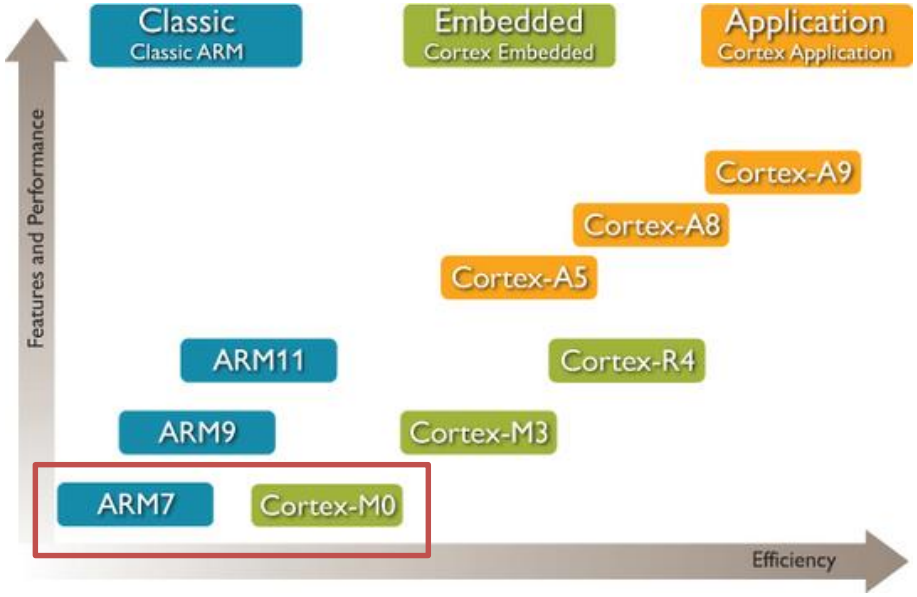
Instructions ARMv7

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

Current processor	Upgrade driver	Alternative ARM processors	Benefits of upgrading
ARM7TDMI-S	Application upgrade	ARM926EJ-S, ARM968E-S, Cortex-A Series	•Higher performance •More features
		Cortex-R Series	•Better determinism for real-time processing •Higher performance •More features
	Socket upgrade	Cortex-M0	•1/3rd the silicon area •3x power savings •Flexible, powerful and fully deterministic interrupt handling •Higher code density •Simplified software development
		Cortex-M3	•Higher performance •Superior efficiency and flexibility •Flexible, powerful and fully deterministic interrupt handling •Low power modes •Higher code density •Simplified software development

Cortex-M0 :
12.5µW/MHz
under 12 K gates
With just 56 instructions
low power connectivity such as Bluetooth Low Energy (BLE), IEEE 802.15 and Z-wave

Taille de code compact

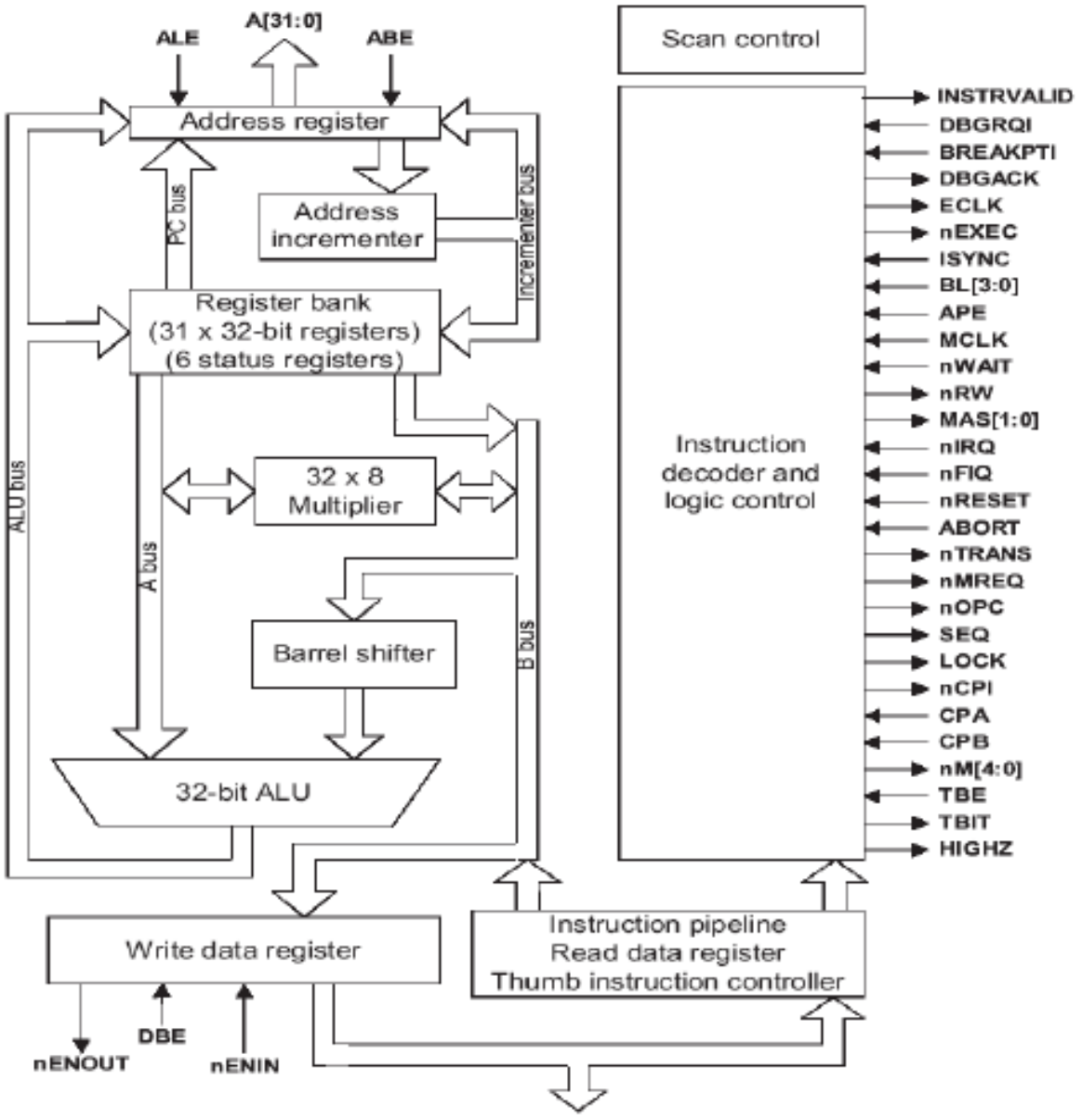
8-bit example		16-bit example	ARM Cortex-M

