

## Registres ( $X_n$ = bits 0 à 63 d'un registre, $W_n$ = bits 0 à 31 d'un registre, $X_n < a : b >$ = bits a à b d'un registre)

Registre	Usage spécial	Notes
xzr	sp	Contient toujours 0 ( <i>zero register</i> ). Aucun effet lors d'une écriture. Pointeur de pile ( <i>stack pointer</i> ) pour certaines instructions d'accès mémoire.
x30	lr	Adresse de retour ( <i>link register</i> )
x29	fp	Pointeur de portée ( <i>frame pointer</i> )
x19 à x28		Registres locaux. Doivent être sauvegardés au début d'un sous-programme (voir SAVE et RESTORE).
x18		Usage dépend de l'implémentation ARM spécifique, peut être réservé pour le système.
x17	ip1	Utilisé par les bibliothèques partagées (libC par exemple). Usage temporaire autrement.
x16	ip2	Utilisé par les bibliothèques partagées (libC par exemple). Usage temporaire autrement.
x9 à x15		Usage temporaire. Ne seront pas sauvegardés par les sous-programmes.
x8		Retour indirect. Utilisé pour retourner un type de données complexe.
x0 à x7		Paramètres et retour. Ne seront pas sauvegardés par les sous-programmes.

## Conditions de branchement (Exemple: b.eq prog10 //Branche si égal vers prog10)

Nom	Signification (entiers)	Signification (virgule flottante)	Codes Condition
eq	Égal	Égal	Z==1
ne	Non égal	Non égal ou non-ordonnancé	Z==0
hs ou cs	Non-signé supérieur ou égal ( <i>report</i> )	Supérieur, égal ou non-ordonnancé	C==1
lo ou cc	Non-signé inférieur ( <i>aucun report</i> )	Inférieur	C==0
mi	Négatif	Inférieur	N==1
pl	Positif ou zéro	Supérieur, égal ou non-ordonnancé	N==0
vs	Débordement	Non-ordonnancé	V==1
vc	Aucun débordement	Ordonnancé	V==0
hi	Non-signé supérieur	Plus grand ou non-ordonnancé	C==1 && Z==0
ls	Non-signé inférieur ou égal	Inférieur ou égal	!(C==1 && Z==0)
ge	Signé supérieur ou égal	Supérieur ou égal	N==V
lt	Signé inférieur	Inférieur ou non-ordonnancé	N!=V
gt	Signé supérieur	Supérieur	Z==0 && N==V
le	Signé inférieur ou égal	Inférieur, égal ou non-ordonnancé	!(Z==0 && N==V)
al ou nv	Toujours	Toujours	Ignorés

**Modes d'adressage** ( $x_b$  = registre contenant l'adresse de base,  $X_i$  = registre contenant l'index,  $\#imm$  = immédiat contenant l'index, **etiq** = étiquette indiquant l'adresse d'une donnée)

Nom	Formes pour l'adresse	Exemples	Calcul de l'adresse
Registre différé	[ $x_b$ ]	ldr x23,[x20]	Adresse = $x_b$
Indexé	[ $x_b, X_i$ ] [ $x_b, \#imm$ ] [ $x_b, X_i, lsl \#imm$ ]	ldr x23,[x20,x21] ldr x23,[x20,#50] ldr x23,[x20,x21,lsl #3]	Adresse = $x_b + X_i$ Adresse = $x_b + \text{imm}$ Adresse = $x_b + (X_i \ll \text{imm})$
Indexé pré-incrémenté	[ $x_b, \#imm$ ]!	ldr x23,[x20,#8]!	Adresse = $x_b + \text{imm}$ Après l'accès mémoire: $x_b += \text{imm}$
Indexé post-incrémenté	[ $x_b, \#imm$ ]	ldr x23,[x20],#8	Adresse = $x_b$ Après l'accès mémoire: $x_b += \text{imm}$
Relatif	etiq	ldr x23,maConstante ... maConstante: .dword 42	Adresse = <b>etiq</b>

## Jeu d'instructions

**Contrôle** (**cond** = condition de branchement, **etiq** = étiquette de destination du branchement,  $X_n$  = registre à tester,  $X_a$  = registre contenant une adresse,  $\#uimm_6$  = immédiat de 6 bits pour choisir un bit à tester dans  $X_n$ )

