Projet PARM

Processeur Polytech-ARM Cortex-M

B. Miramond

Polytech Nice

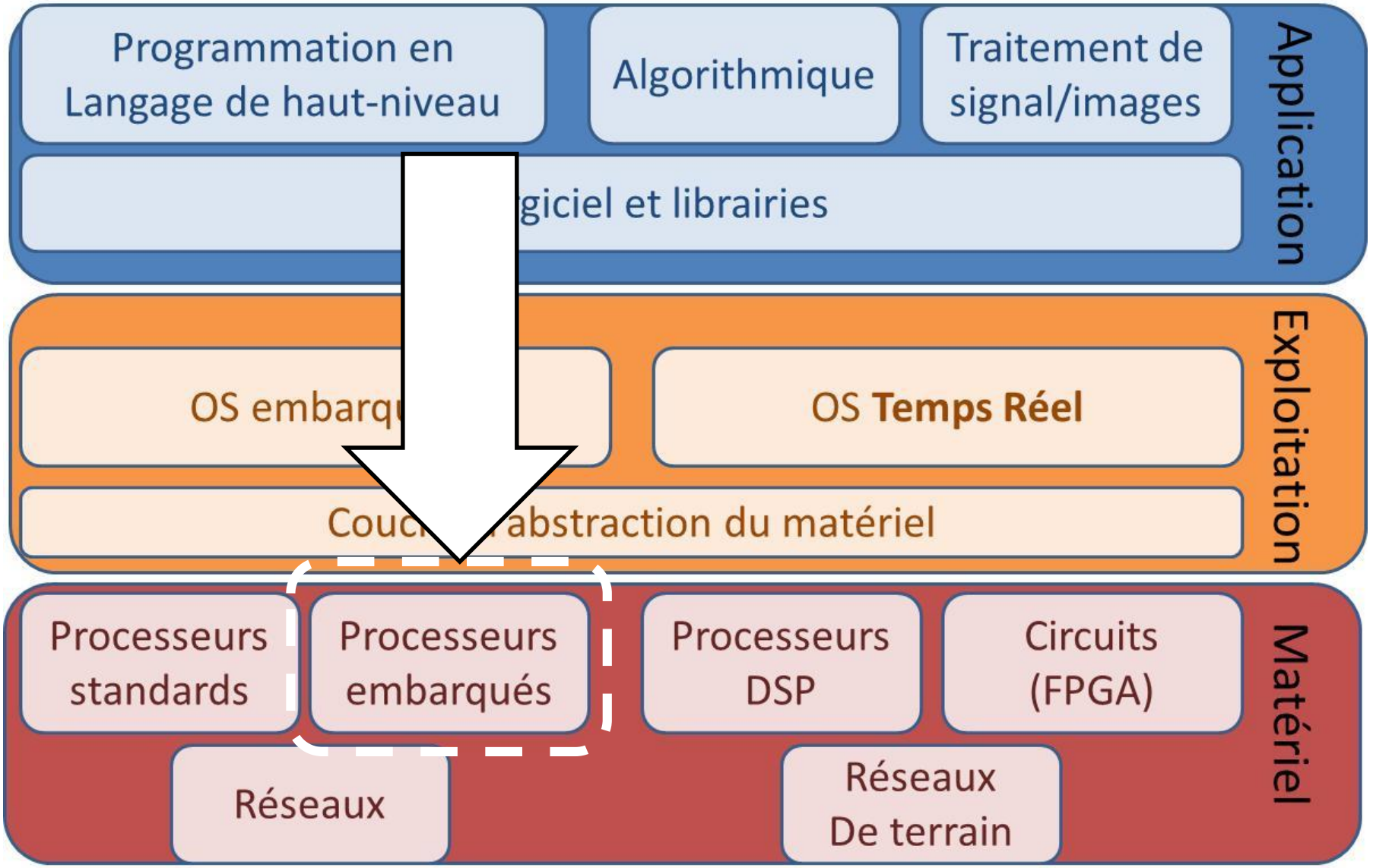
Objectif du module

- Comprendre les mécanismes matériels essentiels de l'informatique.
- Comprendre l'organisation de l'ordinateur autour de l'élément central, le processeur.
- Pour cela, étudier les différentes couches qui structurent la machine et la philosophie de son utilisation.
- Comprendre l'interface entre le logiciel et le matériel

En pratique

- Etudier le jeu d'instruction ARM v7
- Réaliser un simulateur de processeur ARM à travers le **projet P-ARM** : *Polytech ARM-based embedded processor*
- *Concevoir la chaîne permettant de passer d'un code logiciel à son exécution matérielle sur le sous-ensemble d'instructions étudié*

Organisation du système embarqué



Objectifs du projet P-ARM
Polytech ARM-based embedded processor

Concevoir une version allégée du processeur ARM Cortex-M0

1. Utilisation d'un éditeur de circuits numériques
2. Conception matérielle
3. Génération de code binaire depuis l'assembleur, voire depuis du code C
4. Validation par simulation
5. Test de déploiement sur FPGA

Constituer les groupes !

4 personnes par groupe, dont 1 chef d'équipe.

Ré-organisation des efforts du groupe en fonction de l'avancement.

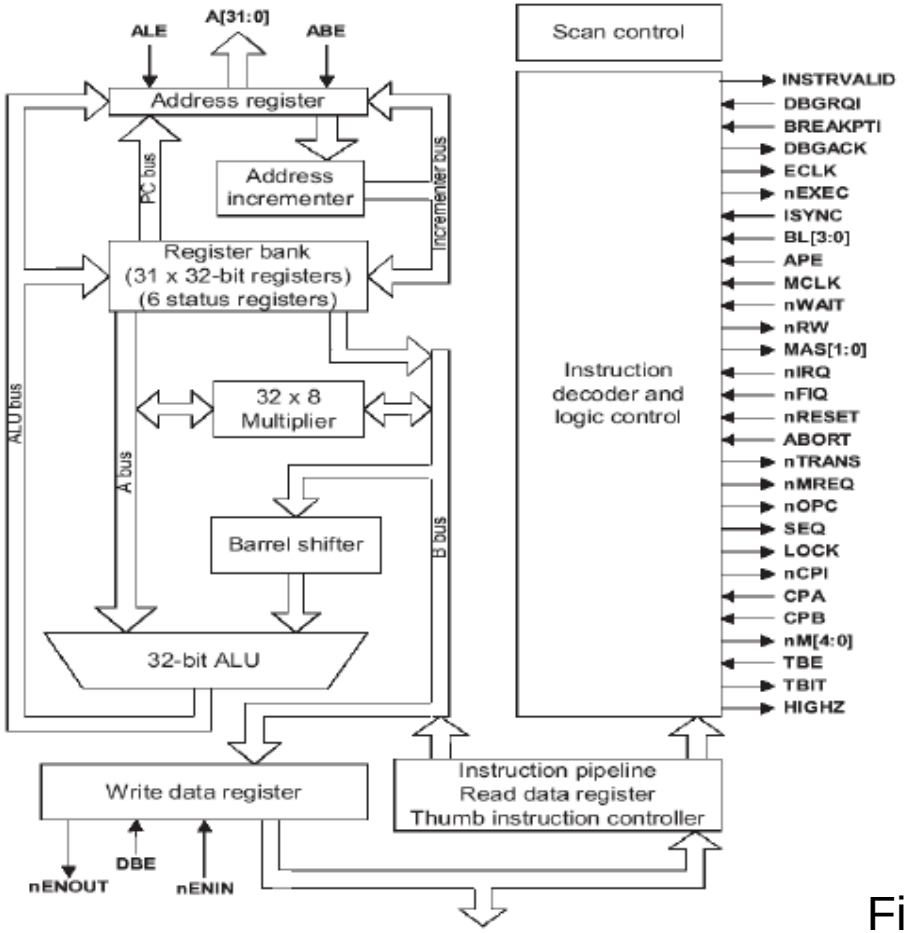
Inscriptions des équipes projets :