MOV A, XL ; 2 bytes
MOV B, YL ; 3 bytes
MUL AB; 1 byte
MOV R0, A; 1 byte
MOV R1, B; 3 bytes
MOV A, XL ; 2 bytes
MOV B, YH ; 3 bytes
MUL AB; 1 byte
ADD A, R1; 1 byte
MOV R1, A; 1 byte
MOV A, B ; 2 bytes
ADDC A, #0 ; 2 bytes
MOV R2, A; 1 byte
MOV A, XH ; 2 bytes
MOV B, YL ; 3 bytes
	MUL AB; 1 byte
	ADD A, R1; 1 byte
	MOV R1, A; 1 byte
	MOV A, B ; 2 bytes
	ADDC A, R2 ; 1 bytes
	MOV R2, A; 1 byte
	MOV A, XH ; 2 bytes
	MOV B, YH ; 3 bytes
	MUL AB; 1 byte
	ADD A, R2; 1 byte
	MOV R2, A; 1 byte
	MOV A, B ; 2 bytes
	ADDC A, #0 ; 2 bytes
	MOV R3, A; 1 byte
		MOV R4,&0130h
		MOV R5,&0138h
		MOV SumLo,R6
		MOV SumHi,R7
		(Operands are moved to and from a memory mapped hardware multiply unit)
			MULS r0,r1,r0

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


ALE
A[31:0]
ABE
Address register
Address incrementer
PC bus
Incrementer bus
Register bank
(31 x 32-bit registers)
(6 status registers)
ALU bus
A bus
B bus
32 x 8
Multiplier
Barrel shifter
32-bit ALU
Write data register
DBE
nENOUT
nENIN
Scan control
Instruction decoder and logic control
Instruction pipeline
Read data register
Thumb instruction controller
INSTRVALID
DBGRQI
BREAKPTI
DBGACK
ECLK
nEXEC
ISYNC
BL[3:0]
APE
MCLK
nWAIT
nRW
MAS[1:0]
nIRQ
nFIQ
nRESET
ABORT
nTRANS
nMREQ
nOPC
SEQ
LOCK
nCPI
CPA
CPB
nM[4:0]
TBE
TBIT
HIGHZ

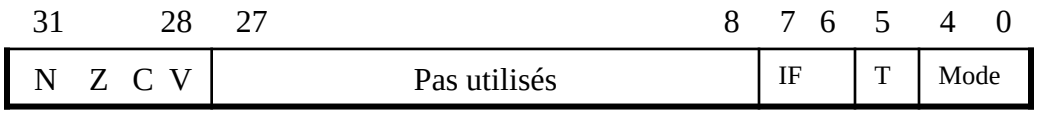
Le banc de registres (mode User)

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

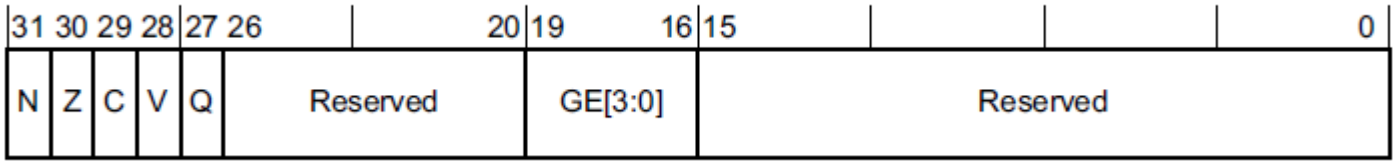
- Current Program Status Register



Registres de l'architecture ARMv7

- R15 = PC – program counter
- R14 = LR – Link register (valeur de retour après une instruction BL)
- R13 = SP – stack pointer

- R12 = [R0,R11] = General purpose registers
- APSR = Registre d'état (Application Program Status Register) : page 37 du document technique



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

Des adresses et des données

- 2^{32} bytes de mémoire maxi
- Un word = 32 bits
- Un half-word = 16 bits
- Un byte = 8 bits
- Little endian (0x01AA $\Rightarrow$ Ad+ AA 01)
- En hexa (0x0F), déc (15), bin (0b00001111)

