

ASSEMBLEUR (ARM)

B. Miramond – Polytech Nice Sophia

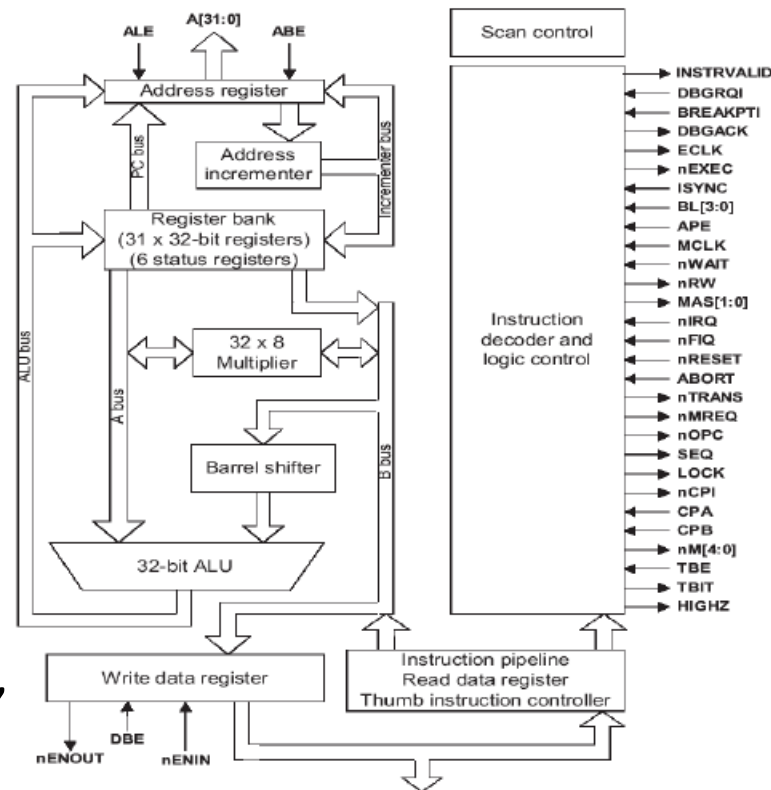
Pourquoi ce retour en arrière ?

Pourquoi s'intéresser au langage d'assemblage ?

- Savoir écrire en langage d'assemblage lorsque l'application demande d'être optimisée en performances.
- Pour le portage d'OS sur de nouvelles architectures
- Dans la plupart des ordinateurs embarqués, les systèmes disposent de quelques Mo de mémoire. Comprendre le code embarqué pour comprendre l'espace occupé.
- Comprendre comment fonctionne un compilateur.
- Comprendre la dynamique d'une architecture matérielle.

Points clefs du ARM

- Architecture load-store
- Instructions de traitement à 2 ou 3 adresses
- Instructions conditionnelles (et branchement)
- Load et store multiples
- APCS (ARM Procedure Call Standard)
- En un cycle horloge: opération ALU+shift
- Interfaçage avec coproc. simplifié (ajout de registres, de type, ...)
- Combine le meilleur du RISC et du CISC
- On le trouve dans la GBA, routeurs, Pockets PC, PDA, ...



