Les instructions courantes de l'ARM Cortex-M3

Vincent Mahout
vincent.mahout@insa-toulouse.fr

January 12, 2017

Liste des suffixes "condition"possibles

Suffixe	Condition	Fanions
EQ	Égalité (**EQ**ual)	Z = 1
NE	Non égalité (**N**on **E**qual)	Z = 0
CS **HS**	Dépassement de capacité (**C**arry **S**et) Plus grand ou égal (non signé) (**H**igher or **S**ame)	C = 1
CC **LO**	Pas de dépassement de capacité (**C**arry **C**lear) Plus petit (non signé) (**L**ower)	C = 0
MI	Négatif (**M**inus)	N = 1
PL	Positif (**PL**us)	N = 0
VS	Dépassement (signé) (o**V**erflow **S**et)	V = 1
VC	Pas de dépassement (signé) (signé)(o**V**erflow **C**lear)	V = 0
HI	Plus grand (non signé) (Unsigned **HI**gher)	C = 1 ET Z = 0
LS	Plus petit ou égal (non signé) (Unsigned **L**ower or **S**ame)	C = 0 OU Z = 1
GE	Plus grand ou égal(signé) (Signed **G**reater than or **E**qual)	N = V
LT	Plus petit (signé) (Signed **L**ess **T**han)	N ≠ V
GT	Plus grand (signé) (Signed **G**reater **T**han)	Z = 0 ET N = V
LE	Plus petit ou égal(signé) (Signed **L**ess than or **E**qual)	Z = 1 OU N ≠ V

CMP a,b ... B??		
	a,b signés	a,b non signés
=	EQ	EQ
<	MI	LO
≤	LE	LS
>	GT	HI
≥	GE	HS

Les instructions arithmétiques

NOP	16/32	**Pas d'opération**
NOP<c>		ne fait rien
ADC	16/32	**Addition avec Carry**
ADC{S}<c> {<Rd>,} <Rn>, #<const>		Rd ← Rn + const + fanion C
ADC{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rd ← Rn +shift(Rm) + fanion C
ADD	16/32	**Addition simple**
ADD{S}<c> {<Rd>,} <Rn>,#<const>		Rd ← Rn + const
ADD{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rd ← Rn + shift(Rm)
ADD{S}<c> {<Rd>,} SP, #<const>		Rd ← SP + const
ADD{S}<c> {<Rd>,} SP, <Rm>{,<shift>}		Rd ← SP + shift(Rm)
MLA	32	**Multiplication et addition**
MLA<c> <Rd>, <Rn>, <Rm>, <Ra>		Rd ← (Rn * Rm) + Ra
MLS	32	**Multiplication et soustraction**
MUL	16/32	**Multiplication - Résultats sur 32 bits**
MUL{S}<c> {<Rd>,} <Rn>, <Rm>		Rd ← Rn * Rm
RSB	16/32	**Soustraction inversée**
RSB{S}<c> {<Rd>,} <Rn>, #<const>		Rd ← - Rn + const
RSB{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rd ← -Rn + shift(Rm)
SBC	16/32	**Soustraction avec Carry**
SBC{S}<c> {<Rd>,} <Rn>, #<const>		Rd ← Rm - const + fanion C
SBC{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rd ← Rm - shift(Rm) + fanion C
SDIV	32	**Division signée**
SDIV<c> {<Rd>,} <Rn>, <Rm>		Rd ← Rn ÷ Rn
SMLAL	32	**Multiplication signée et addition 64 bits**
SMLAL<c> <Rd_{pf}>, <Rd_{PF}>, <Rn>, <Rm>		[Rd_{PF} : Rd_{pf}] ← Rn * Rm +[Rd_{PF} : Rd_{pf}]
SMULL	32	**Multiplication signée - résultats sur 64 bits**
SMULL<c> <Rd_{pf}>, <Rd_{PF}>, <Rn>, <Rm>		[Rd_{PF} : Rd_{pf}] ← Rn * Rm
SSAT	32	**Saturation signée**
SSAT<c> <Rd>,#<imm5>,<Rn>{,<shift>}		si (Rn < 0) Rd ← min($-2^{(imm5-1)}$,shift(Rn)) si (Rn >0) Rd ← max($2^{(imm5-1)} - 1$,shift(Rn))