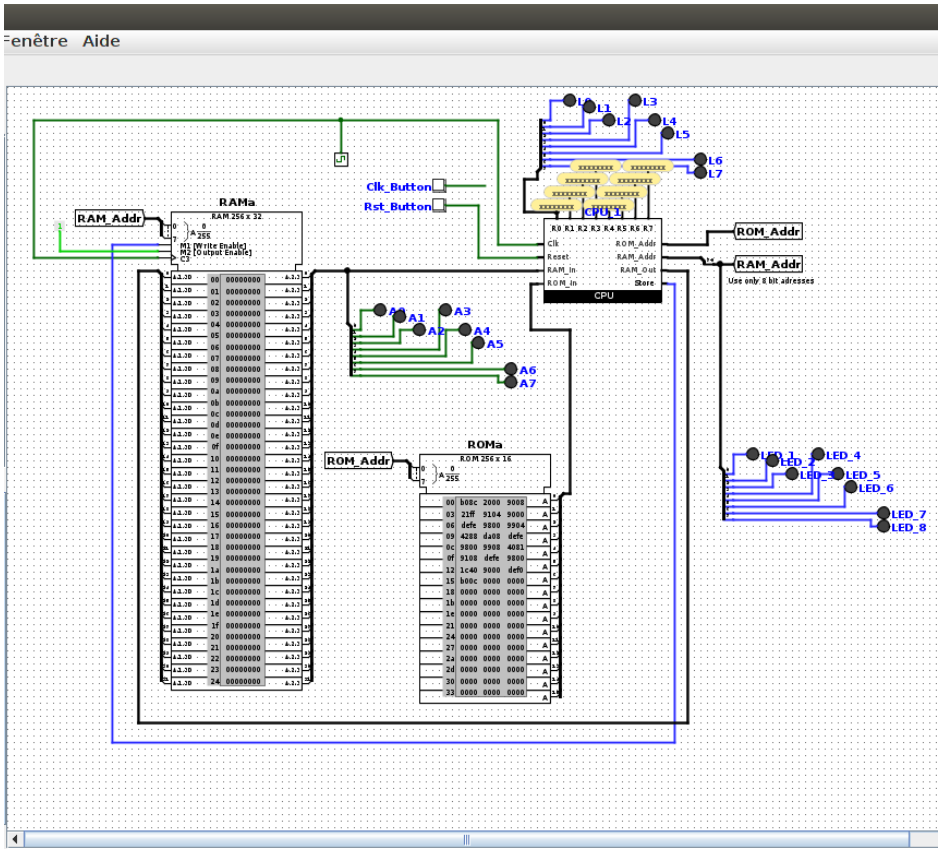
https://lms.univ-cotedazur.fr

Soutenance la première semaine de janvier:

- 15 minutes par groupe (10 min. de prés. / 5 min. de questions)
- 15 planches maximum
- Présentation des parties du projet
- Présentation des résultats de tests avec des exemples
- Répétitions obligatoires

Architecture du processeur P-ARM





Fichiers de démarrage du projet disponibles sur :
lms.univ-cotedazur.fr

Logiciel / Matériel

Assembleur / C

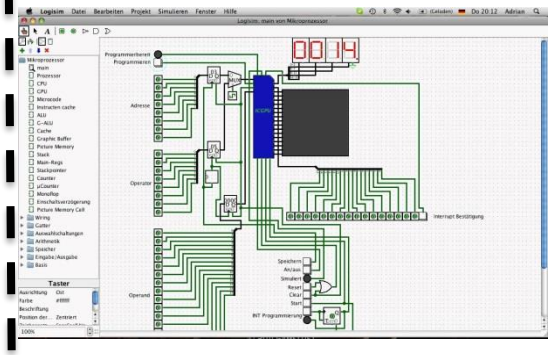
```
pgcd:
while:  cmp R1,R2
                bne do
                b endwhile
do:
if:             cmp R1,R2
                bhi then
                b else
then:   sub R1,R1,R2
                b endif
else:   sub R2,R2,R1
endif:  b while
endwhile:
                mov pc,R14
```

0x2345 4569
0x0124 4828
0x2839 4018
0x3282 0294
…

Fichier binaire d'initialisation des mémoires pour Logisim
+
Désassembleur

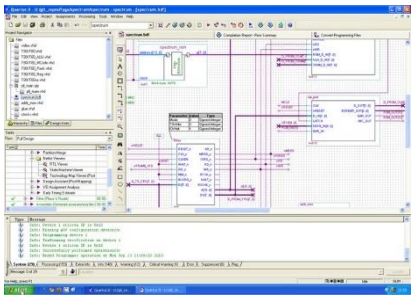
Initialisation de la mémoire dans Logism

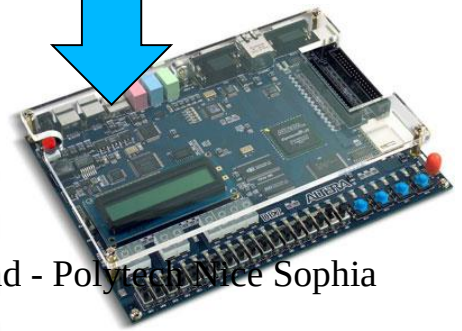
Architecture / Simulation



Simulation

Déploiement et Tests





FPGA (possible mais hors sujet)

Chaîne de compilation / simulation

1

input.c

```
Int main (){

Return 0;
}
```

gcc / scilang

Input.s

2

input.s

```
ADD r0, r1, #4
CMP r0, #4
```

Assembleur

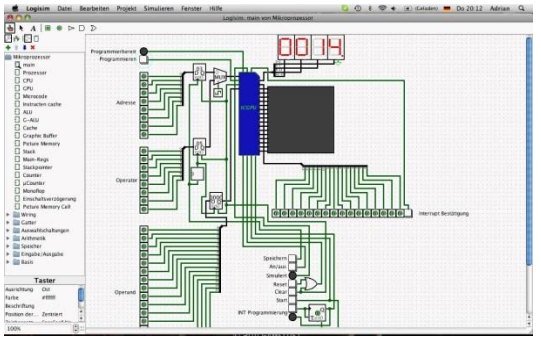
output.hex

```
v2.0 raw
b088 230a 9300
2300 9304 de07
```