Exemple de programme

```
.text ← directive
.globl _main

_start: ← label
        b _main
_main:
    ⇒ code op  opmrs R0,CPSR ← opérandes

#attention à la taille des immédiats !!
    ldr R1,=0xDFFFFFFF
    and R0,R0,R1
    msr CPSR_f,R0

#autre solution:pas de masquage, on force tous les bits
#
    ldrb R0,mys
    ldrb R1,re
    adds R0,R0,R1
    strb R0,de
```

```
    ldrb R0,te
    ldrb R1,boul
    adc R0,R0,R1
    strb R0,gom

    mov pc,lr

    .align
mys:    .space 1
te:    .space 1
re:    .space 1
boul:  .space 1
de:    .space 1
gom:   .space 1

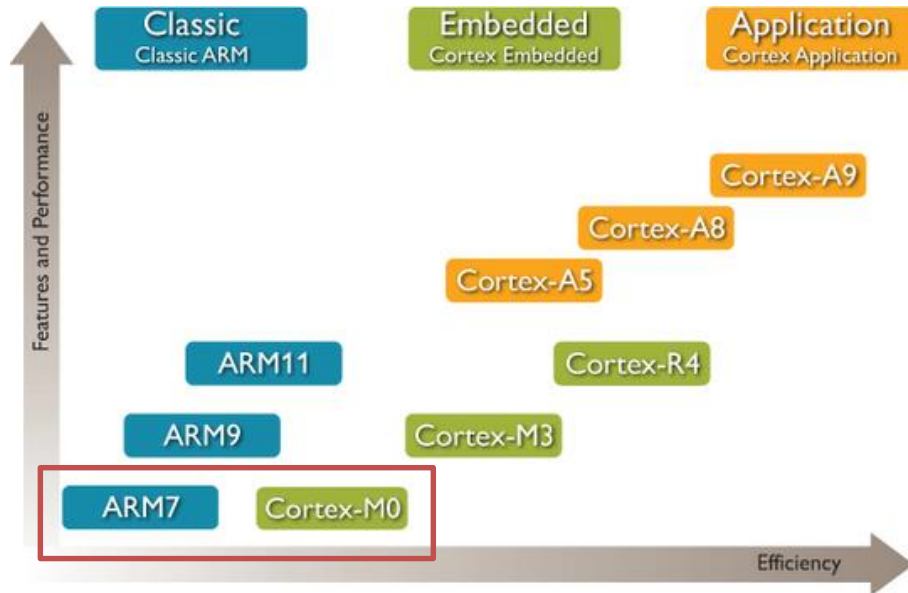
    .ltorg
    .end
```

La famille ARMv7

- ARMv7-A : Applications profile
 - Virtual address
- ARMv7-R : Real-time profile
 - Physical address
- **ARMv7-M : Micro-controler profile**
 - Size and deterministic operation

Evolution de la gamme ARM

De l'ARM7 au Cortex M0



10 Billion units shipped since its introduction in 1994

Cortex-M0 :

12.5 μ W/MHz

under 12 K gates

With just 56 instructions

low power connectivity such as Bluetooth Low Energy (BLE), IEEE 802.15 and Z-wave

Current processor	Upgrade driver	Alternative ARM processors	Benefits of upgrading
ARM7TDMI-S	Application upgrade	ARM926EJ-S, ARM968E-S, Cortex-A Series	<ul style="list-style-type: none"> •Higher performance •More features
		Cortex-R Series	<ul style="list-style-type: none"> •Better determinism for real-time processing •Higher performance •More features
	Socket upgrade	Cortex-M0	<ul style="list-style-type: none"> •1/3rd the silicon area •3x power savings •Flexible, powerful and fully deterministic interrupt handling •Higher code density •Simplified software development
		Cortex-M3	<ul style="list-style-type: none"> •Higher performance •Superior efficiency and flexibility •Flexible, powerful and fully deterministic interrupt handling •Low power modes •Higher code density •Simplified software development

Jeu d'instructions

- Le profile ARMv7-M supporte le jeu d'instructions Thumb (16 bits) et Thumb-2 (32 bits) et non le jeu d'instructions ARM classique
- La plupart des instructions 16 bits n'accèdent qu'aux registres R0 à R7 (low registers)
- Quelques instructions 16 bits accèdent aux high registers (R8-R15)
- Beaucoup d'opérations qui nécessitent plusieurs instructions 16bits s'effectuent en une seule instruction 32 bits

Caractéristiques de ARMv7-M

- Compromis entre performances, consommation et surface
 - Faible niveau de pipeline
- Opérations déterministes
 - Instructions en 1 (ou qqs) cycles
 - Faibles latence d'interruption
 - Peut fonctionner sans caches
- Compatible C/C++
 - Compatible avec les appels standards
- Conçu pour les applications embarquées
- Le cours s'appuie sur le document technique :
 - ARMv7-M Architecture Reference Manual, v2014

LE LANGAGE

Que manipule-t-on dans ce langage ?