SUB	16/32	**Soustraction simple**
SUB{S}<c> {<Rd>} ,<Rn>,#<const>		Rd ← Rn - const
SUB{S}<c> {<Rd>} ,<Rn>, <Rm> {,<shift>}		Rd ← Rn - shift(Rm)
SUB{S}<c> {<Rd>} ,SP,#<const>		Rd ← SP - const
SUB{S}<c> {<Rd>} ,SP,<Rm> {,<shift>}		Rd ← SP - shift(Rm)
UDIV	32	**Division non signée**
UDIV<c> {<Rd>,} <Rn>, <Rm>		Rd ← Rn ÷ Rm
UMLAL	32	**Multiplication non signée et addition 64 bits**
UMLAL<c> <Rd_{pf}>, <Rd_{PF}>, <Rn>, <Rm>		[Rd_{PF} : Rd_{pf}] ← Rn * Rm + [Rd_{PF} : Rd_{pf}]
UMULL	32	**Multiplication non signée - Résultats sur 64 bits**
UMUL<c> <Rd_{pf}>, <Rd_{PF}>, <Rn>, <Rm>		[Rd_{PF} : Rd_{pf}] ← Rn * Rm
USAT	32	**Saturation non signée**
USAT<c> <Rd>,#<imm5>,<Rn>{,<shift>}		Rd ← max($2^{(imm5-1)} - 1$, shift(Rn))

Les instructions logique et de manipulation de bits

AND	16/32	**ET logique**
AND{S}<c> {<Rd>,} <Rn>, #<const>		Rd ← Rn AND const
AND{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rn ← Rn AND shift(Rm)
ASR	16/32	**Décalage arithmétique à droite**
ASR{S}<c> <Rd>, <Rm>, #<imm5>		Rd ← Rm >>$_{imm5}$
ASR{S}<c> <Rd>, <Rn>, <Rm>		Rd ← Rn >>$_{Rm}$
BFC	32	**Effacement de champs de bits**
BFC<c> <Rd>, #<pf>, #<Nb>		Rd[pf+Nb-1 : pf] ← 0
BFI	32	**Recopie de champs de bits**
BFI<c> <Rd>,<Rn>, #<pf>, #<Nb>		Rd[pf+Nb-1 : pf] ← Rn[Nb : 0]
BIC	16/32	**Effacement de bits par masque ET**
BIC{S}<c> {<Rd>,} <Rn>, #<const>		Rd ← Rn AND $\overline{(\text{const})}$
BIC{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rd ← Rn AND $\overline{(\text{shift(Rm)})}$
CLZ	32	**Dénombre les bits de PF à 0 devant le premier bit à 1**
CLZ<c> <Rd>, <Rm>		Rd ← CLZ(Rn)
EOR	16/32	**OU Exclusif**
EOR{S}<c> {<Rd>,} <Rn>, #<const>		Rd ← Rn XOR const
EOR{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rd ← Rn XOR shift(Rm)