Types de données manipulés

- Byte 8 bits
- Halfword 16 bits
- Word 32 bits

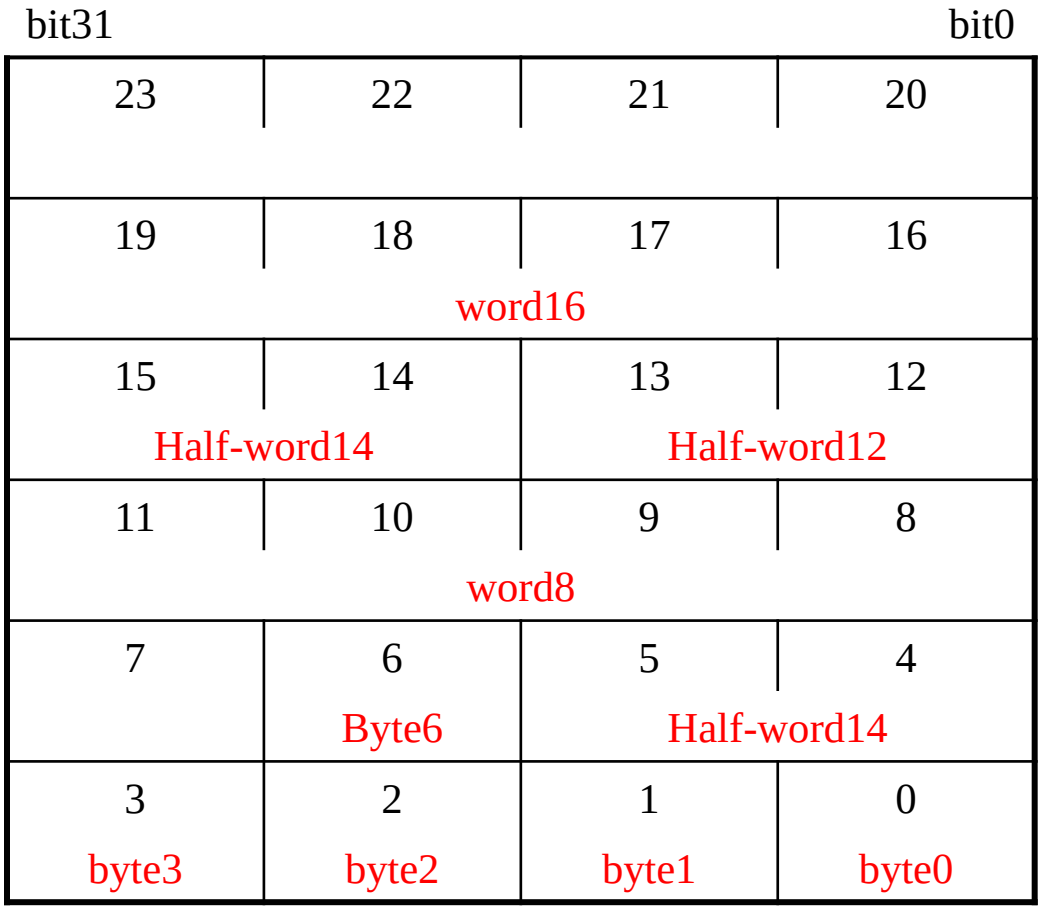
- Les registres du processeurs sont sur 32 bits

- Les instructions sont codées sur
  - 32 bits dans le jeu d'instructions ARM initial (performance)
  - 16 bits pour le jeu d'instructions Thumb (code compact)
  - 32 bits pour le jeu d'instructions Thumb2 (compact et performant)

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



Deux modes de gestions des adresses non-alignées

- Adresses non alignées autorisées
- Génération d'une faute

=> Registre ASPR

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

Codage des instructions Thumb

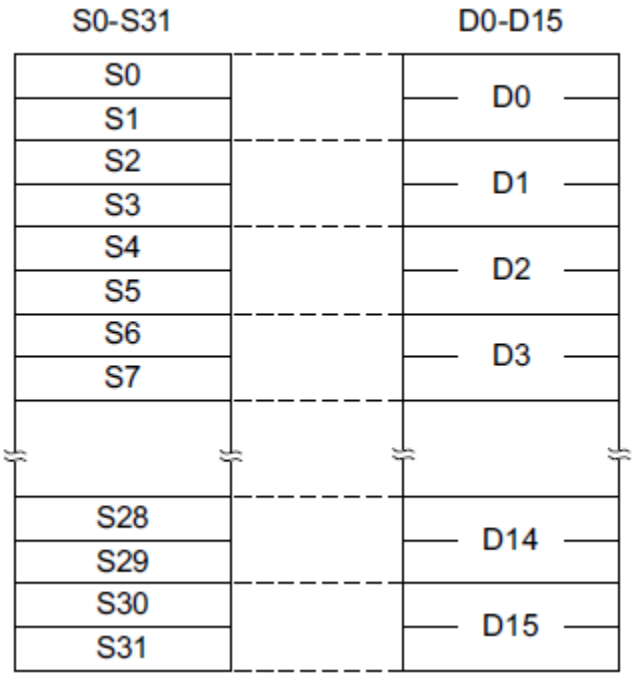
- Les instructions Thumb sont des séquences de demi-mots alignés :
  - Soit 1 instruction de 32 bits si les bits [15,11] sont
    - 0b11101.
    - 0b11110.
    - 0b11111.
  - Soit 2 instructions de 16 bits autrement

OPÉRATIONS DU PROCESSEUR

Opérations du processeur

- Le processeur ARMv7-M dispose d'une Unité Arithmétique et Logique offrant les opérations de base
- Ainsi qu'un barrel shifter permettant au préalable de réaliser un décalage
- Le processeur peut également disposer d'une FPU pour le calcul sur données flottantes (pas le cas dans ce projet)

Calcul flottant



- Le calcul flottant est assuré par l'unité de calcul associée la FPU
- La dernière version (FPv5) supporte :
  - 32 single-precision or 16 double-precision registers
  - Conversions integer, single-precision floating-point, half-precision floating-point formats, double-precision
  - Data transfers de single-precision de double-precision
- Les calculs se font soient sur
  - 32 registres 32 bits
  - 16 registres 64 bits

Codage des nombres flottants (rappel)

Page 39 du document ARM

The interpretation of the format depends on the value of the exponent field, bits[30:23]:

0 < exponent < 0xFF

The value is a *normalized number* and is equal to:

$-1^S \times 2^{(\text{exponent} - 127)} \times (1.\text{fraction})$

The minimum positive normalized number is 2^{-126}, or approximately 1.175×10^{-38}.

The maximum positive normalized number is $(2 - 2^{-23}) \times 2^{127}$, or approximately 3.403 ×1

exponent == 0

The value is either a zero or a *denormalized number*, depending on the fraction bits:

fraction == 0

The value is a zero. There are two distinct zeros:

+0 When S==0.

–0 When S==1.

These usually behave identically. In particular, the result is *equal* if +0 and –0 are compared as floating-point numbers. However, they yield different results in some circumstances. For example, the sign of the infinity produced as the result of dividing by zero depends on the sign of the zero. The two zeros can be distinguished from each other by performing an integer comparison of the two words.

fraction != 0

The value is a denormalized number and is equal to:

$-1^S \times 2^{-126} \times (0.\text{fraction})$

The minimum positive denormalized number is 2^{-149}, or approximately 1.401×10^{-45}.