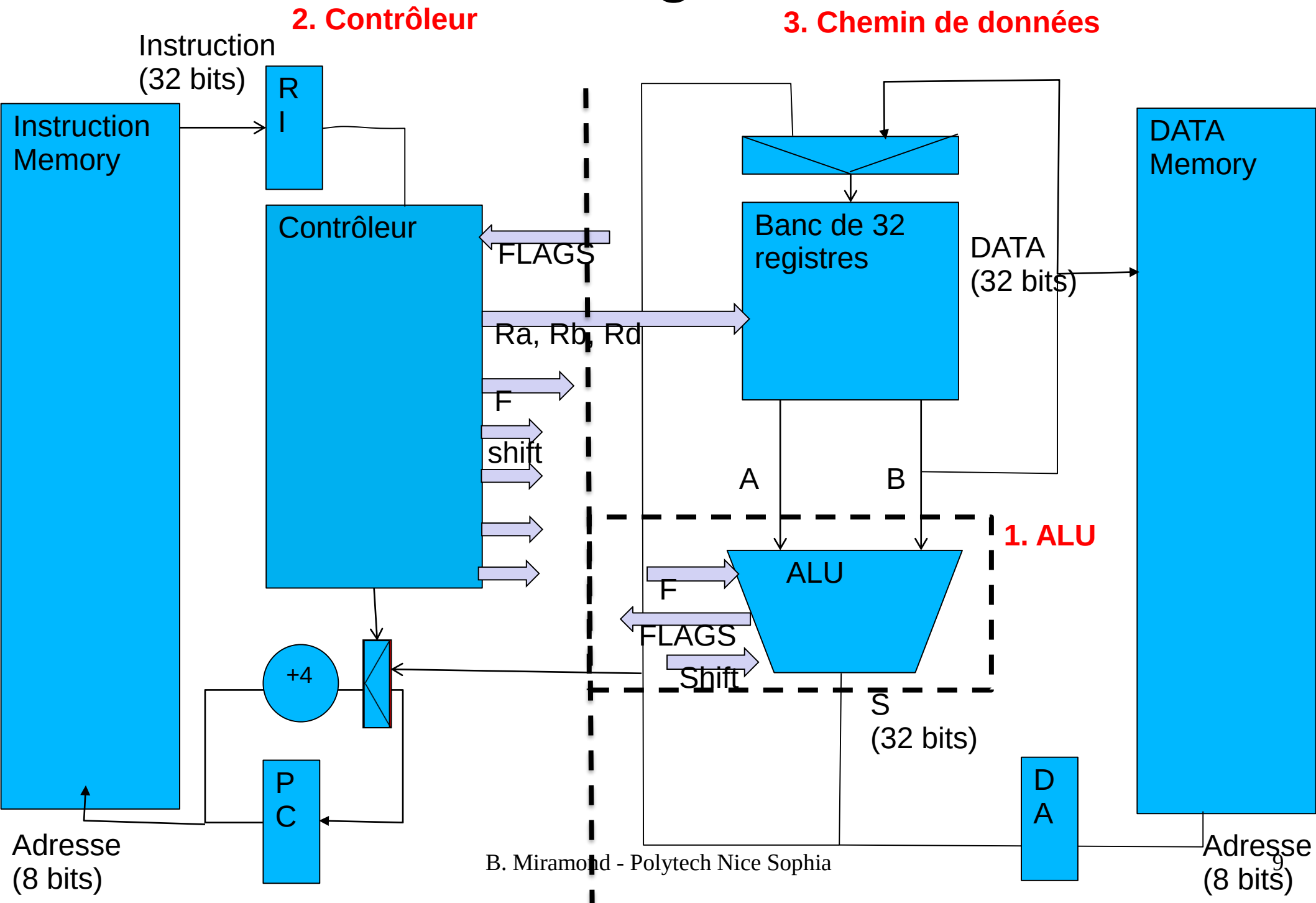
3

vector.txt

output.hex



Architecture générale



4 Types d'instructions

a) Shift, add, sub, mov,
   - 7 instructions

b) Data Processing,
   - 16 instructions

c) Load/Store,
   - 2 instructions

d) 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	*Load/store single data item* on page A5-131
011xxx 100xxx	
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
00 XX XX	Shift, add, sub…	1			
01 00 00	Data processing		1		
01 10 XX	Load/Store			1	
11 01 XX	Branch				1

Type a) Shift, add, sub, mov

Page 128 de la doc ARM,

page 28 de la doc du projet PARM

The encoding of Shift (immediate), add, subtract, move, and compare instructions is:

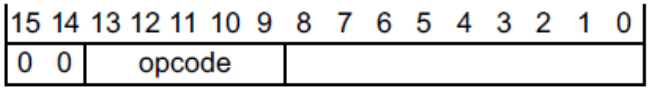


Table A5-2 16-bit shift (immediate), add, subtract, move and compare encoding

opcode	Instruction	See
000xx	Logical Shift Left[a]	*LSL (immediate)* on page A7-298
001xx	Logical Shift Right	*LSR (immediate)* on page A7-302
010xx	Arithmetic Shift Right	*ASR (immediate)* on page A7-203
01100	Add register	*ADD (register)* on page A7-191
01101	Subtract register	*SUB (register)* on page A7-450
01110	Add 3-bit immediate	*ADD (immediate)* on page A7-189
01111	Subtract 3-bit immediate	*SUB (immediate)* on page A7-448
100xx	Move	*MOV (immediate)* on page A7-312
101xx	Compare	*CMP (immediate)* on page A7-229
110xx	Add 8-bit immediate	*ADD (immediate)* on page A7-189
111xx	Subtract 8-bit immediate	*SUB (immediate)* on page A7-448

Type b) Data Processing

Page 129 de la doc ARM,

page 31 de la doc du projet PARM

The encoding of data processing instructions is:

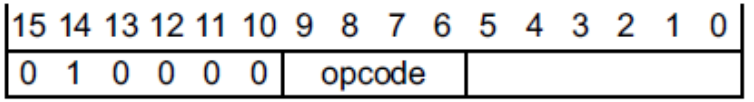


Table A5-3 16-bit data processing instructions

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

Type c) Load/Store

Page 131 de la doc ARM,

page 36 de la doc du projet PARM

The encoding of Load/store single data item instructions is:

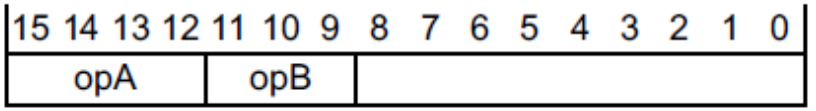


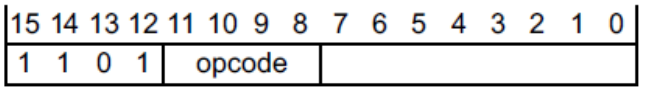
Table A5-5 16-bit Load/store instructions

opA	opB	Instruction	See
0101	000	Store Register	*STR (register)* on page A7-428
0101	001	Store Register Halfword	*STRH (register)* on page A7-444
0101	010	Store Register Byte	*STRB (register)* on page A7-432
0101	011	Load Register Signed Byte	*LDRSB (register)* on page A7-286
0101	100	Load Register	*LDR (register)* on page A7-256
0101	101	Load Register Halfword	*LDRH (register)* on page A7-278
0101	110	Load Register Byte	*LDRB (register)* on page A7-262
0101	111	Load Register Signed Halfword	*LDRSH (register)* on page A7-294
0110	0xx	Store Register	*STR (immediate)* on page A7-426
0110	1xx	Load Register	*LDR (immediate)* on page A7-252
0111	0xx	Store Register Byte	*STRB (immediate)* on page A7-430
0111	1xx	Load Register Byte	*LDRB (immediate)* on page A7-258
1000	0xx	Store Register Halfword	*STRH (immediate)* on page A7-442
1000	1xx	Load Register Halfword	*LDRH (immediate)* on page A7-274
1001	0xx	Store Register SP relative	*STR (immediate)* on page A7-426
1001	1xx	Load Register SP relative	*LDR (immediate)* on page A7-252

Type d) Branch

Page 128 de la doc ARM,

The encoding of conditional branch and supervisor call instructions is:



page 38 de la doc du projet PARM

Table A5-8 Branch and supervisor call instructions

opcode	Instruction	See
not 111x	Conditional branch	*B* on page A7-207
1110	Permanently UNDEFINED	*UDF* on page A7-471
1111	Supervisor call	*SVC* on page A7-455

e) Miscellaneous (Stack management)

Nécessaire uniquement si vous compilez votre code depuis le C plutôt que depuis l'assembleur.

Page 132 de la doc ARM,

page 37 de la doc du projet PARM

The encoding of miscellaneous 16-bit instructions is:

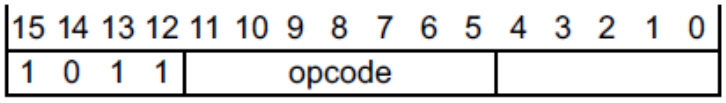
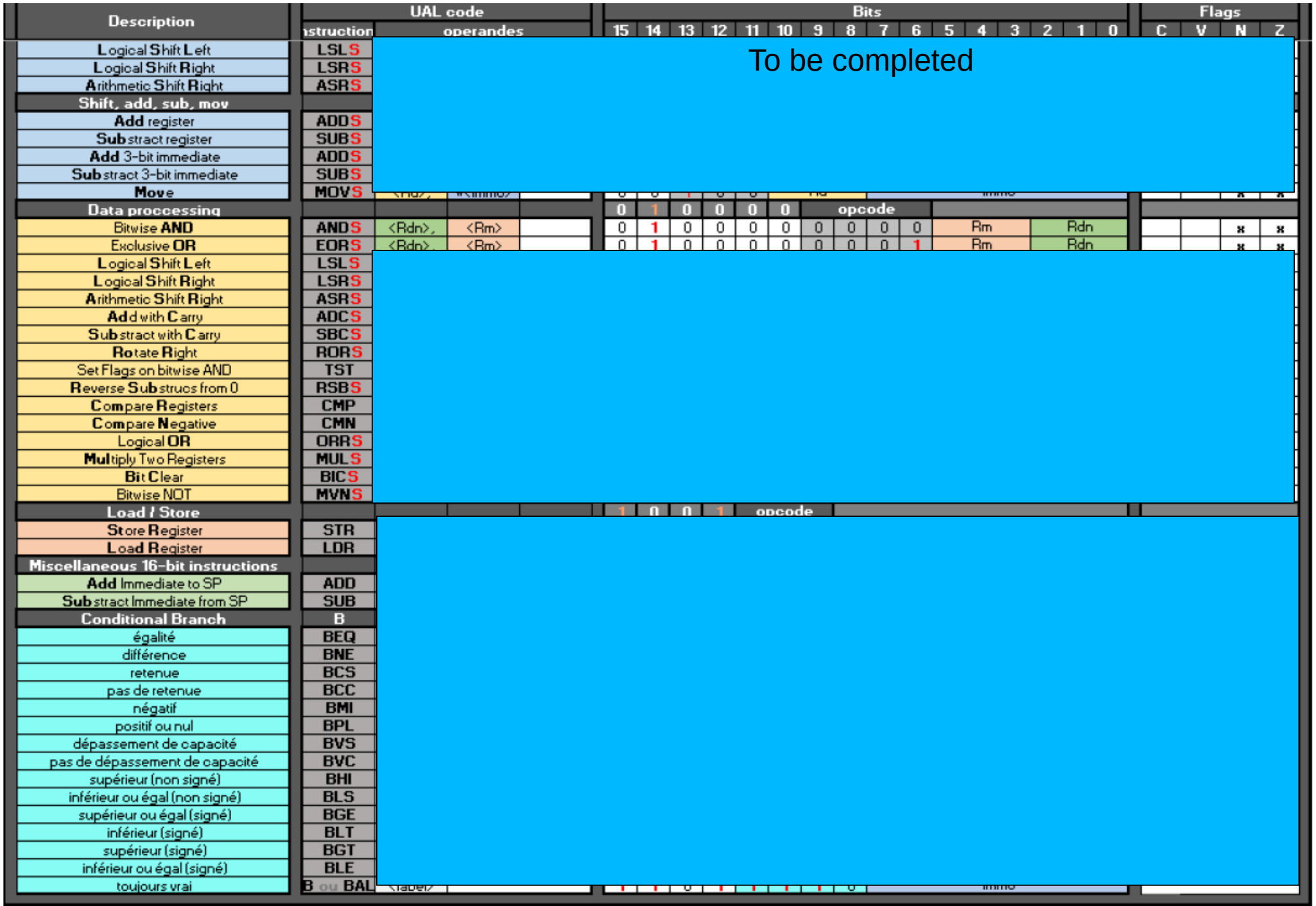