Nom	Forme	Exemples	Description
b	b.cond etiq	b.eq prog40 b.gt prog20 b.al prog10	Branchement conditionnel. Les codes condition doivent être modifiés avant (comparaison ou autre). Voir le tableau des conditions de branchement.
cbnz	cbnz $X_n$ ,etiq	cbnz x20, prog40	Si $X_n \neq 0$ , alors branche à <b>etiq</b>
cbz	cbz $X_n$ ,etiq	cbz x20, prog40	Si $X_n == 0$ , alors branche à <b>etiq</b>
tbnz	tbnz $X_n,\#uimm_6$ ,etiq	tbnz x20,#43,prog40	Si $X_n < uimm_6 \neq 0$ , alors branche à <b>etiq</b>
tbz	tbz $X_n,\#uimm_6$ ,etiq	tbz x20,#43,prog40	Si $X_n < uimm_6 == 0$ , alors branche à <b>etiq</b>
bl	bl etiq	bl printf	Branche à <b>etiq</b> et copie le compteur ordinal dans $x30$ (adresse de retour).
blr	blr $X_a$	blr x20	Branche à l'adresse contenue dans $X_a$ et copie le compteur ordinal dans $x30$ (adresse de retour).
br	br $X_a$	br x20	Branche à l'adresse contenue dans $X_a$
ret	ret ret $X_a$	ret ret x20	Branche à l'adresse contenue dans $X_a$ ( $x30$ par défaut). Retour de sous-programme.

**Accès mémoire** (**x<sub>d</sub>** ou **w<sub>d</sub>** = registre destination pour la donnée lue, **x<sub>s</sub>** ou **w<sub>s</sub>** = registre source pour la donnée écrite, **adr** = adresse exprimée selon l'un des modes d'adressage, **bsg** = bit de signe de la donnée lue)

Nom	Formes	Exemples	Description
ldr	ldr x <sub>d</sub> , adr ldr w <sub>d</sub> , adr	ldr x23,[x20] ldr w23,[x20]	Lecture de 64 bits dans x <sub>d</sub> <64:0> Lecture de 32 bits dans w <sub>d</sub> <32:0>, x <sub>d</sub> <64:32> = 0
ldrb	ldrb w <sub>d</sub> , adr	ldrb w23,[x20]	Lecture de 8 bits dans w <sub>d</sub> <8:0>, x <sub>d</sub> <32:8> = 0
ldrsb	ldrsb x <sub>d</sub> , adr ldrsb w <sub>d</sub> , adr	ldrsb x23,[x20] ldrsb w23,[x20]	Lecture de 8 bits dans w <sub>d</sub> <8:0>, x <sub>d</sub> <64:8> = bsg Lecture de 8 bits dans w <sub>d</sub> <8:0>, x <sub>d</sub> <32:8> = bsg
ldrh	ldrh w <sub>d</sub> , adr	ldrh w23,[x20]	Lecture de 16 bits dans w <sub>d</sub> <16:0>, x <sub>d</sub> <32:16> = 0
ldrsh	ldrsh x <sub>d</sub> , adr ldrsh w <sub>d</sub> , adr	ldrsh x23,[x20] ldrsh w23,[x20]	Lecture de 16 bits dans w <sub>d</sub> <16:0>, x <sub>d</sub> <64:16> = bsg Lecture de 16 bits dans w <sub>d</sub> <16:0>, x <sub>d</sub> <32:16> = bsg
ldrsw	ldrsw x <sub>d</sub> , adr	ldrsw x23,[x20]	Lecture de 32 bits dans w <sub>d</sub> <32:0>, x <sub>d</sub> <64:32> = bsg
str	str x <sub>s</sub> , adr str w <sub>s</sub> , adr	str x23,[x20] str w23,[x20]	Écriture de 64 bits: x <sub>s</sub> <64:0> Écriture de 32 bits: w <sub>s</sub> <32:0>
strb	strb w <sub>s</sub> , adr	strb w23,[x20]	Écriture de 8 bits: w <sub>s</sub> <8:0>
strh	strh w <sub>s</sub> , adr	strh w23,[x20]	Écriture de 16 bits: w <sub>s</sub> <16:0>

**Arithmétique** (**x<sub>d</sub>** = registre destination, **x<sub>s1</sub>**, **x<sub>s2</sub>**, **x<sub>s3</sub>** = registres source pour les opérandes; **imm<sub>24</sub>** = constante positive sur 12 bits, décalée à gauche de 0 ou 12 bits; **imm<sub>6</sub>** = constante positive indiquant le décalage, **decal** = opérateur de décalage, parmi: lsl, asr, lsr)