- Des registres
- Des adresses
- Des données
- Des instructions assembleurs (32 bits)
- Des directives
- Des pseudo-instructions

Le banc de registres (mode User)

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15

16 Registres de 32 bits :

- R0-R10, R12 : généraux
- R11 (fp) Frame pointer
- R13 (sp) Stack pointer
- R14 (lr) Link register
- R15 (pc) Program counter

- Current Program Status Register

31	28	27					8	7	6	5	4	0
N	Z	C	V	Pas utilisés				IF		T	Mode	

Registre d'état APSR

N	Z	C	V	Q	...	I	F	T	M4	M3	M2	M1	M0
---	---	---	---	---	-----	---	---	---	----	----	----	----	----

N, Z, C, V, Q représentent les Flags provenant de l'ALU :

- N – Negative
- Z – Zero
- C – Carry
- V - oVerflow
- Q – saturation en traitement de signal

Bits de contrôle :

- I et F pour désactiver les interruptions IRQ et FIQ
- T indique le jeu d'instructions utilisé (0, ARM) et (1, Thumb)
- M – Mode [4..0] détermine le mode de fonctionnement du processeur

M[4..0]	Mode
10000	User
10001	Fast Interrupt (FIQ)
10010	Interrupt (IRQ)
10011	Supervisor
10111	Abort
11011	Undefined
11111	System

Organisation mémoire

- Le processeur dispose d'un bus de 32 bits d'adresses
- Soit 2^{32} adresses d'octets (8 bits)
- Ce qui correspond à 2^{30} adresses alignées sur mots (32 bits), soit multiples de 4
- Les calculs d'adresses sont réalisés comme des calculs sur entiers, par un additionneur spécifique
- Le calcul d'adresses de la prochaine instruction
 - En mode séquentiel
 - PC + 4 pour des instructions 32 bits (ARM, Thumb2)
 - PC + 2 pour des instructions 16 bits (Thumb1)
 - PC + Offset pour un saut ou un branchement (Offset >0 ou <0)

Organisation mémoire

Mots alignés

bit31		bit0	
23	22	21	20
19	18	17	16
word16			
15	14	13	12
Half-word14		Half-word12	
11	10	9	8
word8			
7	6	5	4
Byte6		Half-word14	
3	2	1	0
byte3	byte2	byte1	byte0

adresses

Deux modes de gestions des adresses non-alignées

- Adresses non alignées autorisées
- Génération d'une faute => Registre ASPR

Dans logisim, par défaut l'adressage se fait par mot complet. L'adressage par octets se fait par l'ajout de signaux byte enable (optionnel)

Structure d'un programme ASM

- A) Des directives de compilation
 - A1. Allocation de variables
 - A2. Réservation, initialisation mémoire
 - A3. De segmentation de la mémoire
- B) Des instructions à assembler en langage machine, éventuellement préfixées d'étiquettes
- C) Des directives de compilation
 - Allocation de textes en mémoire (affichage : printf)
- D) Des appels de procédure
 - Branchements
 - Gestion de la pile

Pour le projet seules les points A1 et B seront traités !

A) Directives de compilation

- Préfixées par une étiquette représentant le symbole (nom de variable)
- .int/.word permet d'allouer un ou plusieurs mots de 32 bits en mémoire
- .halfword idem pour 16bits
- .byte idem pour 8 bits
 - {<label>} .int <expression> {, <expression>...}
- .fill permet d'effectuer une réservation mémoire avec initialisation éventuelle
 - <label> .fill <repeat>{, <size>{, <value>}}
- .EQU permet d'associer une expression à un symbole
 - {<label>} .equ <expression>

Les directives principales

- .byte: déclare une variable type octet et initialisation
var1: .byte 0x11
var2: .byte 0x22,0x33
- .hword: déclare une variable type deux octets et initialisation
var3: .hword 0x1122
var4: .hword 0x44
- .word: déclare une variable type word (32bits) et initialisation
var5: .word 0x11223344

Les directives principales

- `.equ`: associe une valeur à un symbol
`.equ dix, 5+5`
`mov R3,#dix`
- `.global` (ou `.globl`): l'étiquette (symbol) est visible globalement
`.global _start` **`_start` est reconnue par le linker GNU**
`_start` doit apparaître le code principal
- `.text`, `.data`, `.bss` : début de section text, data, bss (pour le linker pour générer l'ELF)
- `.end` : fin du code source
- `.ltorg` : insère « ici » les constantes temporaires pour **LDR =**
`ldr R1,=0x11111111`
`.ltorg`