Table A5-6 Miscellaneous 16-bit instructions

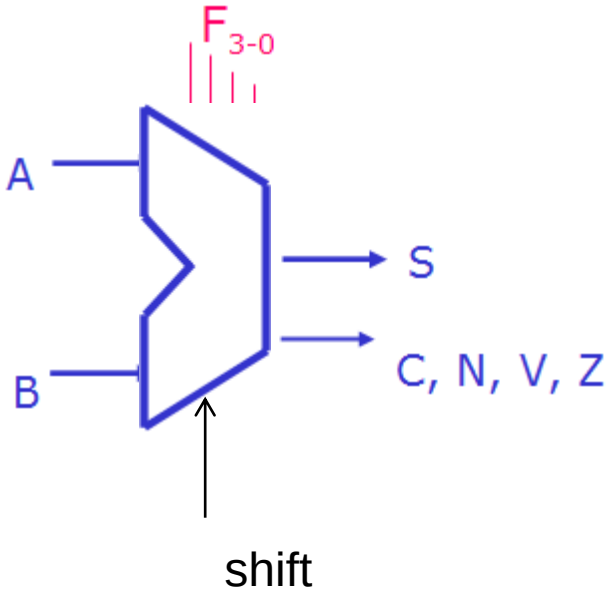
opcode	Instruction	See
0110011	Change Processor State	*CPS* on page B5-731
00000xx	Add Immediate to SP	*ADD (SP plus immediate)* on page A7-193
00001xx	Subtract Immediate from SP	*SUB (SP minus immediate)* on page A7-452
0001xxx	Compare and Branch on Zero	*CBNZ, CBZ* on page A7-219
001000x	Signed Extend Halfword	*SXTH* on page A7-461
001001x	Signed Extend Byte	*SXTB* on page A7-459
001010x	Unsigned Extend Halfword	*UXTH* on page A7-500
001011x	Unsigned Extend Byte	*UXTB* on page A7-498
0011xxx	Compare and Branch on Zero	*CBNZ, CBZ* on page A7-219
010xxxx	Push Multiple Registers	*PUSH* on page A7-350
1001xxx	Compare and Branch on Nonzero	*CBNZ, CBZ* on page A7-219
101000x	Byte-Reverse Word	*REV* on page A7-363
101001x	Byte-Reverse Packed Halfword	*REV16* on page A7-364
101011x	Byte-Reverse Signed Halfword	*REVSH* on page A7-365
1011xxx	Compare and Branch on Nonzero	*CBNZ, CBZ* on page A7-219
110xxxx	Pop Multiple Registers	*POP* on page A7-348
1110xxx	Breakpoint	*BKPT* on page A7-215
1111xxx	If-Then, and hints	*If-Then, and hints* on page A5-133

Jeu d'instructions PARM



Code de commande de l'ALU

F3 F2 F1 F0	Opération	Instructions
0 0 0 0	A and B	AND
0 0 0 1	A xor B	EOR
0 0 1 0	B << shift	LSL
0 0 1 1	B >> shift	LSR
0 1 0 0	B >> shift (arith)	ASR
0 1 0 1	A + B + Cin	ADC
0 1 1 0	A – B + Cin – 1	SBC
0 1 1 1	B >> shift (rot)	ROR
1 0 0 0	A and B	TST
1 0 0 1	0 – B	RSB
1 0 1 0	A – B	CMP
1 0 1 1	A + B	CMN
1 1 0 0	A or B	ORR
1 1 0 1	A * B	MUL
1 1 1 0	A and not B	BIC
1 1 1 1	Not B	MVN



Codes opérations de l'ALU

Sur immédiat principalement :

opcode	Instruction	See
000xx	Logical Shift Left[a]	*LSL (immediate)* on page A7-298
001xx	Logical Shift Right	*LSR (immediate)* on page A7-302
010xx	Arithmetic Shift Right	*ASR (immediate)* on page A7-203
01100	Add register	*ADD (register)* on page A7-191
01101	Subtract register	*SUB (register)* on page A7-450
01110	Add 3-bit immediate	*ADD (immediate)* on page A7-189
01111	Subtract 3-bit immediate	*SUB (immediate)* on page A7-448
100xx	Move	*MOV (immediate)* on page A7-312
101xx	Compare	*CMP (immediate)* on page A7-229
110xx	Add 8-bit immediate	*ADD (immediate)* on page A7-189
111xx	Subtract 8-bit immediate	*SUB (immediate)* on page A7-448

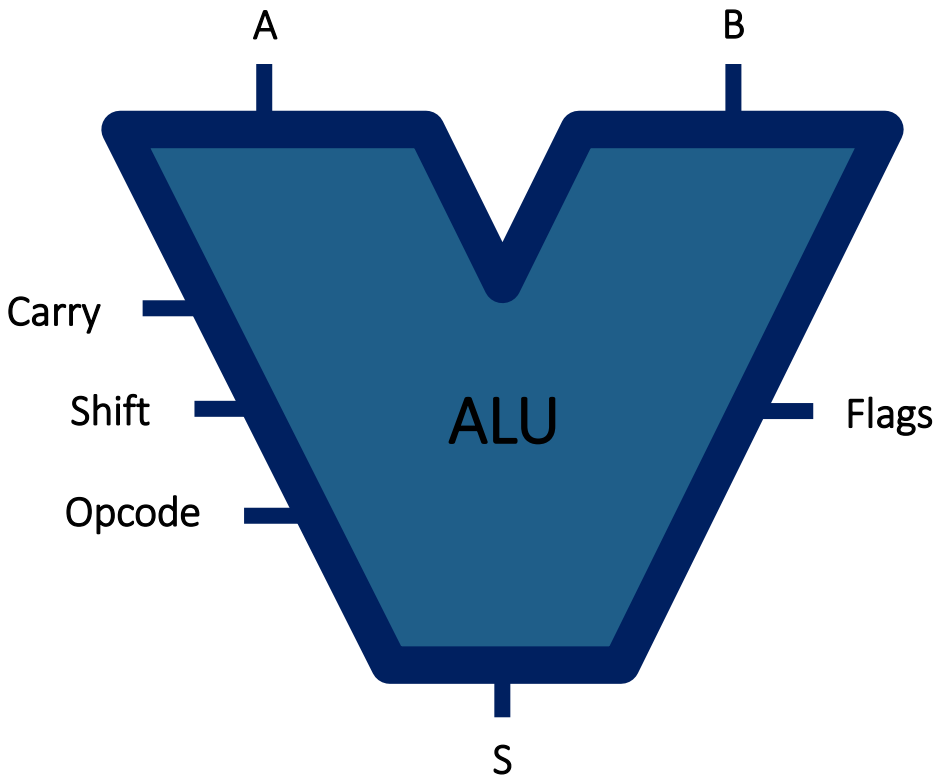
Sur Registres principalement :

