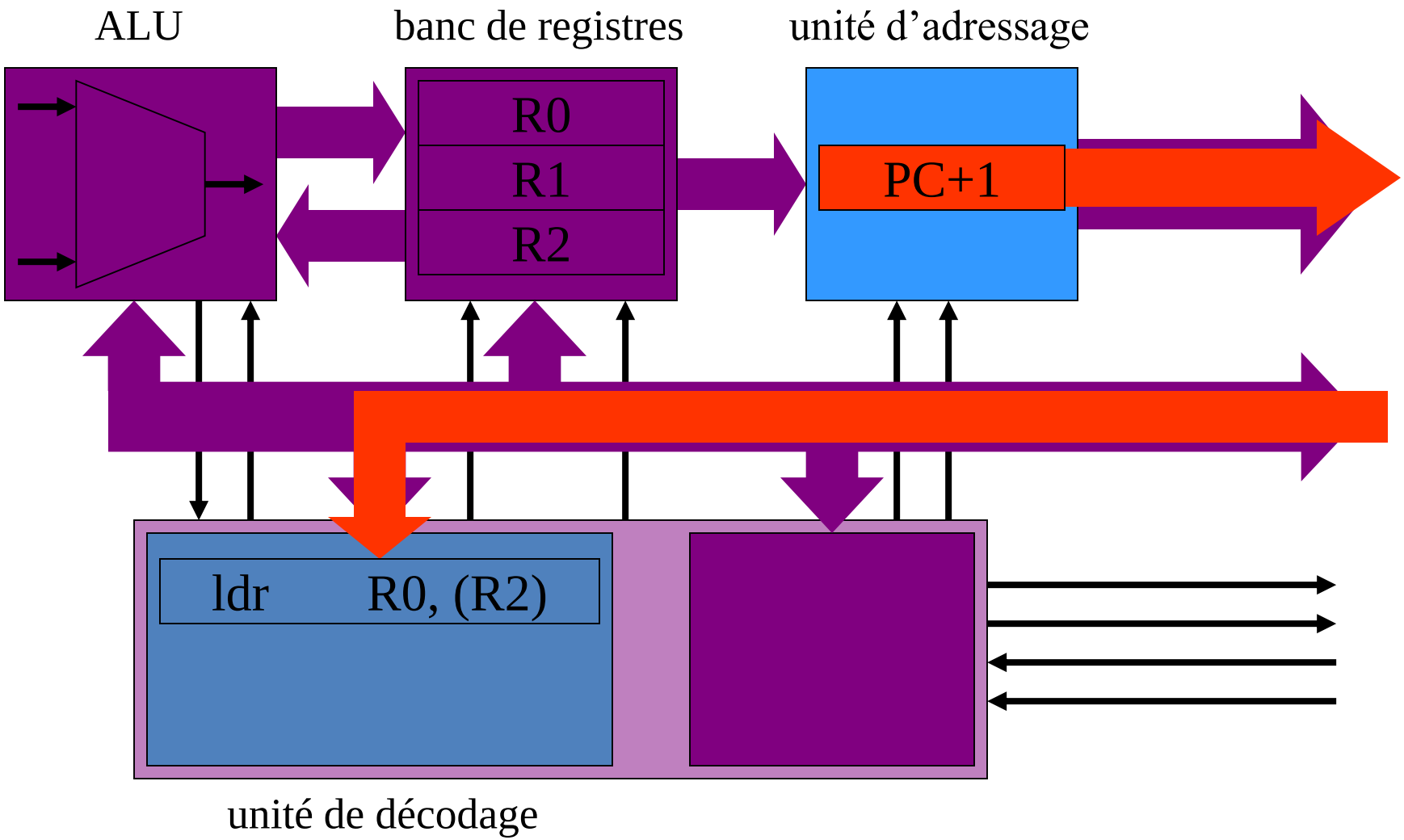



```
/* code assembleur */
/* a:R0   b:R1   Tab:R2 */
ldr     R2,Tab;
ldr     R0,(R2);
ldr     R1,12;
add     R3,R0,R1;
str     (R2),R3;
```