Denormalized numbers are optionally flushed to zero in the FP extension. For details see *Flush-to-zero* on page A2-43.

exponent == 0xFF

The value is either an *infinity* or a *Not a Number* (NaN), depending on the fraction bits:

fraction == 0

The value is an infinity. There are two distinct infinities:

+∞ When S==0. This represents all positive numbers that are too big to be represented accurately as a normalized number.

-∞ When S==1. This represents all negative numbers with an absolute value that is too big to be represented accurately as a normalized number.

fraction != 0

The value is a NaN, and is either a *quiet NaN* or a *signaling NaN*.

In the FP architecture, the two types of NaN are distinguished on the basis of their most significant fraction bit, bit[22]:

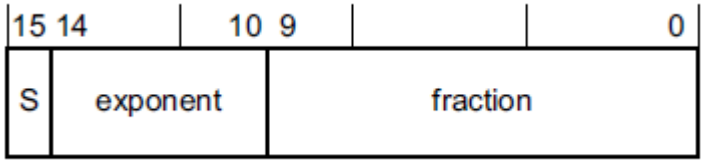
bit[22] == 0

The NaN is a signaling NaN. The sign bit can take any value, and the remaining fraction bits can take any value except all zeros.

bit[22] == 1

The NaN is a quiet NaN. The sign bit and remaining fraction bits can take any value.

ARM propose également le mode Half-precision (IEEE-754 2008) sur 16 bits :

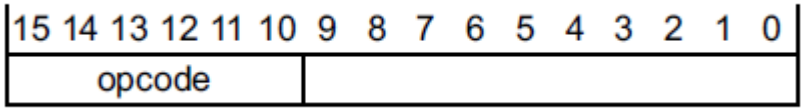


CODAGE DES INSTRUCTIONS

Codage 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**, **100xxx** ***Load/store single data***

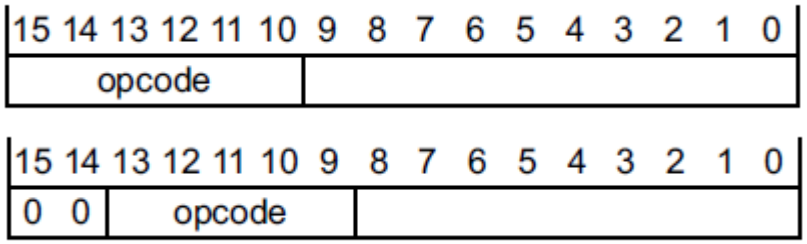


- 10100x Generate PC-relative address
- 10101x Generate SP-relative address
- 1011xx *Miscellaneous 16-bit instructions*
- 11000x Store multiple registers
- 11001x Load multiple registers
4. **1101xx** ***Conditional branch, and supervisor call***
- 11100x Unconditional Branch

1. Shift, add, substract, move and compare

Opcod 00xxxx

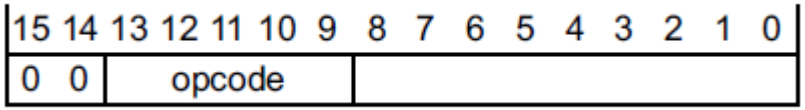
- LSL, Logical Shift Left Immediate
- LSR, Logical Shift Right Immediate
- ASR, Arithmetic Shift Right Immediate
- ADD, Add register / Immediate (3 or 8 bits)
- SUB, Sub register / Immediate (3 or 8 bits)
- MOV, Move
- CMP, Compare



Shift, add, substract, move and compare

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

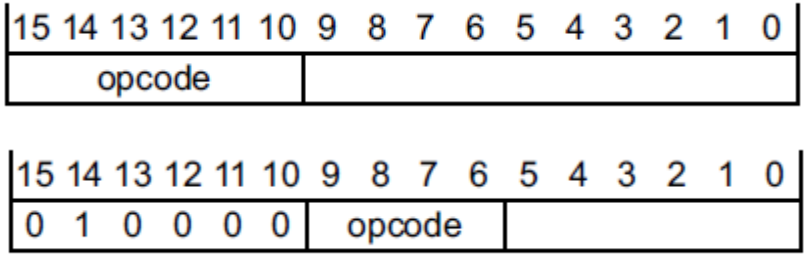
opcode	Instruction	See
000xx	Logical Shift Left[a]	*LSL (immediate)* on page A7-298
001xx	Logical Shift Right	*LSR (immediate)* on page A7-302
010xx	Arithmetic Shift Right	*ASR (immediate)* on page A7-203
01100	Add register	*ADD (register)* on page A7-191
01101	Subtract register	*SUB (register)* on page A7-450
100xx	Move	*MOV (immediate)* on page A7-312



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



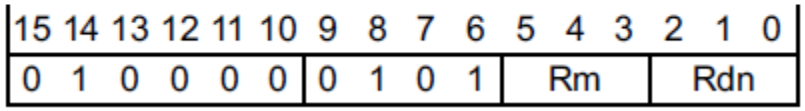
opcode	Instruction	See
0000	Bitwise AND	*AND (register)* on page A7-201
0001	Exclusive OR	*EOR (register)* on page A7-239
0010	Logical Shift Left	*LSL (register)* on page A7-300
0011	Logical Shift Right	*LSR (register)* on page A7-304
0100	Arithmetic Shift Right	*ASR (register)* on page A7-205
0101	Add 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

Quelles différences entre registres et immédiats ?