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

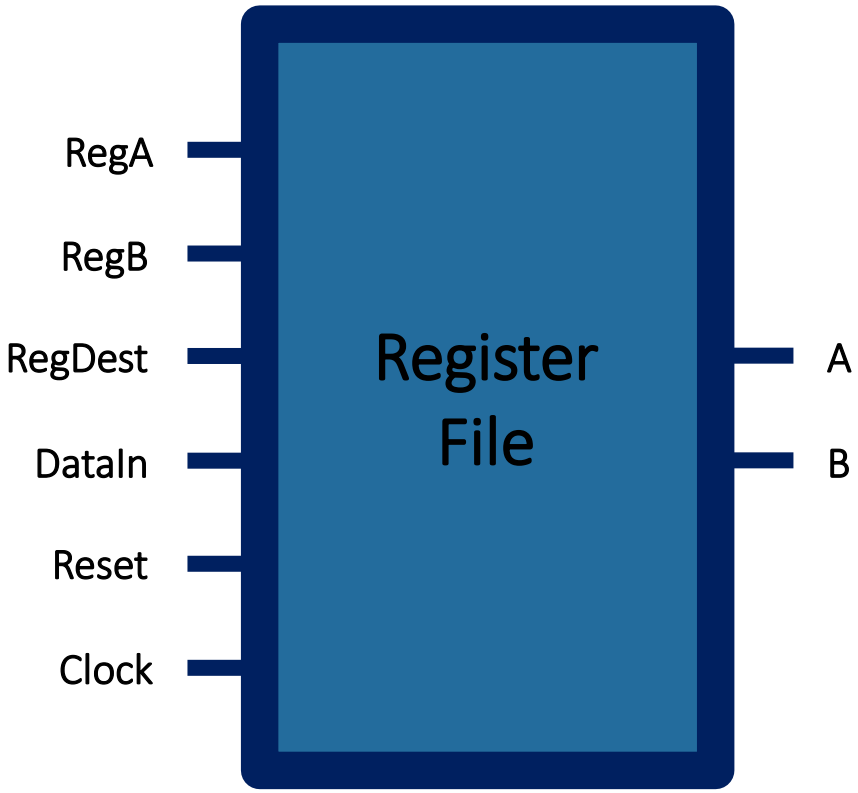
Comparaison des instructions travaillant sur registre et sur immédiat

Opcod	Inst	Opcod	Inst
000xx	LSL I	0000	AND
001xx	LSR I	0001	EOR
010xx	ASR I	0010	LSL
01100	ADD R	0011	LSR
01101	SUB R	0100	ASR
01110	ADD I3	0101	ADC
01111	SUB I3	0110	SBC
100xx	MOV I	0111	ROR
101xx	CMP I	1000	TST
110xx	ADD I8	1001	RSB
111xx	SUB I8	1010	CMP
		1011	CMN
		1100	ORR
		1101	MUL
		1110	BIC
		1111	MVN