Nom	Formes	Exemples	Description
add adds	add x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub> add x <sub>d</sub> , x <sub>s1</sub> , #imm <sub>24</sub> add x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub> , decal #imm <sub>6</sub>	add x23,x20,x21 add x23,x20,#100 add x23,x20,x21,lsl #3	x <sub>d</sub> = x <sub>s1</sub> + x <sub>s2</sub> x <sub>d</sub> = x <sub>s1</sub> + imm <sub>24</sub> x <sub>d</sub> = x <sub>s1</sub> + (x <sub>s2</sub> decal imm <sub>6</sub> )
sub subs	sub x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub> sub x <sub>d</sub> , x <sub>s1</sub> , #imm <sub>24</sub> sub x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub> , decal #imm <sub>6</sub>	sub x23,x20,x21 sub x23,x20,#100 sub x23,x20,x21,lsl #3	x <sub>d</sub> = x <sub>s1</sub> - x <sub>s2</sub> x <sub>d</sub> = x <sub>s1</sub> - imm <sub>24</sub> x <sub>d</sub> = x <sub>s1</sub> - (x <sub>s2</sub> decal imm <sub>6</sub> )
cmp	cmp x <sub>s1</sub> , x <sub>s2</sub> cmp x <sub>s1</sub> , #imm <sub>24</sub> cmp x <sub>s1</sub> , x <sub>s2</sub> , decal #imm <sub>6</sub>	cmp x20,x21 cmp x20,#100 cmp x20,x21, lsl #3	subs xzr, x <sub>s1</sub> , x <sub>s2</sub> subs xzr, x <sub>s1</sub> , imm <sub>24</sub> subs xzr, x <sub>s1</sub> , x <sub>s2</sub> decal imm <sub>6</sub> Compare deux valeurs.
neg negs	neg x <sub>d</sub> , x <sub>s1</sub> neg x <sub>d</sub> , x <sub>s2</sub> , decal #imm <sub>6</sub>	neg x21,x20 neg x21,x20, lsl #3	subs x <sub>d</sub> , xzr, x <sub>s1</sub> subs x <sub>d</sub> , xzr, x <sub>s2</sub> decal imm <sub>6</sub> Négation d'une valeur.
mul	mul x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub>	mul x22,x20,x21	x <sub>d</sub> = x <sub>s1</sub> * x <sub>s2</sub>
madd	madd x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub> , x <sub>s3</sub>	madd x23,x20,x22,x21	x <sub>d</sub> = (x <sub>s1</sub> * x <sub>s2</sub> ) + x <sub>s3</sub>
msub	msub x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub> , x <sub>s3</sub>	msub x23,x20,x22,x21	x <sub>d</sub> = x <sub>s3</sub> - (x <sub>s1</sub> * x <sub>s2</sub> )
sdiv	sdiv x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub>	sdiv x22,x20,x21	x <sub>d</sub> = x <sub>s1</sub> / x <sub>s2</sub> Opérandes considérés signés
udiv	sdiv x <sub>d</sub> , x <sub>s1</sub> , x <sub>s2</sub>	sdiv x22,x20,x21	x <sub>d</sub> = x <sub>s1</sub> / x <sub>s2</sub> Opérandes considérés non-signés

Note: Les instructions **adds**, **subs**, **cmp** et **negs** modifient les codes condition.

**Logique (chaînes de bits)**  $x_d$  = registre destination,  $x_{s1}, x_{s2}$  = registres source pour les opérandes;

$imm_{64}$  = constante positive sur 16 bits, peut être décalée, répétée ou sujet à inversion binaire pour l'étendre sur 64 bits; **lsb** = immédiat de 6 bits indiquant le bit le moins significatif choisi; **msb** = immédiat de 6 bits indiquant le bit le plus significatif choisi; **etiq** = étiquette indiquant une adresse, distance permise de +/- 1Mo de l'instruction courante; **bsg** = bit de signe du registre; **decal** = opérateur de décalage, parmi: lsl, asr, lsr