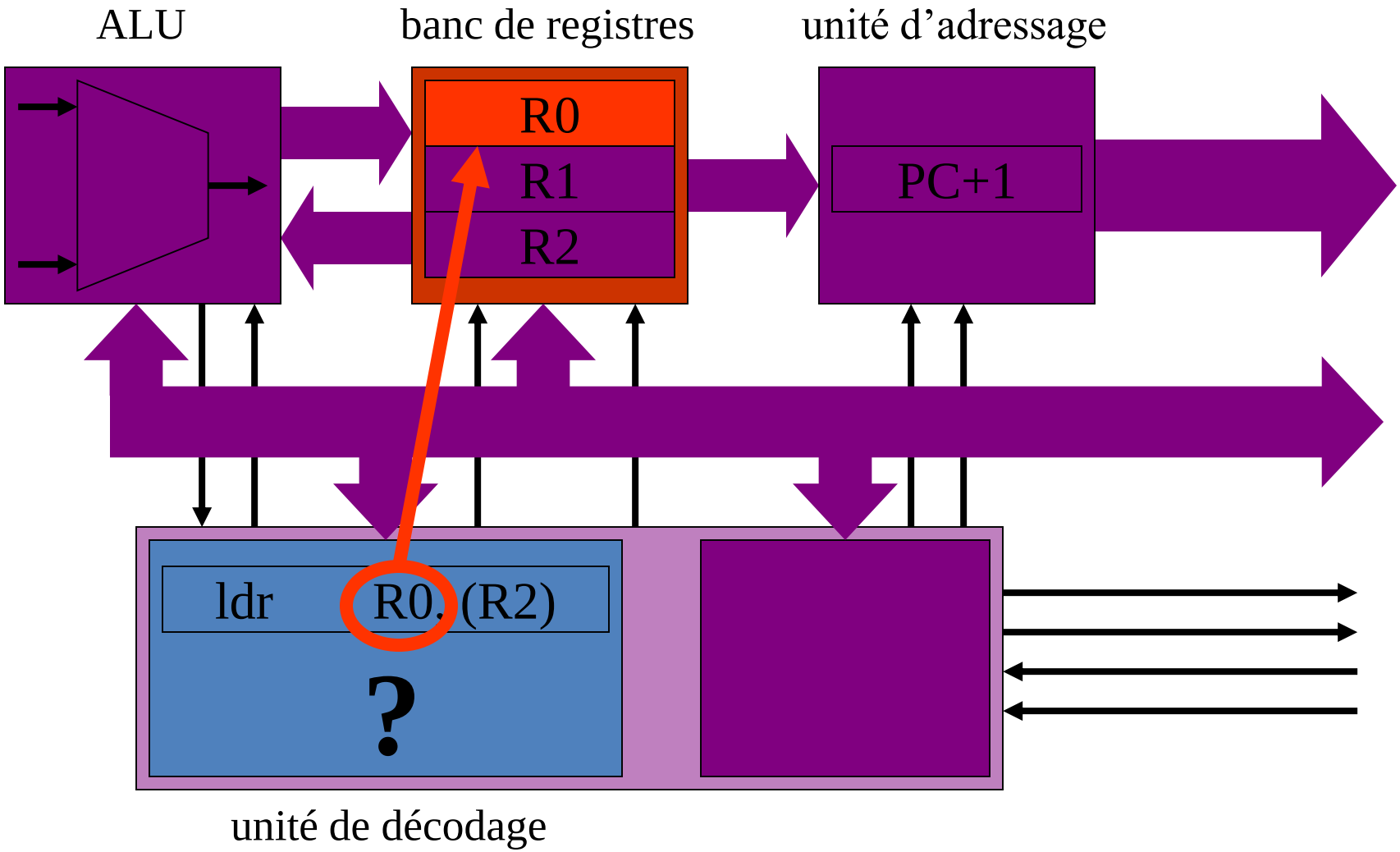
Le compilateur doit effectuer une allocation d'espace mémoire aux structures de données (statiques) utilisées dans le programme. En fonction des cas (espaces mémoire dédiés, mémoires multiples), le code compilé peut être différent.

```
ldr R0,(R2)
```

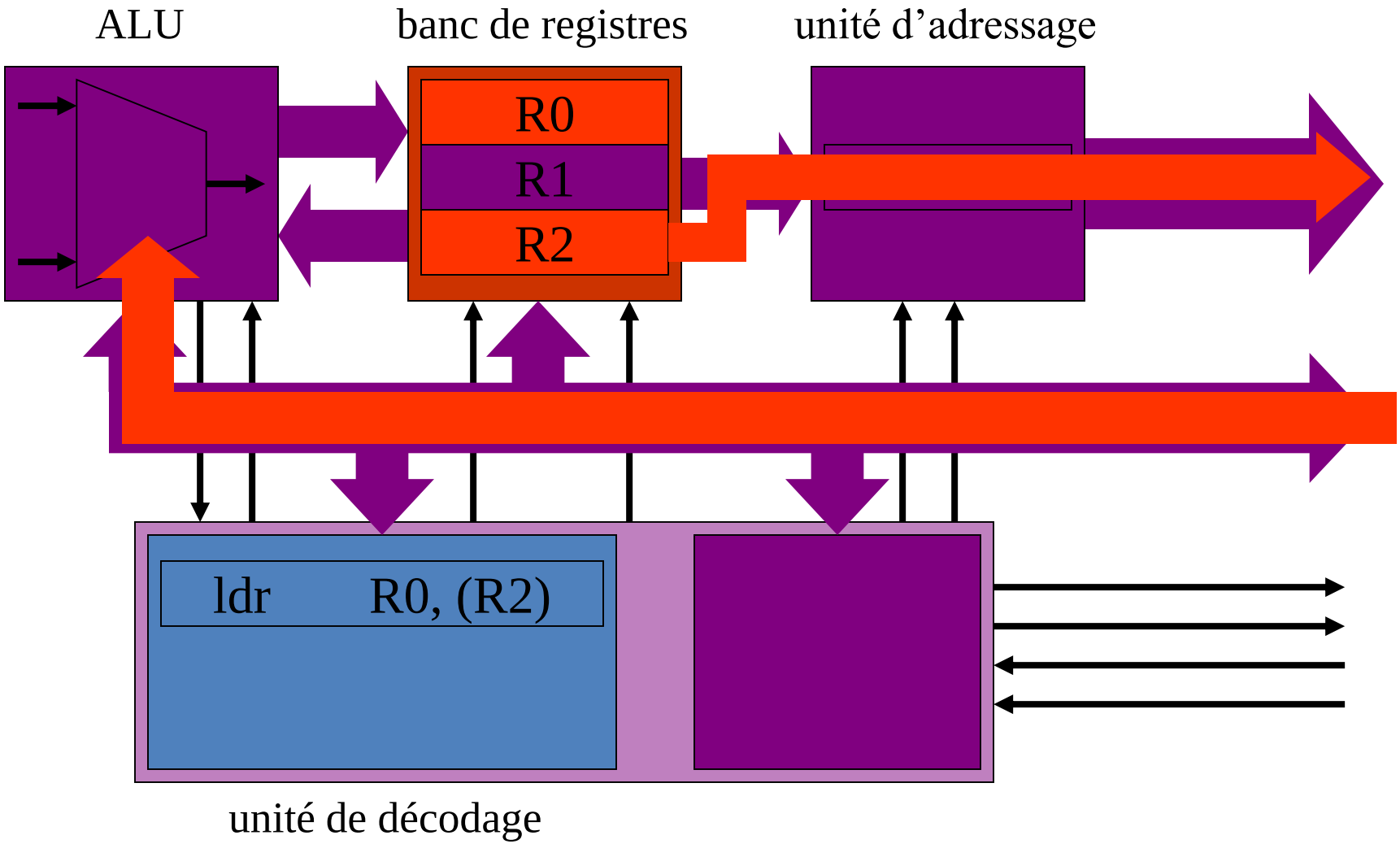
Phase de recherche de l'instruction (I-Fetch)