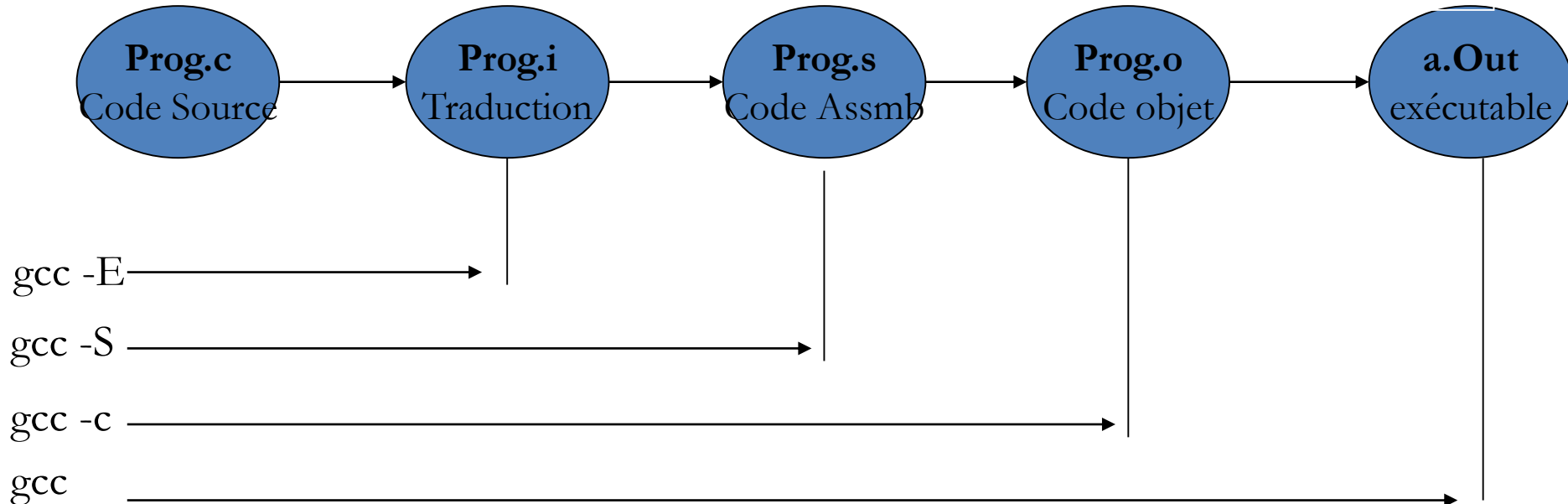
Etapes du compilateur GNU

Preprocesseur

Compilation

Assembleur

Edition de lien



Format des modules objets (.o)

- De manière à pouvoir utiliser des compilateurs, assembleurs et éditeurs de liens provenant de vendeurs différents (interopérabilité), 2 formats de fichiers objets (sections) ont été standardisés
 - COFF (Common Object File Format)
 - ELF (Executable and Linker Format)

Structure d'un module objet ELF

www.x86.org/ftp/manuals/tools/eld.pdf

- En-tête
 - Nom de fichier, Taille, adresse de début
- Espace objet (divisé en sections)
 - Code binaire
 - Zone de données
- Table des symboles
 - Symboles utilisables et à satisfaire
- Informations complémentaires
 - Auteurs, outils utilisé, versions, environnement...

Types de contenu

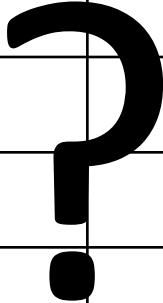
- Le compilateur organise le programme par types de contenus appelés *sections* :
- `.text/.code` = Instructions binaires
- `.data` = Données binaires initialisées
- `.bss` = (Block Started by Symbol) Données globales non initialisées
- `.rodata` = Read Only Data (Chaîne de caractères)
- `.comment` = commentaires
- `.symtab` = table des symboles
- `.rel` = table de résolution
- ...
- Le Standard ELF permet de définir autant de sections qu'on le souhaite avec n'importe quel nom
- Outils pour lire les sections : **objdump** (tous fichiers binaires) et **readelf** (ELF seulement)

Types de contenu affichés par la commande **'nm'**

- B – dans la zone .bss
- D – dans la zone .data
- C – non initialisé
- T – dans la zone .text
- U – Undefined symbol

Rangement des variables

			.data	.bss	.rodata	Pile
Globale	static	initialisée				
		non init.				
	dyna	init.				
		non init				
Locale	static	init				
		non init				
	dyna	init				
		non init				
G/L	const					



Rangement des variables

			.data	.bss	.rodata	Pile
Globale	static	initialisée				
		non init.				
	dyna	init.				
		non init				
Locale	static	init				
		non init				
	dyna	init				
		non init				
G/L	const					

B) Instructions

On les rassemble en 3 groupes:

- I. Instructions de contrôles
- II. Mouvements de données
- III. Opérations de traitements (arith. & log.)