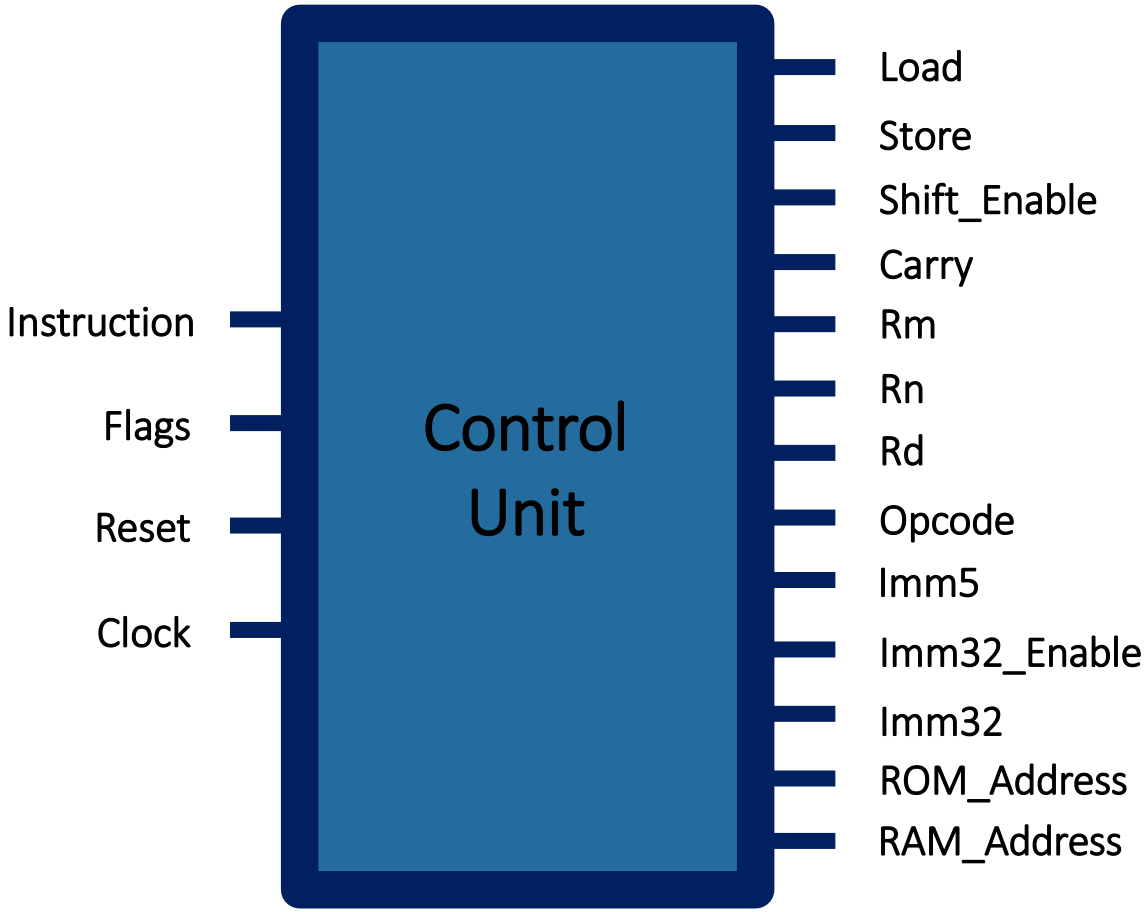
ARITHMETIC LOGIC UNIT



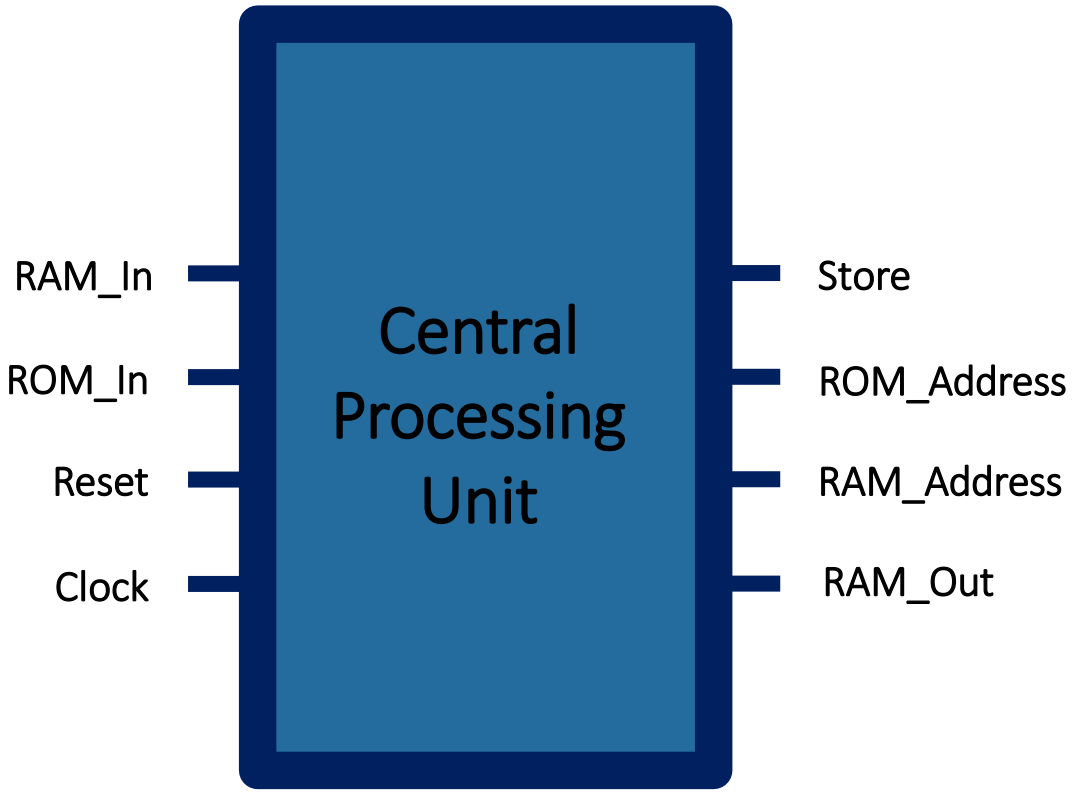
REGISTER FILE



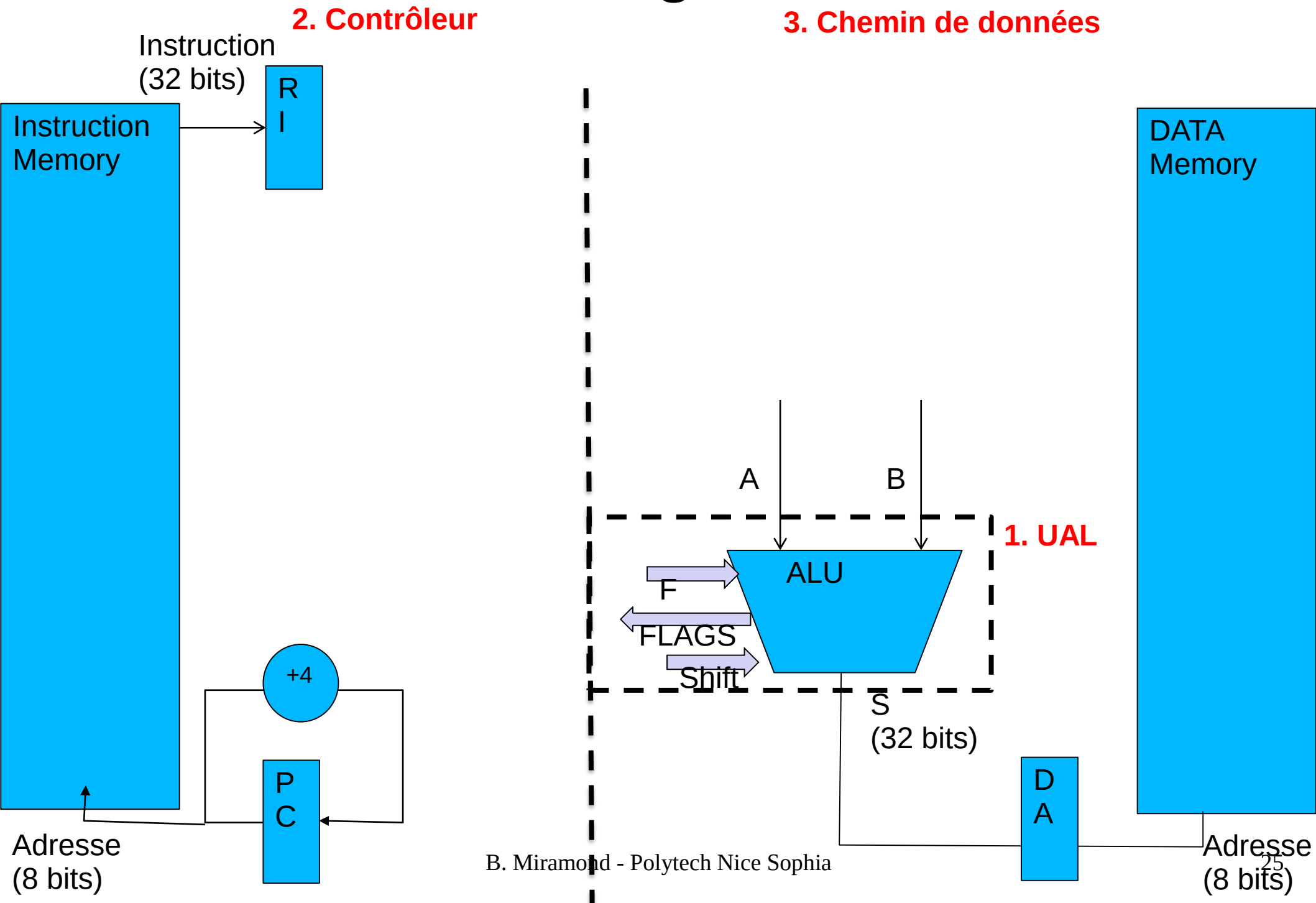
CONTROL UNIT



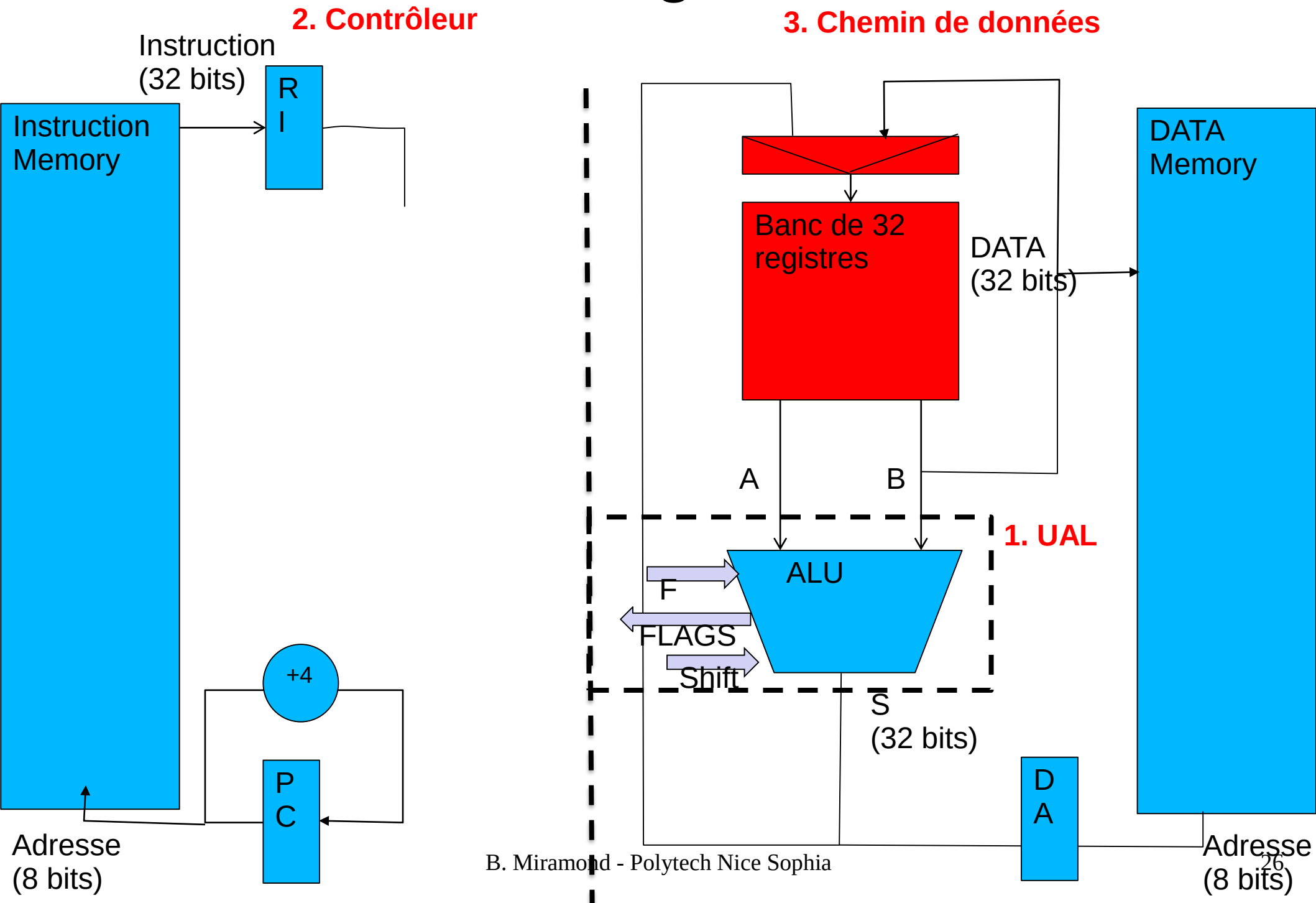
Central Processing Unit



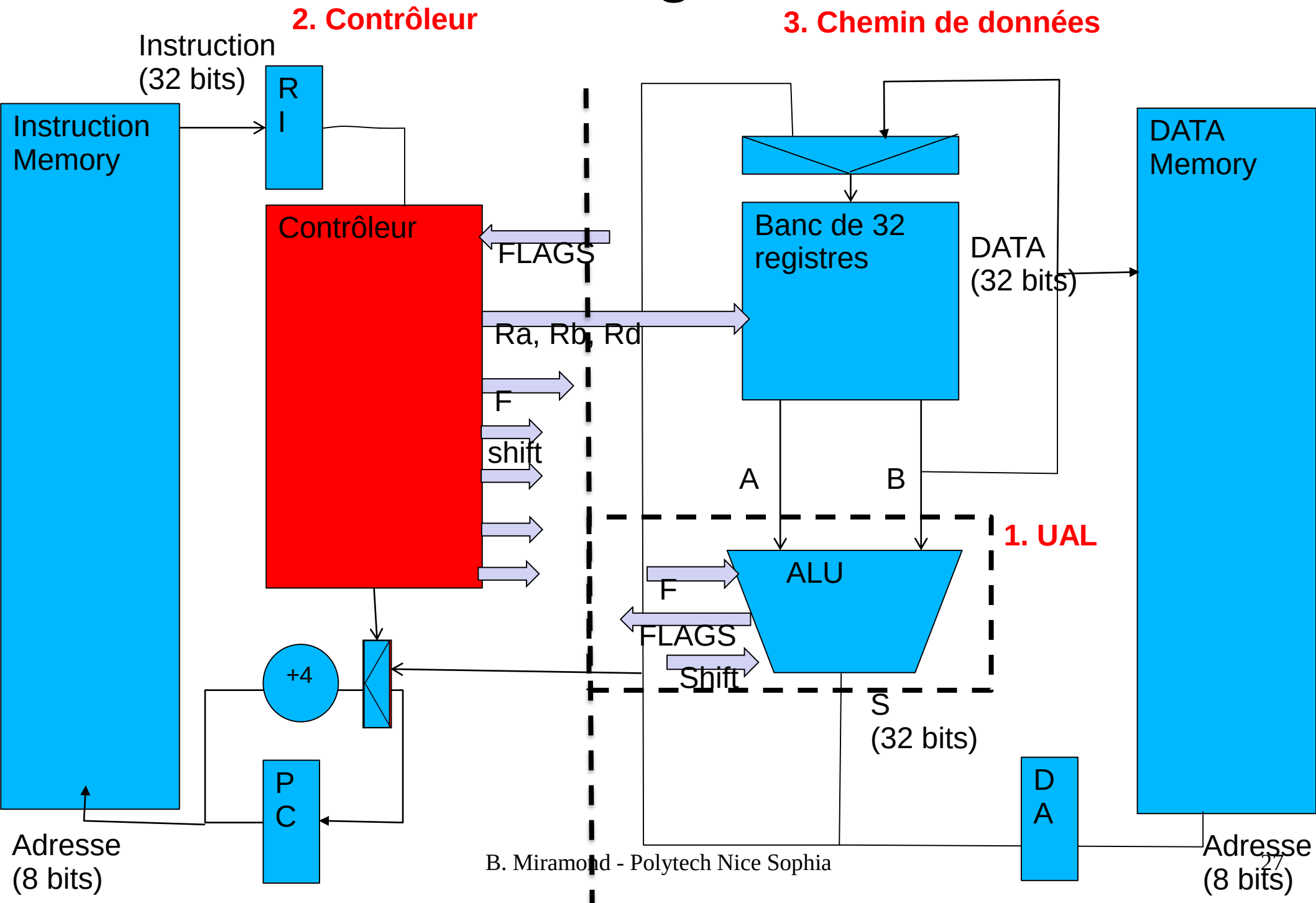
Architecture générale



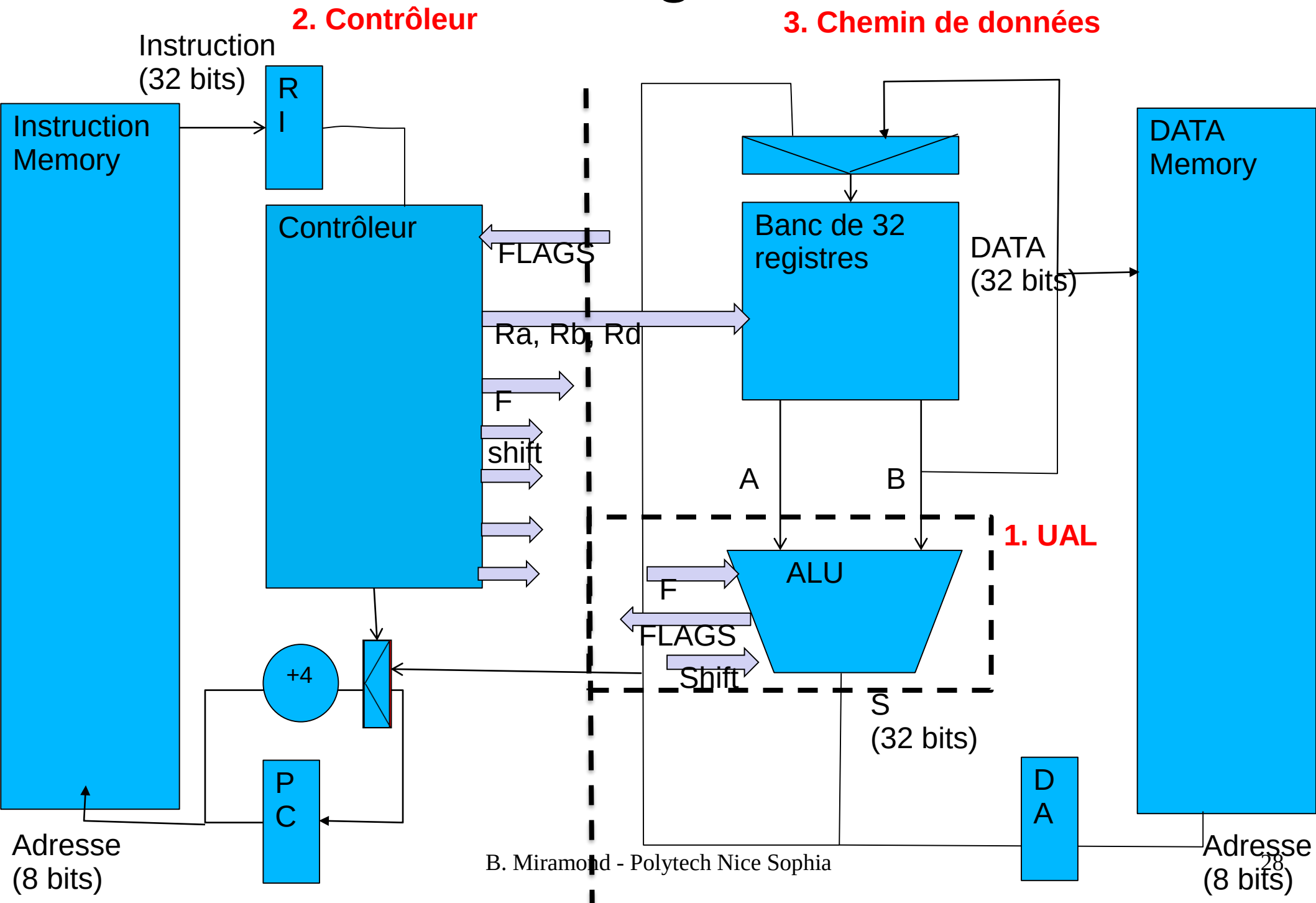
Architecture générale



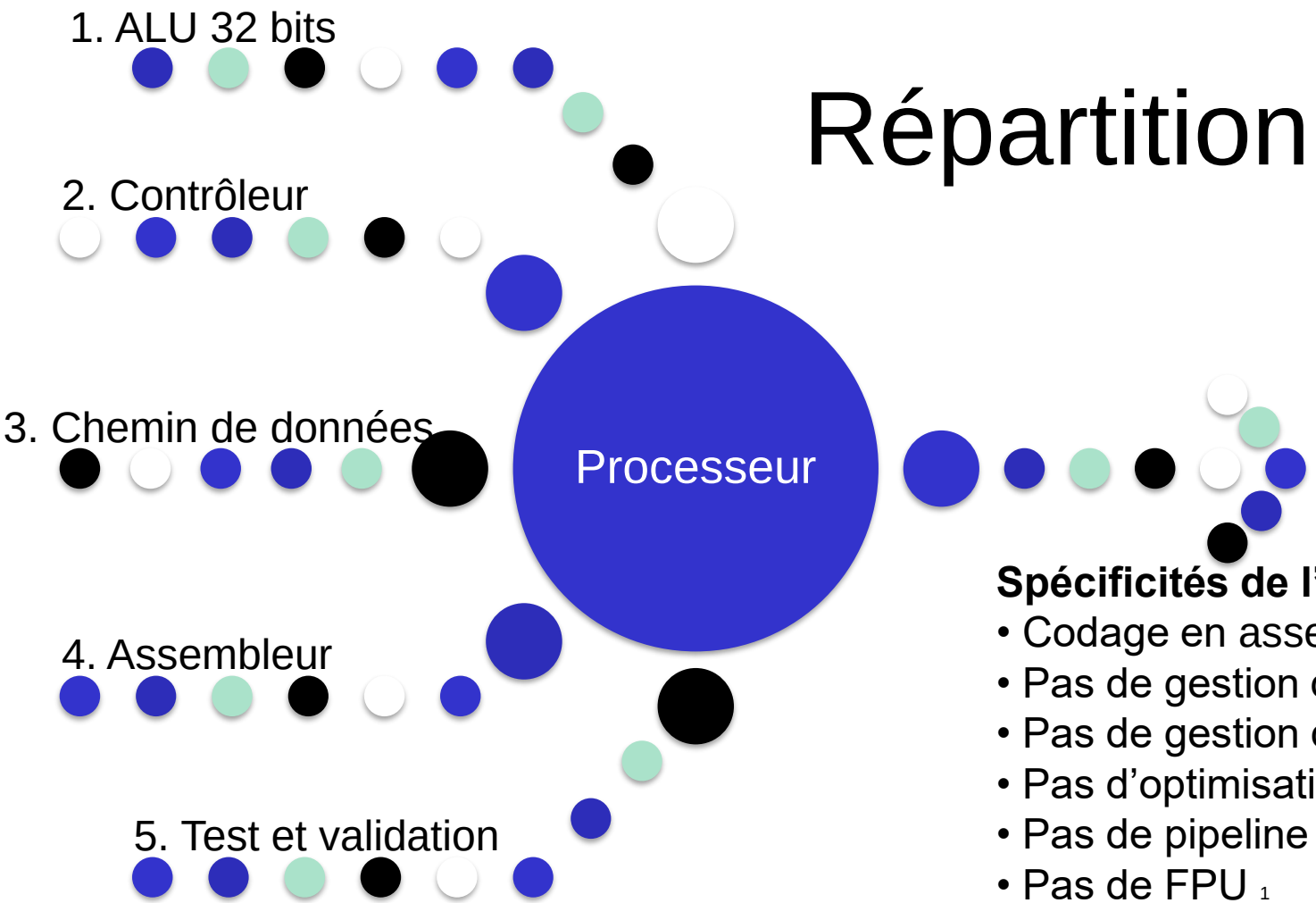
Architecture générale



Architecture générale



Répartition des tâches



Validation sur circuit FPGA

Spécificités de l'implémentation

- Codage en assembleur et/ou C
- Pas de gestion des interruptions
- Pas de gestion des appels de fonctions
- Pas d'optimisation
- Pas de pipeline
- Pas de FPU [1]
- Pas de MMU [2]
- Toutes les instructions s'exécutent en 1 cycle (excepté les instructions LDR/STR en 2 cycles)
- Instructions sur 16 bits
- Données sur 32 bits
- Adressage RAM/ROM sur 8 bits
- Adressage RAM uniquement sur la pile (en utilisant le *Stack Pointer*)

Rôles de chaque partenaire

1. ALU (Hardware)	2. Contrôleur (Hardware)	3. Chemin de données (Hardware)	4. Assembleur (Software)	Tests et Validation
Réaliser les blocs d'opérateurs arithmétiques et logiques	Lecture instruction depuis la mémoire de programme	Mouvement de registre à registre	Parser un fichier assembleur	Intégration de tous les composants