2 Registres [R0,R7]

ADCS <Rdn>,<Rm>

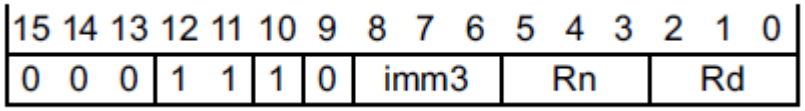
ADC<c> <Rdn>,<Rm>



Immédiat 3 bits

ADDS <Rd>,<Rn>,#<imm3>

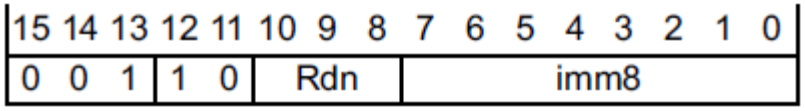
ADD<c> <Rd>,<Rn>,#<imm3>



Immédiat 8 bits

ADDS <Rdn>,#<imm8>

ADD<c> <Rdn>,#<imm8>

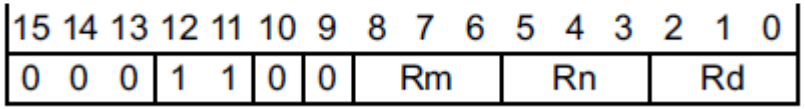


Rd = registre destination
Rn = opérande 1
Rm = Opérande 2
Rdn = destination = opérande 1
Extension de l'immédiat à zéro

Exemple sur l'addition

- 3 Registres [R0, R7]

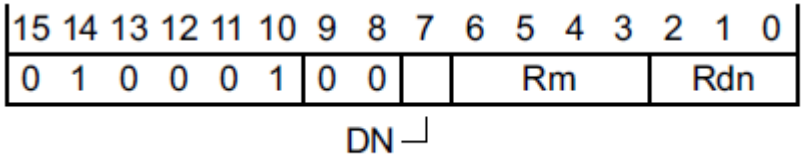
```
ADDS <Rd>,<Rn>,<Rm>
ADD<c> <Rd>,<Rn>,<Rm>
```



- registres [R0, R7] pour Rdn, [R0, R15] pour Rm

Encoding T2 All versions of the Thumb instruction set.

```
ADD<c> <Rdn>,<Rm>
```

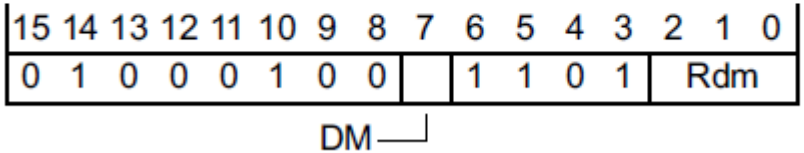


Addition sur registres spéciaux

- Add SP

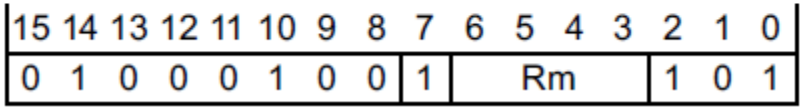
Encoding T1 All versions of the Thumb instruction set.

```
ADD<c> <Rdm>, SP, <Rdm>
```



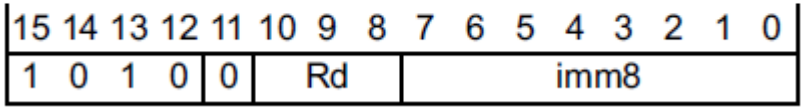
Encoding T2 All versions of the Thumb instruction set.

```
ADD<c> SP,<Rm>
```