Inst./struct. de contrôles

- Elles déroutent le cours normal du programme
- Basées sur le couple (CMP, B{cond})
- Tests complexes (CMP{cond}) (voir TD)
- Elles cassent le pipeline
- Elles servent à réaliser
 - Les alternatives (if, case)
 - Les itérations (for, while, repeat)
 - Les procédures (BL) \Rightarrow normalisation

Branchements

The encoding of conditional branch and supervisor call instructions is:

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

Table A5-8 Branch and supervisor call instructions

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

Encoding T1

All versions of the Thumb instruction set.

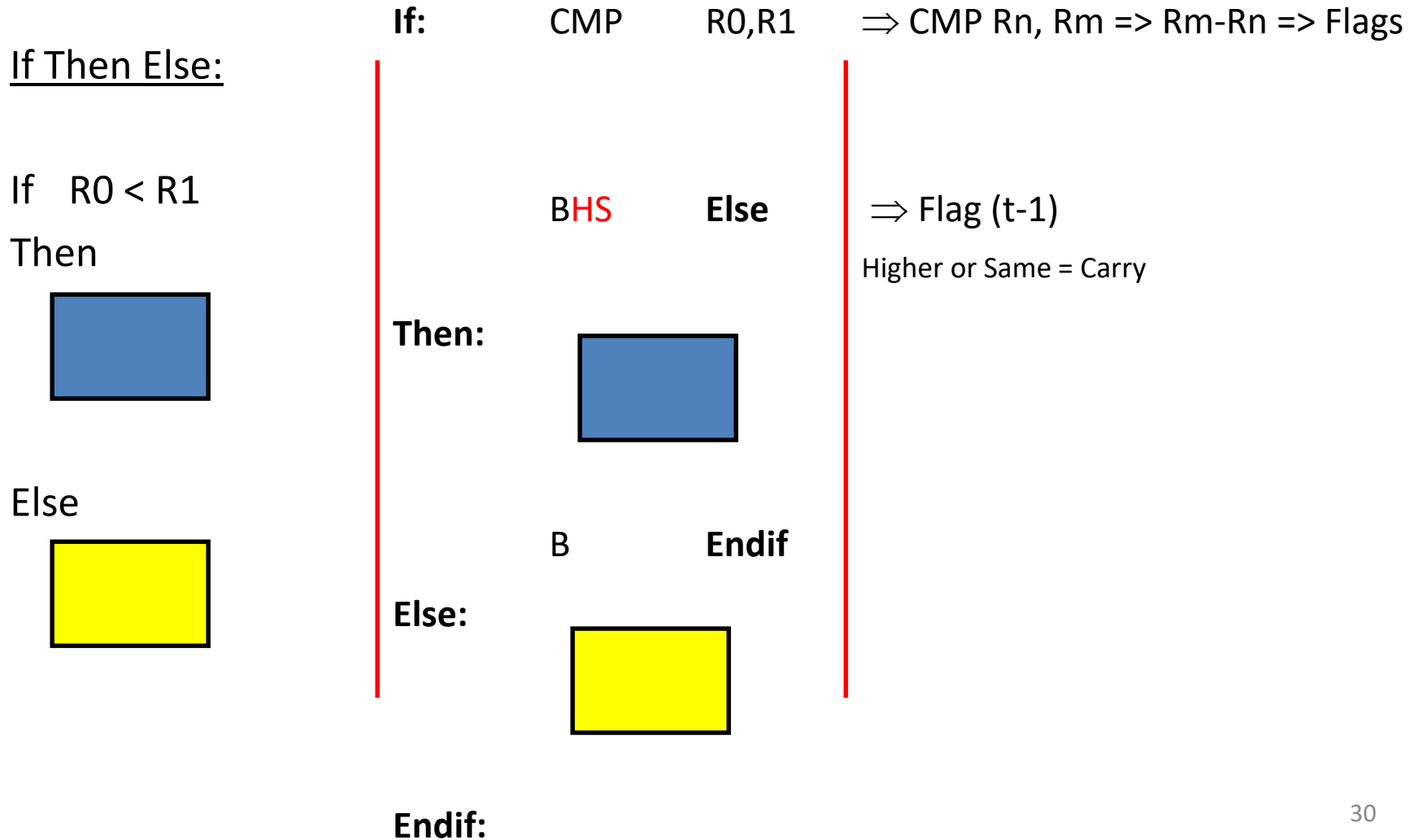
B<c> <label>

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

Cond (toutes les instructions !)