Générer les Flags	Décoder les instructions et générer les signaux de commande du chemin de données	Lecture mémoire de données Ecriture en mémoire de données	Générer le fichier binaire à charger dans la mémoire d'instruction de logisim	Validation par simulation manuelle
	Calcul d'adresse de la prochaine instruction	Envoie d'adresse pour les lectures / écritures	Générer le fichier binaire à charger dans la mémoire de données de logisim	Puis tests automatiques par vecteurs de tests

Organisation du travail
(Proposition à adapter)

7 séances dédiées :

1) Introduction (Décodeur 7-segments)
2) Projet 1 – (ALU)
3) Projet 2 – (ALU + Banc de registres + Control + ASM)
4) Projet 3 – (Mem + Tests + Control + ASM)
5) Projet 4 – (Tests + Control + ASM)
6) Intégration, validation HW / SW séparées
7) Intégration, validation HW + SW
8) Soutenance => Janvier

A chaque séance

Chaque groupe aura 15 -20 min d'encadrement alternativement avec l'encadrant

Pour cela, à chaque séance vous devez avoit rempli les 3 planches qui suivent en début de discussion :

- Qui fait quoi ?
- Ou en est chaque partie ?
- Questions et difficultés

1. Qui fait quoi ?

Séances	S 1	S 2	S 3	S 4	S 5	S 6	S 7
ALU		ALL					
BdR							
Mem							
Controler							
ASM							
Code C							
Tests							
Integration							
Presentation							

2. Ou en est chaque partie ?

Séances	Pas commencé	Commencé mais difficultés rencontrées	En cours	Bien avancé	Terminé
ALU					
BdR					
Mem					
Controler					
ASM					
Code C					
Tests					
Integration					
Presentation					

3. Questions & Difficultés