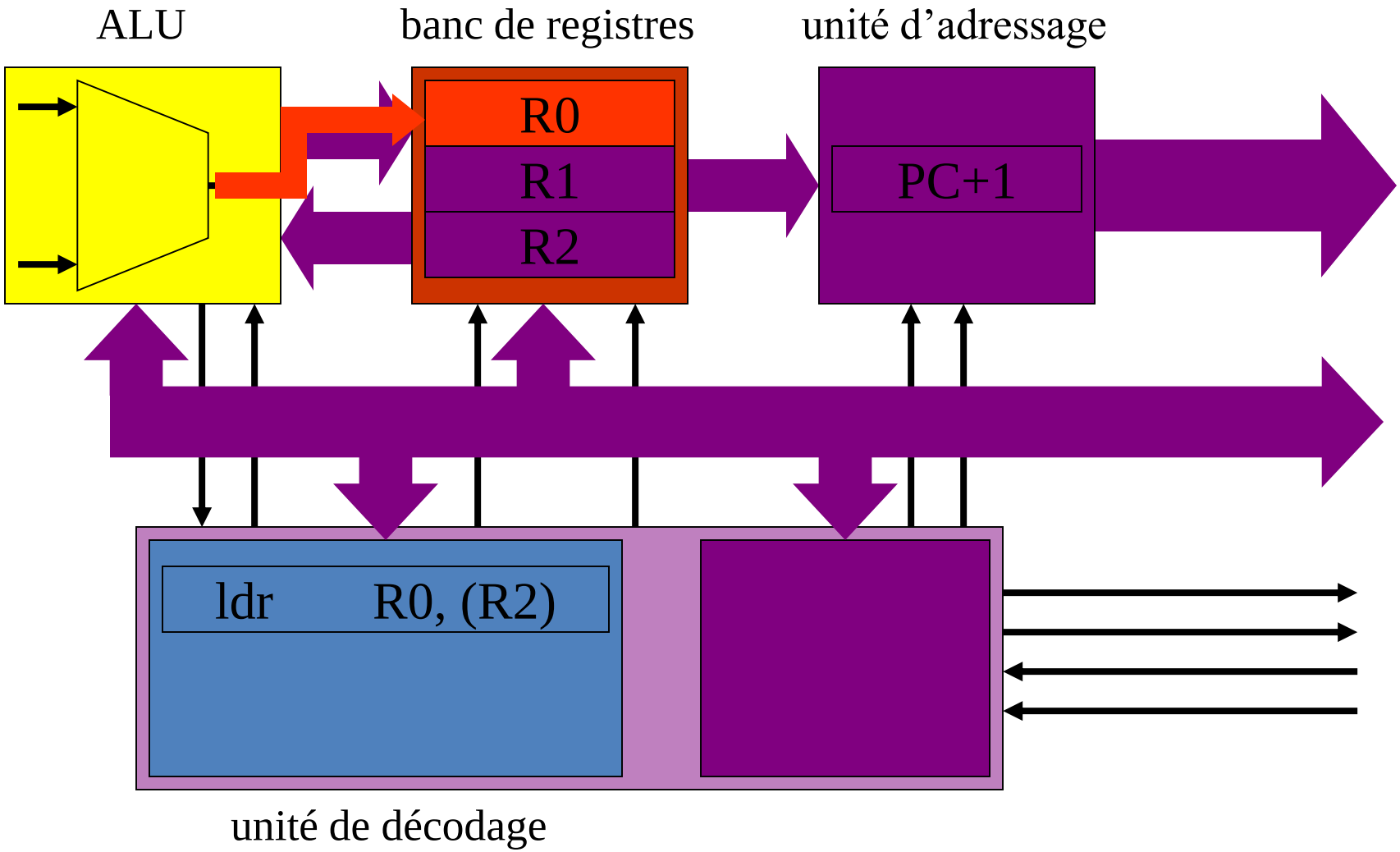
Phase de décodage de l'instruction (Decode)