Opcode [31:28]	monic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never (do not use!)	none

Inst./struct. de contrôles



Inst./struct. de contrôles

Case Of:

Case R2 Of

-2:



-3:



Otherwise:



Case:

mDeux: CMP R2, #-2 (peut être remplacé par CMN R1, #2) !!
BNE mTrois



B Endcase

mTrois: CMP R2, #-3
BNE Otherwise



B Endcase

Otherwise:




Endcase:

Inst./struct. de contrôles

While do:

While $R3 \geq R1$
do



While:	CMP	R3,R1	⇒ CMP Rn,<Operand2>
	BLT	Endwhile	⇒ si Rn LT Operand2 en signé
#	BLO	Endwhile	⇒ si Rn LO Operand2 en nonsigné
do:			
			
	B	While	
Endwhile:			

Inst./struct. de contrôles

For:

For R3 = 1 to n fin
do



For:

LDR R3, n



SUBS R3,R3,#1

BNE **For**

Endfor:

⇒ décrémentation de 1

Donc dans tous les cas de structures de contrôles, les
branchements sont décidés en fonction des Flags calculés par
l'ALU sur l'instruction précédente !!!

II. INSTRUCTIONS DE DÉPLACEMENT DE DONNÉES

Instructions load/store (data flow)

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

These instructions have one of the following values in opA:

- 0b0101.
- 0b011x.
- 0b100x.

Encoding T1 All versions of the Thumb instruction set.
STR<c> <Rt>, [<Rn>{, #<imm5>}]

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

Encoding T2 All versions of the Thumb instruction set.
STR<c> <Rt>, [SP, #<imm8>]

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

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

Instructions (Data Flow)

Types d'adressage :

- LDR R0, Imm8 \Rightarrow direct
- LDR R0, [R1] \Rightarrow indirect
- LDR R0, [R1, #offset] \Rightarrow indirect pré-indexé
- LDR R0, [R1, #offset]! \Rightarrow indirect pré-indexé et auto-incrémenté
- LDR R0, [R1], #offset \Rightarrow indirect post-indexé et auto-incrémenté
- LDR R0, [R1, -R2]! \Rightarrow indirect pré-indexé et auto-incrémenté
- LDR R0, [R1], -R2 \Rightarrow indirect post-indexé et auto-incrémenté

Instructions auto-incrémentées

Si R1 « pointe » (ADR R1,tab) sur une zone mémoire (vecteur, matrice)

Rem: une zone mémoire = vecteur, matrice, tableau, ...

ADD R1, R1, #2 \Rightarrow R1 pointe maintenant sur le prochain half-word

LDRH R0, [R1] \Rightarrow R0 reçoit le half-word : $R0 = \text{mem}_{16}[R1]$

On peut utiliser une **pré**-indexation (R1 est modifié avant le transfert):

LDRH R0, [R1,#2] $\Rightarrow R0 = \text{mem}_{16}[R1+2]$

\Rightarrow R1 mis à jour ??? Quelle syntaxe ???

III. INSTRUCTIONS DE TRAITEMENT

Instructions de traitement

Règles pour le traitement des données:

- Les opérandes sont 32 bits, constantes ou registres
- Le résultat 32 bits dans un registre
- 3 opérandes: 2 sources et 1 destination
- En signé ou non signé
- Peut mettre à jour les Flags (S)
- Le registre destination peut être un des registres sources
- Les opérandes peuvent être
 - des registres
 - Des constantes (immédiats)

Instructions arithmétiques et logiques

The encoding of data processing instructions is:

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

Encoding T1

All versions of the Thumb instruction set.