- Add PC

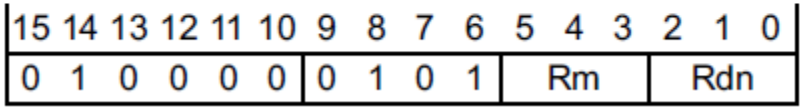
```
ADR<c> <Rd>,<label>
```



Exemple de déclinaison

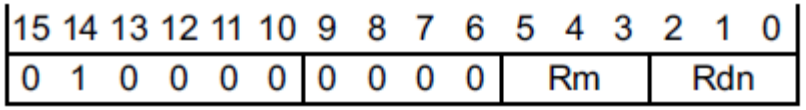
ADCS <Rdn>,<Rm>

ADC<c> <Rdn>,<Rm>



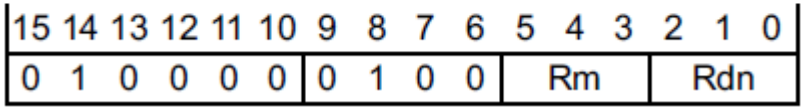
ANDS <Rdn>,<Rm>

AND<c> <Rdn>,<Rm>



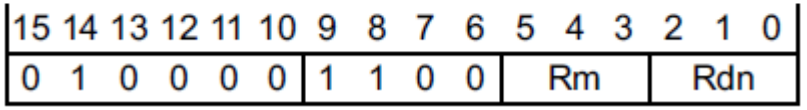
ASRS <Rdn>,<Rm>

ASR<c> <Rdn>,<Rm>



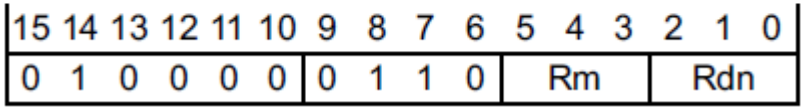
ORRS <Rdn>,<Rm>

ORR<c> <Rdn>,<Rm>



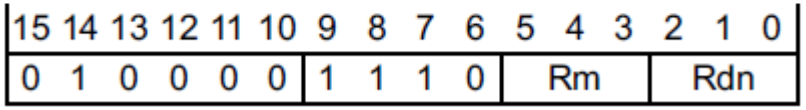
SBCS <Rdn>,<Rm>

SBC<c> <Rdn>,<Rm>

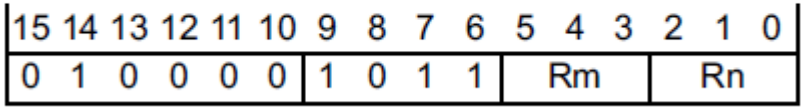


BICS <Rdn>,<Rm>

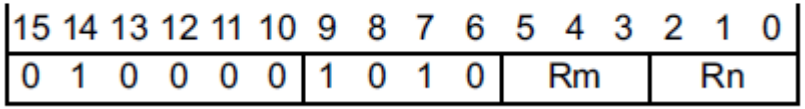
BIC<c> <Rdn>,<Rm>



CMN<c> <Rn>,<Rm>

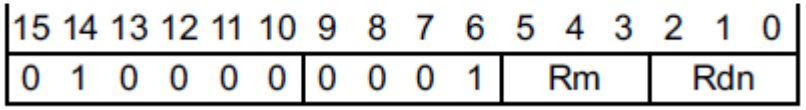


CMP<c> <Rn>,<Rm>



EORS <Rdn>,<Rm>

EOR<c> <Rdn>,<Rm>



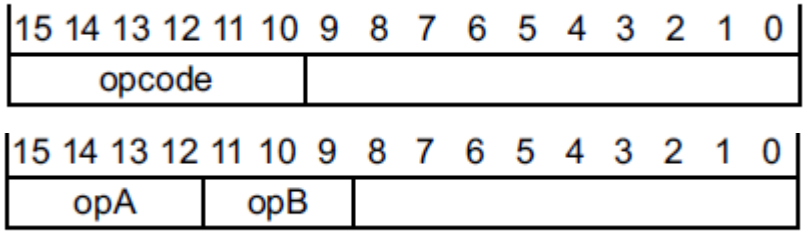
3. Load/Store instructions

- 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

Load/Store instructions

- opA prend une des valeurs suivantes :
  - 0b0101.
  - 0b011x.
  - 0b100x.
- 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

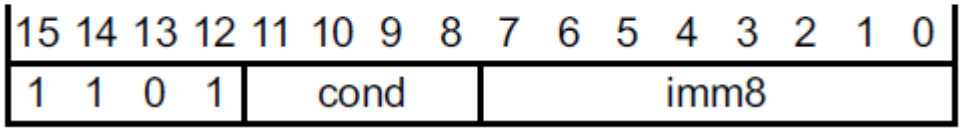


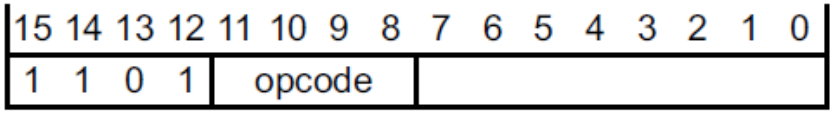
opA	opB	Instruction	See
0110	0xx	Store Register	*STR (immediate)* on page A7-426
0110	1xx	Load Register	*LDR (immediate)* on page A7-252

Branchements

- Branchement conditionnel avec immédiat sur 8 bits:

```
B<c> <label>
```

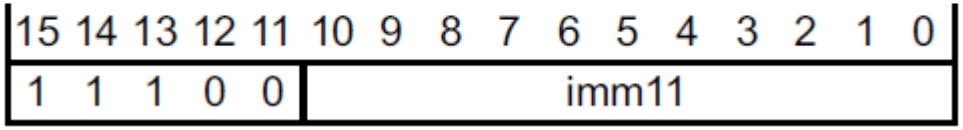




opcode	Instruction	See
not 111x	Conditional branch	*B* on page A7-207

- Branchement avec immédiat sur 11 bits

```
B<c> <label>
```



- Si la condition est vérifiée, alors PC = PC + Imm32
- Avec extension de signe

Exécution conditionnelle

- Toutes les instructions possèdent 2 formes d'exécution : conditionnelle ou non
- Dans le cas conditionnel, l'instruction n'est exécutée que si les bits de code condition sont actifs, sinon l'instruction exécutée est un NOP
- Chaque instruction possède donc un champ de 4 bits I[31..28]. Les 16 valeurs associées sont décrites dans le tableau suivant.
- L'exécution conditionnelle s'écrit :
  - INST {<cond>} {<operands>}

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

Instructions de comparaison

- Ces instructions réalisent une mise à jour des codes condition du CPSR
  - Soustraction pour CMP : CoMPare
  - Addition pour CMN : CoMpare Negative
  - ET logique pour TST : TeST
  - OU exclusif pour TEQ : Test EQuivalence

Mode d'adressage

- Adressage avec déplacement simple
  - Un registre précise l'adresse mémoire
    - Avec offset : [<Rn>, +/-<offset_12bits>]
    - Avec index : [<Rn>, +/-<Rm>]
    - Avec décalage : [<Rn>, +/-<Rm>, {LSL|LSR|ASR|ROR|RRX}]
- Adressage avec déplacement pré-indexé
  - Si l'instruction est suivie de '!', la valeur du registre <Rn> est automatiquement modifié de sa valeur +/- l'offset
- Adressage avec déplacement post-indexé
  - La modification du registre <Rn> se fait après l'accès mémoire

ORGANISATION MÉMOIRE

Instructions d’accès mémoire

Table A3-1 Load-store and element size association

Instruction class	Instructions	Element size
Load or store byte	LDR{S}B{T}, STRB{T}, TBB, LDREXB, STREXB	Byte
Load or store halfword	LDR{S}H{T}, STRH{T}, TBH, LDREXH, STREXH	Halfword
Load or store word	LDR{T}, STR{T}, LDREX, STREX, VLDR.F32, VSTR.F32	Word
Load or store two words	LDRD, STRD, VLDR.F64, VSTR.F64	Word
Load or store multiple words	LDM{IA,DB}, STM{IA,DB}, PUSH, POP, LDC, STC, VLDM, VSTM, VPUSH, VPOP	Word