Phase d'exécution de l'instruction (Execute)

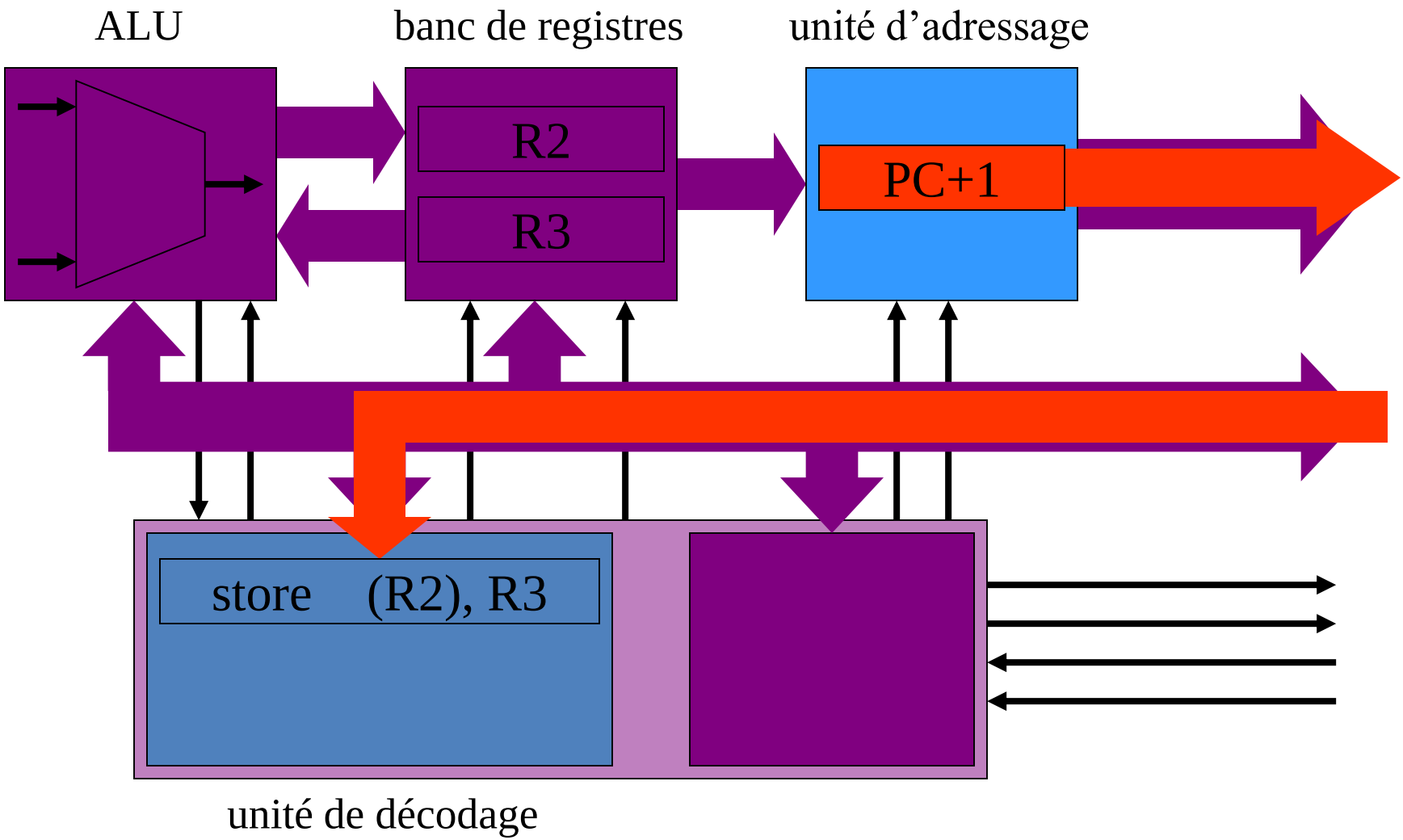


Phase d'écriture du résultat (ReadBack)