CMP<C> <Rn>, <Rm>

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

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

a) Instructions Logiques

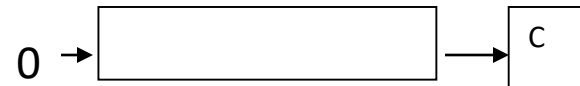
Elles opèrent bit à bit

(Elles sont très utilisées pour les E/S)

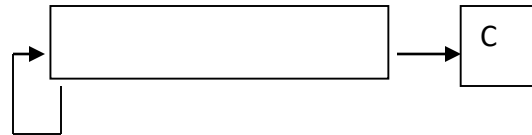
- Des instructions en soit:
 - TST, TEQ (and et eor) \Rightarrow pas de destination, CPSR mis à jour
 - AND, EOR, ORR, BIC (and not)
 \Rightarrow destination, CPSR mis à jour si {S}
- Des instructions indirectes (hors projet PARM):
 - MOV R2,R1 LSL #5
 - MOV R3,R4 LSR R6
 - \Rightarrow LSL, LSR, ASR, ROR, RRX

Instructions Logiques

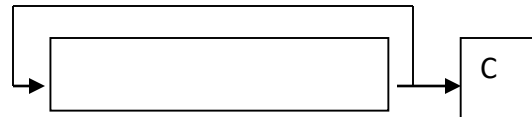
- LSR: logical shift right



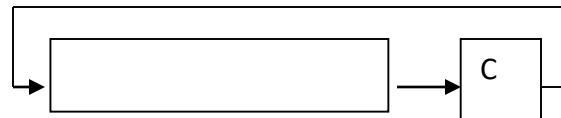
- ASR: arithmetic shift right



- ROR: rotate right



- RRX: rotate right extended



Instructions logiques

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

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

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

Catégorie A

Encoding T1 All versions of the Thumb instruction set.

LSLS <Rd>, <Rm>, #<imm5>

LSL<C> <Rd>, <Rm>, #<imm5>

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

Catégorie B

Encoding T1 All versions of the Thumb instruction set.

LSLS <Rdn>, <Rm> Outside IT block.

LSL<C> <Rdn>, <Rm> Inside IT block.

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

Les bits LSB de Rm contiennent la quantité de bits à décaler

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

b) Instructions Arithmétiques

- Elles permettent l'addition, la soustraction, la multiplication
- Elles peuvent prendre en compte le flag C (retenue) et le mettre à jour

-ADD, ADC: addition, addition plus Carry

-SUB, SBC, RSB, RSC: soustraction, soustraction –NOT(C), inversée, ...

SUBS R1,R2,R3 \Rightarrow R1=R2-R3

RSBS R1,R2,R3 \Rightarrow R1=R3-R2

-multiplications

Instructions Arithmétiques

Un exemple utilisant l'instruction MLA (Mul/Acc):

```
      MOV    R11, #20
      MOV    R10, #0
LOOP:  LDR    R0,[R8], #4
      LDR    R1,[R9], #4
      MLA    R10,R0,R1,R10  $\Rightarrow$   $R10=R10+R0*R1$ 
      SUBS   R11,R11,#1  $\Rightarrow$  SUB et CMP car {S}
      BNE    LOOP
```

\Rightarrow C'est le produit scalaire de deux vecteurs

Instructions Arithmétiques

Les flags sont mis à jour avec l'écriture {S} à la fin du mnémonique
⇒ on peut également utiliser CMP (SUBS) ou CMN (ADDS)

CMP R1,#5 ⇒ R1-5 : les flags sont mis à jour, résultat non stocké

- Utilisé avant une instruction conditionnelle (ARM)
- Utilisé avant un branchement conditionnel (B{cond} ou B{cond}L)
- Associé aux structures de contrôles

REM: TST (and) et TEQ (eor) pour comparaisons logiques

Projet PARM

- il est possible d'exécuter du code C compilé par Clang. Il doit cependant rester relativement simple.
- on évitera :
 - Les appels de fonctions (LR,PUSH,POP non implémentés)
 - Les variables globales et static (adressage uniquement sur la pile)
- On s'assurera donc :
 - D'écrire tout le code dans la fonction main()
 - De placer toutes les valeurs (y compris celles de comparaison dans les conditions) dans des variables
 - De déclarer toutes les variables dans le corps de la fonction main()

La commande à utiliser est la suivante (cf. documentation) :

```
clang -S -target arm-none-eabi -mcpu=cortex-m0 -O0 -mfloat-abi=softmain.C
```

Qui crée un fichier main.s