LSL	16/32	**Décalage logique à gauche**
LSL{S}<c> <Rd>, <Rm>, #<imm5>		Rd ← Rm << imm5
LSL{S}<c> <Rd>, <Rn>, <Rm>		Rd ← Rn << Rm
LSR	16/32	**Décalage logique à droite**
LSR{S}<c> <Rd>, <Rm>, #<imm5>		Rd ← Rm >> imm5
LSR{S}<c> <Rd>, <Rn>, <Rm>		Rd ← Rn >> Rm
MVN	16/32	**Complément à 1 logique**
MVN{S}<c> <Rd>, #<const>		Rd ← NOT(const)
MVN{S}<c> <Rd>, <Rm> {, <shift>}		Rd ← NOT(shift(Rn))
NEG	16/32	**Complément à 2**
NEG<c> {<Rd>,} <Rm>		Rd ← -Rm
ORN	16/32	**OU logique complémenté**
ORN{S}<c> {<Rd>,} <Rn>, #<const>		Rd ← Rm OU NOT(const)
ORN{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rd ← Rm OU NOT(shift(Rm))
ORR	16/32	**OU logique**
ORR{S}<c> {<Rd>,} <Rn>, #<const>		Rd ← Rm OR const
ORR{S}<c> {<Rd>,} <Rn>, <Rm> {,<shift>}		Rd ← Rm OR shift(Rm)
RBIT	32	**Transposition de bits**
RBIT<c> <Rd>, <Rm>		Rd[31-k] ← Rm[k] avec k = 0 ... 31
REV	16/32	**Inversion octets PF et pf**
REV<c> <Rd>, <Rm>		Rd[31 : 24] ← Rm[7 : 0] Rd[23 : 16] ← Rm[15 : 8] Rd[15 : 8] ← Rm[23 : 16] Rd[7 : 0] ← Rm[31 : 24]
REV16	16/32	**Inversion octet PF et pf par $\frac{1}{2}$ mot**
REV16<c> <Rd>, <Rm>		Rd[31 : 24] ← Rm[23 : 16] Rd[23 : 16] ← Rm[31 : 24] Rd[15 : 8] ← Rm[7 : 0] Rd[7 : 0] ← Rm[15 : 8]
REVSH	16/32	**Inversion signée d'un $\frac{1}{2}$ mot**
REVSH<c> <Rd>, <Rm>		Rd[31 : 8] ← Promotion signée (Rm[7 : 0]) Rd[7 : 0] ← Rm[15 : 8]
ROR	16/32	**Rotation vers la droite**
ROR{S}<c> <Rd>, <Rm>, #<imm5>		Rd ← rotation(Rm, imm5 bits)
ROR{S}<c> <Rd>, <Rn>, <Rm>		Rd ← rotation(Rn, Rm bits)

RRX	32	**Rotation étendue vers la droite**
ROR{S}<c> <Rd>, <Rn>, <Rm>		Rd ← rotation([Rn,C], Rm bits
SBFX	32	**Promotion signée sur 32 bits d'un champs de bits**
SBFX<c> <Rd>, <Rn>, #<pf>, #<Nb>		Rd[Nb-1 : 0] ← Rn[pf+Nb-1 : pf] Rd[31 : Nb] ← Rd[pf+Nb-1]
SXTB	16/32	**Promotion signée sur 32 bits d'un octet**
SXTB<c> <Rd>, <Rm> {, <rotation>}		Rd ← rotation_{32}(Rn)[7 : 0] Rd[31 : 8] ← Rd[7]
SXTH	16/32	**Promotion signée sur 32 bits d'un $\frac{1}{2}$ mot**
SXTH<c> <Rd>, <Rm> {, <rotation>}		Rd ← rotation_{32}(Rn)[15 : 0] Rd[31 : 8] ← Rd[15]
UBFX	16/32	**Promotion non signée sur 32 bits d'un champs de bits**
UBFX<c> <Rd>, <Rn>, #<pf>, #<Nb>		Rd[Nb-1 : 0] ← Rn[pf+Nb-1 : pf] Rd[31 : Nb] ← 0
UXTB	16/32	**Promotion non signée sur 32 bits d'un octet**
UXTB<c> <Rd>, <Rm> {, <rotation>}		Rd ← rotation_{32}((Rn)[7 : 0] Rd[31 : 8] ← 0
UXTH	16/32	**Promotion non signée sur 32 bits d'un $\frac{1}{2}$ mot**
UXTH<c> <Rd>, <Rm> {, <rotation>}		Rd ← rotation_{32}((Rn)[15 : 0] Rd[31 : 8] ← 0

Les instructions de transfert interne

ADR	16/32	**Chargement d'adresse CODE**
ADR<c> <Rd>, <label>		Rd ← adresse du label
MOV	16/32	**Transfert interne de registre**
MOV{S}<c> <Rd>, #<const>		Rd ← const
MOV{S}<c> <Rd>, <Rm>		Rd ← Rm
MOVT	32	**Affectation des 16 bits de pf d'un registre**
MOVT<c> <Rd>, #<imm16>		Rd[16 : 31] ← imm16
MRS	32	**Lecture d'un registre spécial**
MSR<c> <Rn>,<spec_reg>		Rn ← spec_reg
MSR	32	**Ecriture sur un registre spécial**