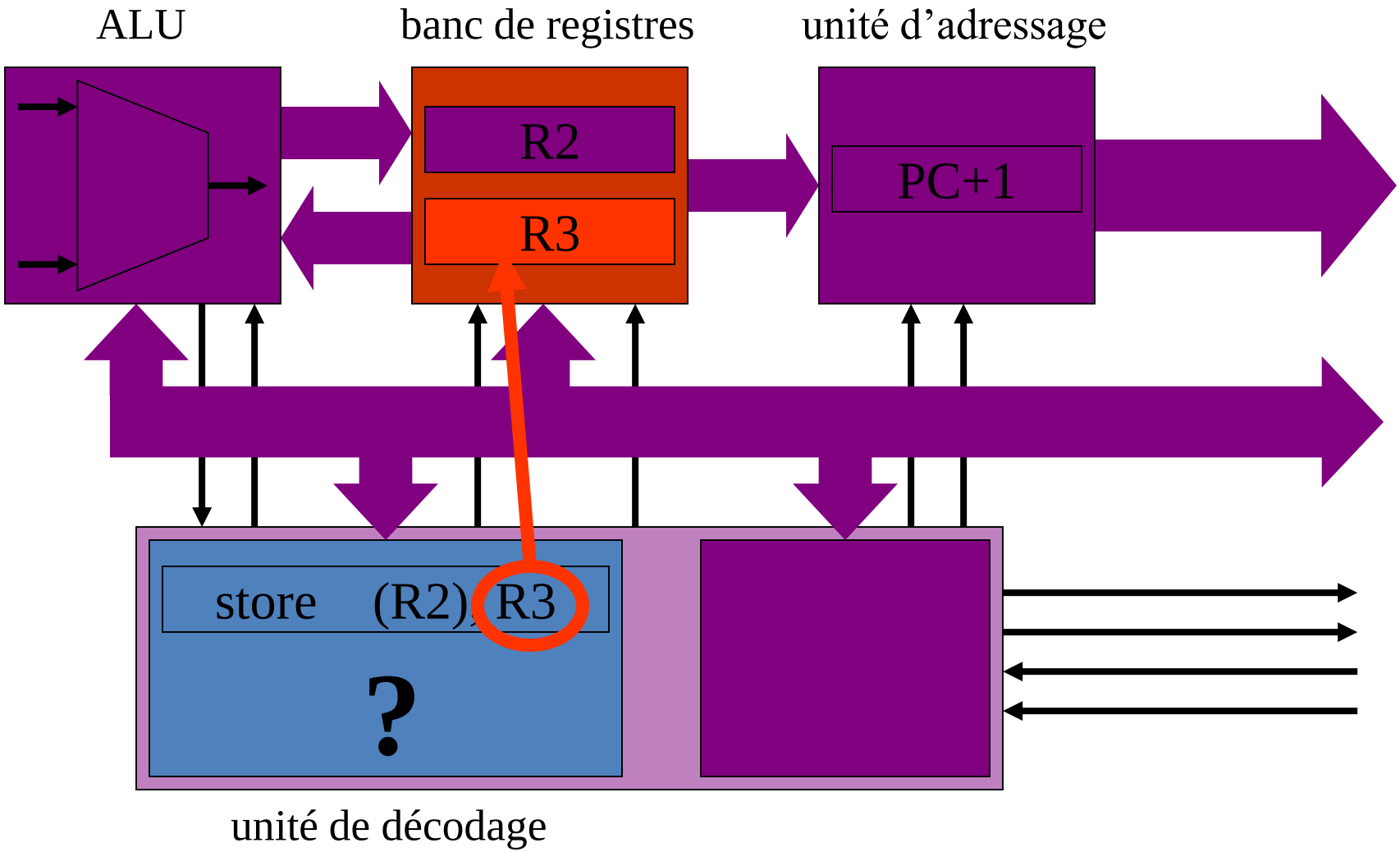


```
store   (R2),R3
```

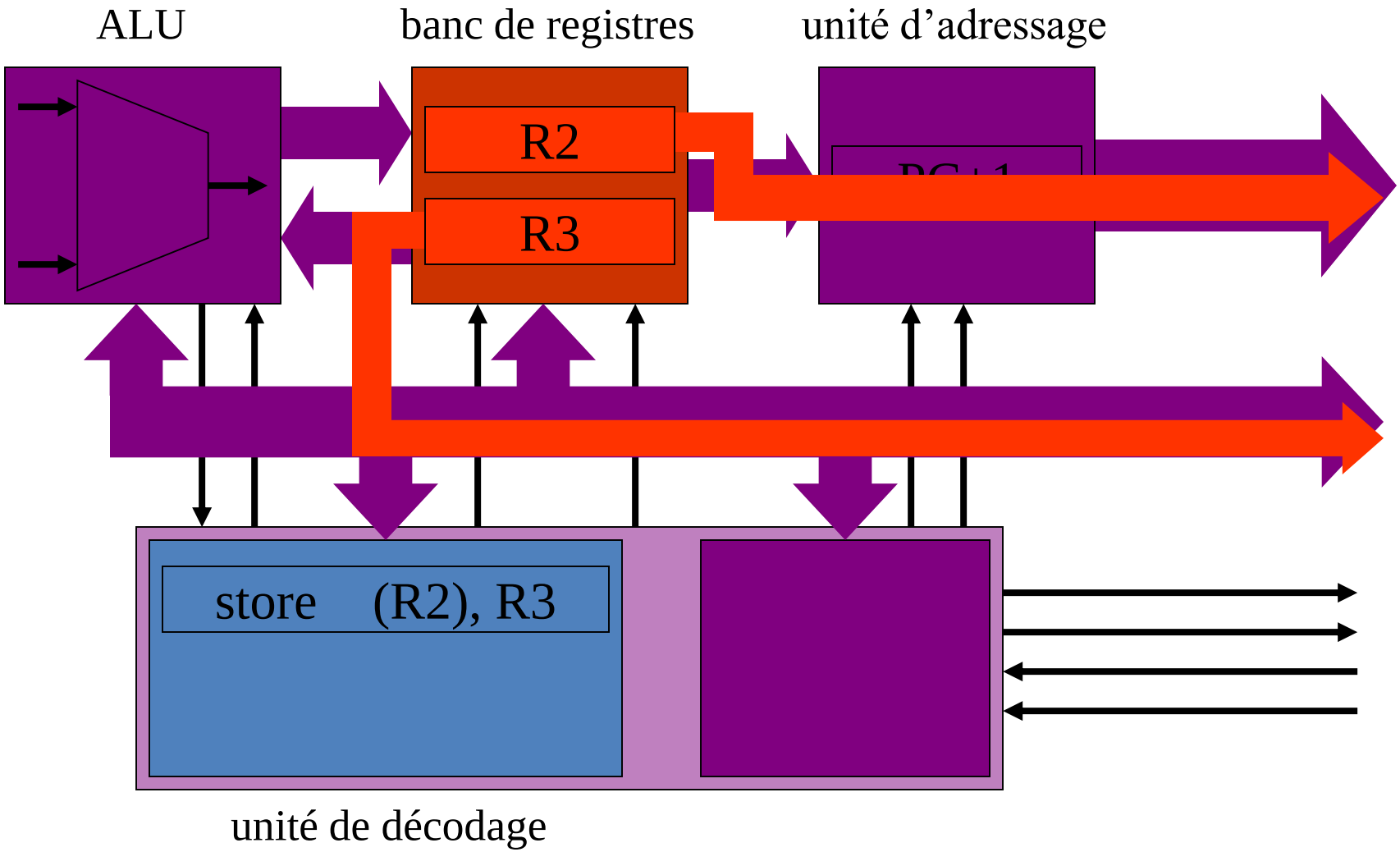
Phase de recherche de l'instruction (I-Fetch)