Little / Big Endian

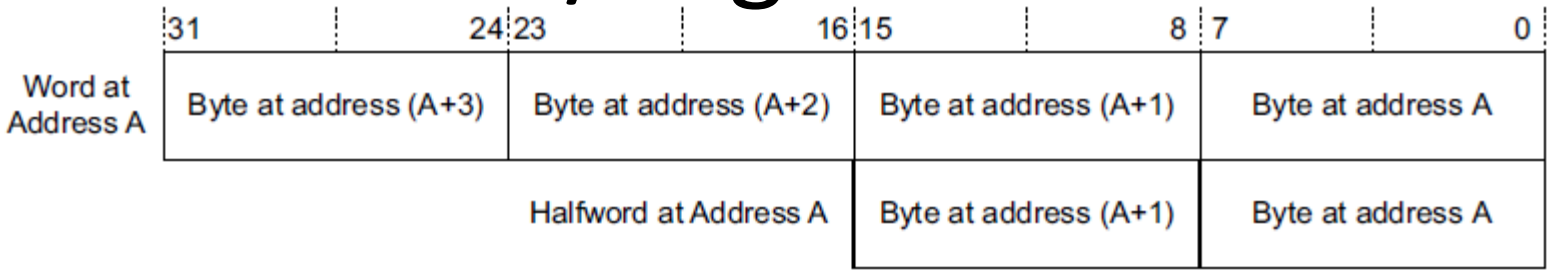


Figure A3-1 Little-endian byte format

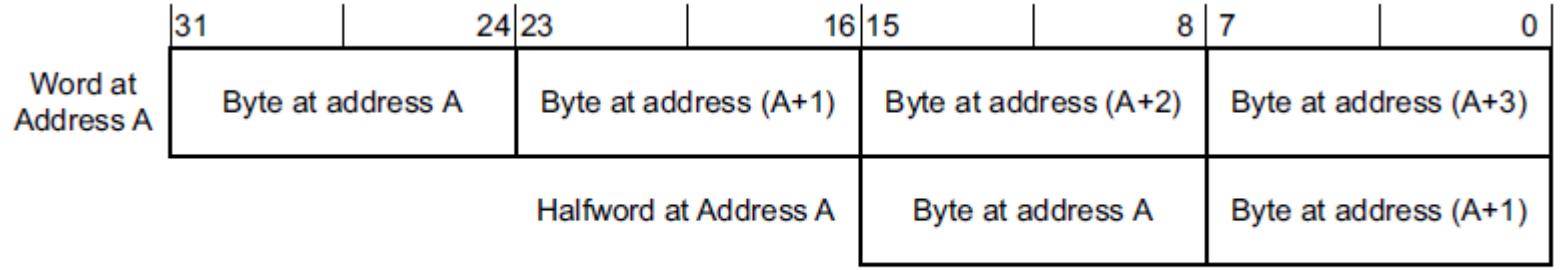


Figure A3-2 Big-endian byte format

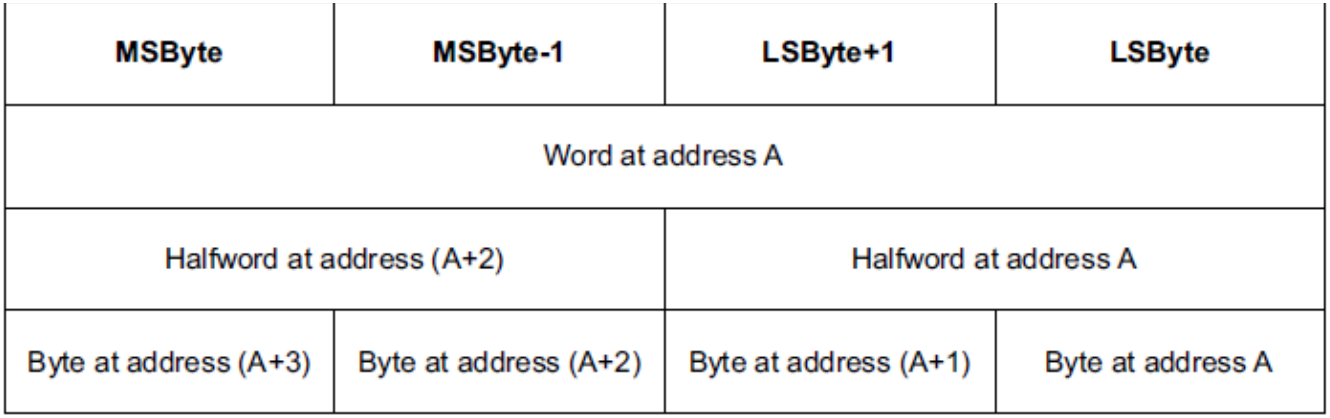


Figure A3-3 Little-endian memory system

Adresses physiques ou virtuelles

- Les adresses manipulées par l'ARMv7-M sont exclusivement des adresses physiques (PA)
- L'ARMv7-M, contrairement aux variantes A et R, n'a pas de notion d'adresse virtuelle (VA)
- Pour des raisons de compatibilité, la notion d'adresse virtuelle modifiée (MVA) a été définie