Les instructions de test

CMN	16/32	**Addition sans affectation - Modification des fanions**
CMN<c> <Rn>, #<const>		Fanions ← test(Rn + const)
CMN<c> <Rn>, <Rm>{,<shift>}		Fanions ← test(Rn + shift(Rm))
CMP	16/32	**Soustraction sans affectation - Modification des fanions**
CMP<c> <Rn>, #<const>		Fanions ← test(Rn - const)
CMP<c> <Rn>, <Rm>{,<shift>}		Fanions ← test(Rn - shift(Rm))
TEQ	32	**OU exclusif sans affectation - Modification des fanions**
TEQ<c> <Rn>, #<const>		Fanions ← test(Rn XOR const)
TEQ<c> <Rn>, <Rm>{,<shift>}		Fanions ← test(Rn XOR shift(Rm))
TST	16/32	**ET logique sans affectation - Modification des fanions**
TST<c> <Rn>, #<const>		Fanions ← test(Rn AND const)
TST<c> <Rn>, <Rm>{,<shift>}		Fanions ← test(Rn AND shift(Rm))

1 Les instructions de saut

B	16/32	**Branchement simple**
B<c> <label>		PC ← label
BL	32	**Branchement avec lien**
BL<c> <label>		LR ← @ de retour PC ← label
BLX	16	**Branchement avec lien par registre**
BLX<c> <Rm>		LR ← @ de retour PC ← Rm
BX	16	**Branchement par registre**
BX<c> <Rm>		PC ← Rm
CBZ, CBNZ	16	**Branchement conditionné sur la nullité d'un registre**
CBZ<c> <Rm> <label>		PC ← label si (Rm = 0)
CBNZ<c> <Rm> <label>		PC ← label si (Rm ≠ 0)
IT	16	**Condition et saut de type si...alors**
IT{x{y{z}}} <firstcond>		Fixe l'exécution du bloc d'instructions suivantes (max 4)
TBB, TBH	32	**Table de saut relatif**
TBB<c> [<Rn>, <Rm>] TBH<c> [<Rn>, <Rm>, LSL #1]		Pc ← PC + Rn[Rm] PC + Rn[Rm]

2 Les instructions de load/store

LDR et STR sont exprimées ici en version 32 bits mais se déclinent en 8 ou 16 bit en ajoutant 'B' ou 'H' au mnémonique.