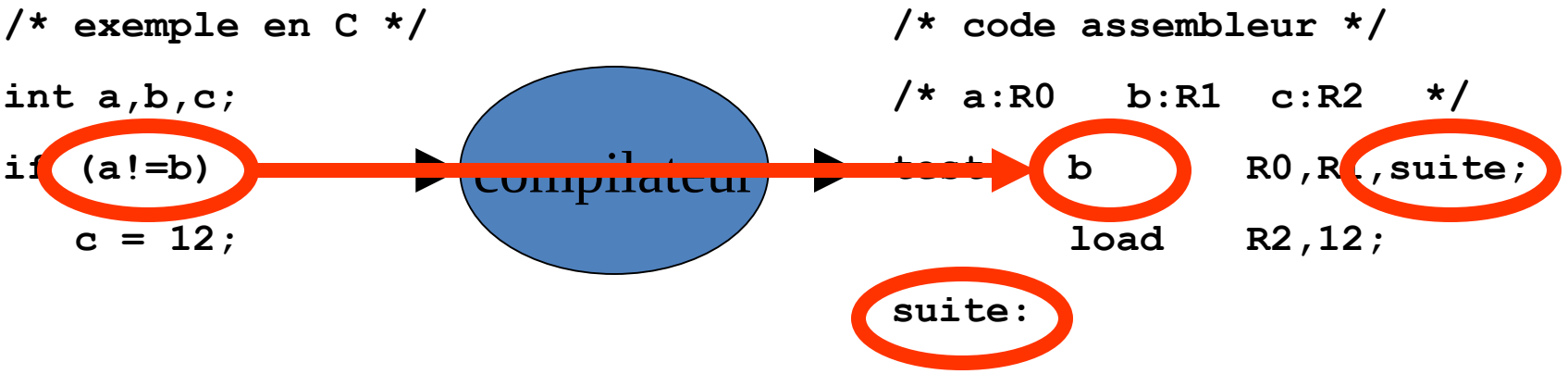
Phase de décodage de l'instruction (Decode)



Phase d'exécution de l'instruction (Execute)