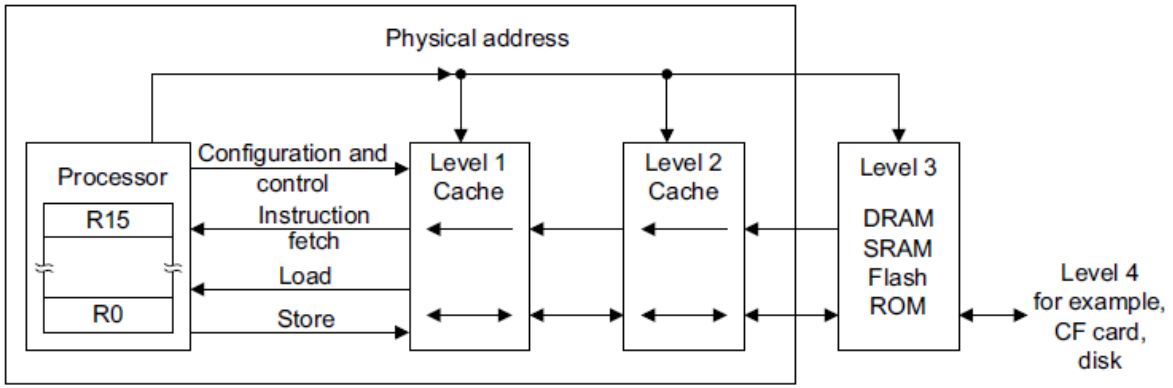


Figure A3-9 Multiple levels of cache in a memory hierarchy

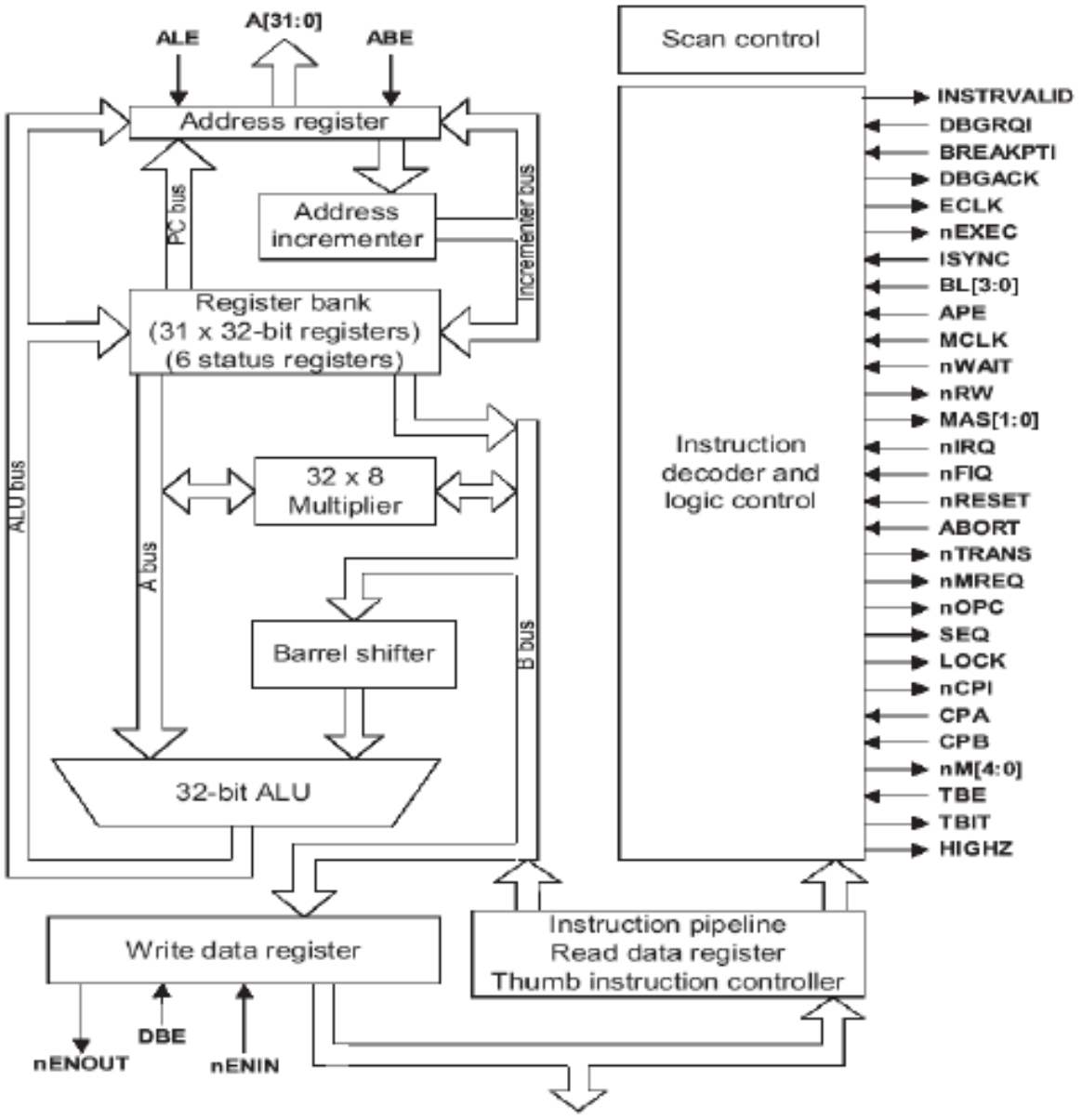
Carte d'adresses de l'ARM v7

Table B3-1 ARMv7-M address map

Address	Name	Device type	XN?	Cache	Description
0x00000000-0x1FFFFFFF	Code	Normal	-	WT	Typically ROM or flash memory.
0x20000000-0x3FFFFFFF	SRAM	Normal	-	WBWA	SRAM region typically used for on-chip RAM.
0x40000000-0x5FFFFFFF	Peripheral	Device	XN	-	On-chip peripheral address space.
0x60000000-0x7FFFFFFF	RAM	Normal	-	WBWA	Memory with write-back, write allocate cache attribute for L2/L3 cache support.

Table B3-1 ARMv7-M address map (continued)

Address	Name	Device type	XN?	Cache	Description
0x80000000-0x9FFFFFFF	RAM	Normal	-	WT	Memory with write-through cache attribute.
0xA0000000-0xBFFFFFFF	Device	Device, shareable	XN	-	Shared device space.
0xC0000000-0xDFFFFFFF	Device	Device, non-shareable	XN	-	Non-shared device space.
0xE0000000-0xFFFFFFFF	System	See *Description*	XN	-	System segment for the PPB and vendor system peripherals, see Table B3-2.


ALE
A[31:0]
ABE
Address register
Address incrementer
PC bus
Incrementer bus
Register bank
(31 x 32-bit registers)
(6 status registers)
ALU bus
A bus
B bus
32 x 8
Multiplier
Barrel shifter
32-bit ALU
Write data register
Instruction pipeline
Read data register
Thumb instruction controller
DBE
nENOUT
nENIN
Scan control
Instruction
decoder and
logic control
INSTRVALID
DBGRQI
BREAKPTI
DBGACK
ECLK
nEXEC
ISYNC
BL[3:0]
APE
MCLK
nWAIT
nRW
MAS[1:0]
nIRQ
nFIQ
nRESET
ABORT
nTRANS
nMREQ
nOPC
SEQ
LOCK
nCPI
CPA
CPB
nM[4:0]
TBE
TBIT
HIGHZ