LDR	**Chargement d'un registre avec un mot mémoire**	
LDR<c> <Rt>, [<Rn> {, #±<imm>}]		Rt←$\mathcal{M}_{32}$(Rn±imm)
LDR<c> <Rt>, [<Rn>, #±<imm>]!		Rn←Rn+imm puis Rt← $\mathcal{M}_{32}$(Rn)
LDR<c> <Rt>, [<Rn>], #±<imm>		Rt←$\mathcal{M}_{32}$(Rn) puis Rn←Rn+imm
LDR<c> <Rt>, <label>		Rt ← label
LDR<c> <Rt>, [PC, #±<imm>]		Rt←$\mathcal{M}_{32}$(Pc±imm)
LDR<c> <Rt>, [<Rn>, <Rm> {LSL,#<shift>}]		Rt←$\mathcal{M}_{32}$(Rn+shift(Rm))
STR	**Déchargement d'un registre vers un mot mémoire**	
STR<c> <Rt>, [<Rn> {,#±<imm>}]		$\mathcal{M}_{32}$(Rn±imm)←Rt
STR<c> <Rt>, [<Rn>, #±<imm>]!		Rn←Rn+imm puis $\mathcal{M}_{32}$(Rn)←Rt
STR<c> <Rt>, [<Rn>], #±<imm>		$\mathcal{M}_{32}$(Rn)←Rt puis Rn←Rn+imm
STR<c> <Rt>, [<Rn>, <Rm> {, LSL #<shift>}]		$\mathcal{M}_{32}$(Rn+shift(Rm))←Rt
LDM	16/32	**Chargement multiple à partir d'adresses ascendantes**
LDM<c> <R_k>,{R_i-R_j}		R_k← $\mathcal{M}_{32}$(Rn + $4*(k-i)$) avec $k=i...j$
LDM<c> <R_k>!,{R_i-R_j}		R_k← $\mathcal{M}_{32}$(Rn + $4*(k-i)$) avec $k=i...j$ puis Rn← Rn + $4*(j-i)$
LDMDB	32	**Chargement multiple à partir d'adresses descendantes**
LDMDB<c> <R_k>,{R_i-R_j}		R_k ← $\mathcal{M}_{32}$(Rn - $4*(k-i+1)$) avec $k=i...j$
LDMDB<c> <R_k>!,{R_i-R_j}		R_k ← $\mathcal{M}_{32}$(Rn - $4*(k-i+1)$) avec $k=i...j$ puis Rn← Rn - $4*(j-i)$
LDRD	32	**Chargement double**
LDRD<c> <Rt>, <Rt2>, <litteral>		Rt← $\mathcal{M}_{32}$(litteral) Rt2← $\mathcal{M}_{32}$(litteral+4)
LDRD<c> <Rt>, <Rt2>, [PC, #±<imm>]		Rt← $\mathcal{M}_{32}$(PC+imm) Rt2 ←$\mathcal{M}_{32}$(PC+imm+4)
LDRD<c> <Rt>,<Rt2>,[<Rn>{,#±<imm>}]		Rt← $\mathcal{M}_{32}$(Rn+imm) Rt2← $\mathcal{M}_{32}$(Rn+imm+4)
LDRD<c> <Rt>,<Rt2>,[<Rn>,#±<imm>]!		Rn←Rn+imm puis Rt← $\mathcal{M}_{32}$(Rn+imm) et Rt2← $\mathcal{M}_{32}$(Rn+imm+4)
LDRD<c> <Rt>,<Rt2>,[<Rn>],#±<imm>		Rt← $\mathcal{M}_{32}$(Rn+imm) Rt2← $\mathcal{M}_{32}$(Rn+imm+4) puis Rn←Rn+imm

STM	16/32	**Déchargement multiple vers des adresses ascendantes**
STM<c> <R_k>,{R_i-R_j}		$\mathcal{M}_{32}$(Rn + $4*(k-i)$)←R_k avec $k=i...j$
STM<c> <R_k>!,{R_i-R_j}		$\mathcal{M}_{32}$(Rn + $4*(k-i)$)←R_k avec $k=i...j$ puis Rn← Rn + $4*(j-i)$
STMDB	32	**Déchargement multiple vers des adresses descendantes**
STMDB<c> <R_k>,{R_i-R_j}		$\mathcal{M}_{32}$(Rn - $4*(k-i+1)$)←R_k avec $k=i...j$
STMDB<c> <R_k>!,{R_i-R_j}		$\mathcal{M}_{32}$(Rn - $4*(k-i+1)$)←R_k avec $k=i...j$ puis Rn← Rn - $4*(j-i)$
STRD	32	**Déchargement double**
STRD<c> <Rt>,<Rt2>,[<Rn>{,#±<imm>}]		$\mathcal{M}_{32}$(Rn+imm)←Rt $\mathcal{M}_{32}$(Rn+imm+4)←Rt2
STRD<c> <Rt>,<Rt2>,[<Rn>,#±<imm>]!		Rn←Rn+imm puis $\mathcal{M}_{32}$(Rn+imm)←Rt et $\mathcal{M}_{32}$(Rn+imm+4)←Rt2
STRD<c> <Rt>,<Rt2>,[<Rn>],#±<imm>		$\mathcal{M}_{32}$(Rn+imm)←Rt $\mathcal{M}_{32}$(Rn+imm+4)←Rt2 puis Rn←Rn+imm
POP	16/32	**Déstockage depuis la pile système**
POP<c> {R_i-R_j}		R_k ←$\mathcal{M}_{32}$(SP + $4*(k-i)$) avec $k=i...j$ puis SP← SP + $4*(j-i)$
PUSH	16/32	**Stockage vers la pile système**
PUSH<c> {R_i-R_j}		$\mathcal{M}_{32}$(SP - $4*(k-i+1)$) ← R_k avec $k=i...j$ puis SP← SP - $4*(j-i)$