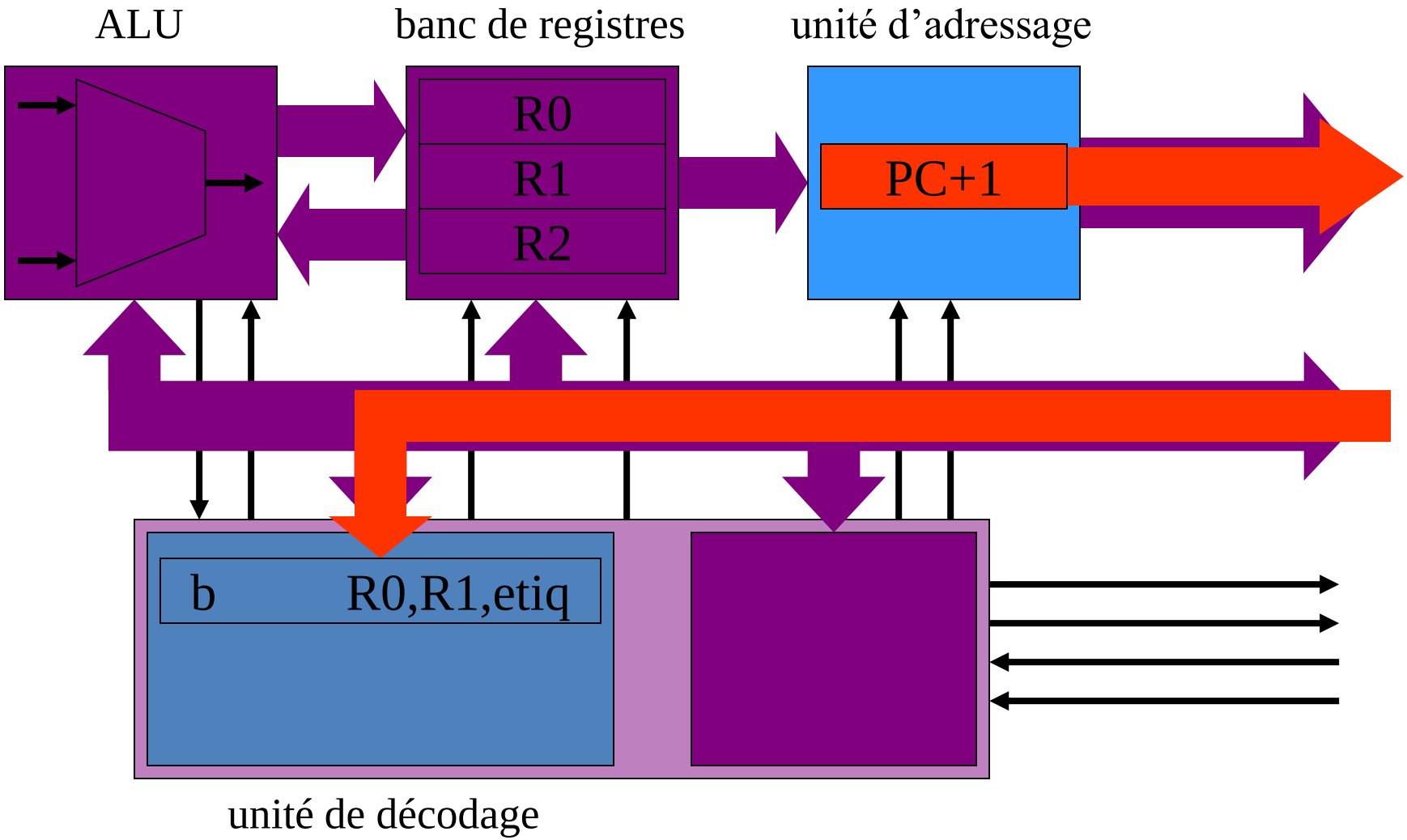


Cas des branchements conditionnels

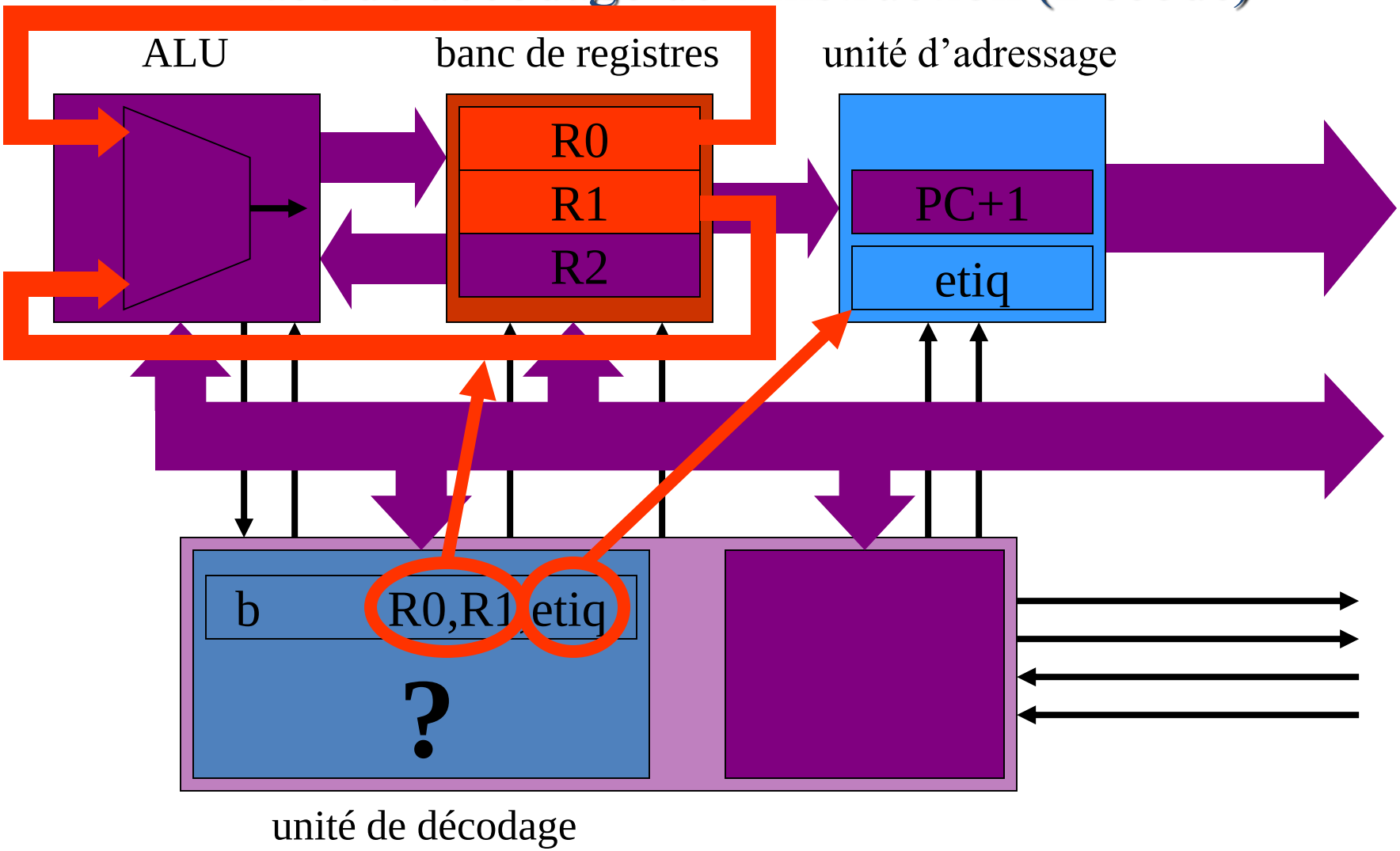


Ici, le compilateur introduit des « étiquettes » dans le programme permettant de localiser les suites d'instructions exécutées de manière conditionnelle. De plus, le compilateur traduit le code en remplaçant la condition : (a!=b) devient beq (***b****ranch if* ***eq****ual*)