Etapes de travail	Insuffisant	En cours	Bien	Terminé
Compréhension générale				
Outils utilisés				
Mise en œuvre				
Jeu d’instructions				
Tests et validation				

Notation du projet
(cf. fichier sur moodle)

Notation du projet P-ARM

TOTAL de points (notation sur 22, note max : 20)	**20,00**	**7 Points**		**1 Points**		**1 Point**		**1 Point**		**7 Points**		**4 Points**		**4 Points**		**2 Points**	
Parties du projet	**Projet complet**		**Assembleur**		**ALU**		**Chemin de données**		**Contrôleur**		**Validation**		**Présentation / Soutenance**		**Bonus**		
SASM	Instructions A prise en charge	1	Langage :		Flags :				Unité fonctionnelle sur instructions A	1	Tests unitaires (-1/0/+1)	1,00	Effort pédagogique et explication du fonctionnement du processeur (-1/0/+1)	1			
DP	Instructions B prise en charge	1			Multiplication :				Unité fonctionnelle sur instructions B	1	Jeux de test utilisés (-1/0/+1)	1,00	Présentation des résultats (-1/0/+1)	1			
LDR	Instructions C prise en charge	1							Unité fonctionnelle sur instructions C	1	Démonstration de validation du processeur complet	1,00	Analyse et critique du travail final (-1/0/+1)	1			
B	Instructions D prise en charge	1							Unité fonctionnelle sur instructions D	1	Démonstration de validation de la chaîne assembleur-processeur	1,00	Travail collectif (-1/0/+1)	1			
SP en bonus	Instructions d'allocation de données prise en charge	1															

Si non précisé à noter sur 1 point

Parties obligatoires

-1 si non traité