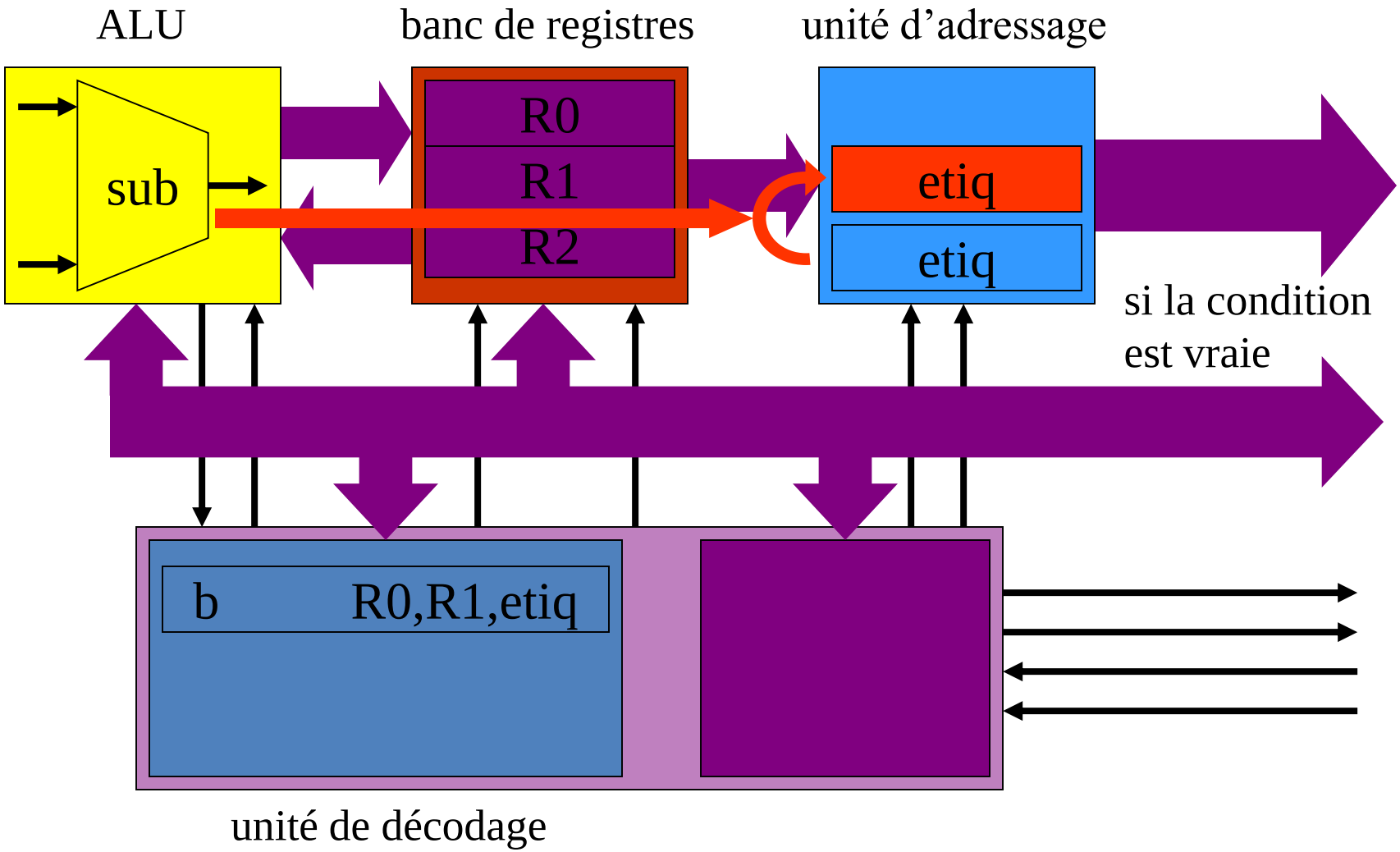
Phase de recherche de l'instruction (I-Fetch)



Phase de décodage de l'instruction (Decode)



Phase d'exécution de l'instruction (Execute)



Résumé

- Une architecture de Von Neumann est constituée de 5 unités:
  1. de Calcul
  2. Banc de Registres
  3. Contrôleur
  4. Mémoire de données
  5. Mémoire d'instructions
- Elle s'exécute en 7 étapes :
  1. Fetch instruction
  2. Incrémenter PC
  3. Décoder l'instruction
  4. Charger données
  5. Exécuter l'opération
  6. Ranger le résultat
  7. Retour
- Le contrôleur utilise les 2 registres suivants :
  - PC
  - RI
- Les échanges avec la mémoire sont de 3 sortes :
  - Données
  - Instructions
  - Adresses