Nom	Formes	Exemples	Description
mov	mov $x_d, x_{s1}$ mov $x_d, \#imm_{64}$	mov x23,x20 mov x23,#300	$x_d = x_{s1}$ $x_d = imm_{64}$
adr	adr $x_d, etiq$	adr x20,tablo	$x_d = etiq$
and ands	and $x_d, x_{s1}, x_{s2}$ and $x_d, x_{s1}, \#imm_{64}$ and $x_d, x_{s1}, x_{s2}, decal \#imm_6$	and x22,x20,x21 and x22,x20,#0xFF and x22,x20,x21 lsl #3	$x_d = x_{s1} \&& x_{s2}$ $x_d = x_{s1} \&& imm_{64}$ $x_d = x_{s1} \&& (x_{s2} decal imm_6)$
eor	eor $x_d, x_{s1}, x_{s2}$ eor $x_d, x_{s1}, \#imm_{64}$ eor $x_d, x_{s1}, x_{s2}, decal \#imm_6$	eor x22,x20,x21 eor x22,,x20,#0xFF eor x22,x20,x21 lsl #3	$x_d = x_{s1} ^\wedge x_{s2}$ $x_d = x_{s1} ^\wedge imm_{64}$ $x_d = x_{s1} ^\wedge (x_{s2} decal imm_6)$
orr	orr $x_d, x_{s1}, x_{s2}$ orr $x_d, x_{s1}, \#imm_{64}$ orr $x_d, x_{s1}, x_{s2}, decal \#imm_6$	orr x22,x20,x21 orr x22,x20,#0x7B orr x22,x20,x21 lsl #3	$x_d = x_{s1}    x_{s2}$ $x_d = x_{s1}    imm_{64}$ $x_d = x_{s1}    (x_{s2} decal imm_6)$
tst	tst $x_{s1}, x_{s2}$ tst $x_{s1}, \#imm_{64}$ tst $x_{s1}, x_{s2}, decal \#imm_6$	tst x20,x21 tst x20,#0xE4 tst x20,x21 lsl #2	ands xzr, $x_{s1}, x_{s2}$ ands xzr, $x_{s1}, \#imm_{64}$ ands xzr, $x_{s1}, x_{s2}, decal \#imm_6$
bfrm	bfrm $x_d, x_{s1}, \#lsb, \#msb$	bfrm x21,x20,#8,#15	<b>si lsb &lt;= msb:</b> $x_d < msb-lsb:0 >= x_{s1} < msb:lsb >$ <b>sinon:</b> $x_d < 64+msb-lsb:64-lsb >= x_{s1} < msb:0 >$
sbfm	sbfm $x_d, x_{s1}, \#lsb, \#msb$	sbfm x21,x20,#8,#15	<b>si lsb &lt;= msb:</b> $x_d < msb-lsb:0 >= x_{s1} < msb:lsb >$ $x_d < 64:msb-lsb > = x_{s1} < msb >$ <b>sinon:</b> $x_d < 64+msb-lsb:64-lsb >= x_{s1} < msb:0 >$ $x_d < 64-lsb:0 > = 0$
ubfm	ubfm $x_d, x_{s1}, \#lsb, \#msb$	ubfm x21,x20,#8,#15	<b>si lsb &lt;= msb:</b> $x_d < msb-lsb:0 >= x_{s1} < msb:lsb >$ $x_d < 64:msb-lsb > = 0$ <b>sinon:</b> $x_d < 64+msb-lsb:64-lsb >= x_{s1} < msb:0 >$ $x_d < 64-lsb:0 > = 0$
extr	extr $x_d, x_{s1}, x_{s2}, \#lsb$	extr x22,x20,x21,#16	$x_d = [x_{s1} : x_{s2}] < 63+lsb, lsb >$
asr	asr $x_d, x_{s1}, \#imm_6$	asr x21,x20,#35	$x_d = x_{s1} >> imm_6$ $x_d < 64:64-imm_6 > = x_{s1} < bsg >$
lsl	lsl $x_d, x_{s1}, \#imm_6$	lsl x21,x20,#35	$x_d = x_{s1} << imm_6$ $x_d < imm_6:0 > = 0$
lsr	lsr $x_d, x_{s1}, \#imm_6$	lsr x21,x20,#35	$x_d = x_{s1} >> imm_6$ $x_d < 64:64-imm_6 > = 0$
ror	ror $x_d, x_{s1}, \#imm_6$	ror x21,x20,#42	$x_d = [x_{s1} : x_{s1}] < 63+lsb, lsb >$
sxtb sxth sxtw	sxtb $x_d, w_{s1}$ sxtb $w_d, w_{s1}$ sxth $x_d, w_{s1}$ sxth $w_d, w_{s1}$ sxtw $x_d, w_{s1}$	sxtb x21,w20 sxtb w21,w20 sxth x21,w20 sxth w21,w20 sxtw x21,w20	$x_d = x_{s1} < 8:0 >, x_d < 64:8 > = s_{s1} < 7 >$ $x_d = x_{s1} < 8:0 >, x_d < 32:8 > = s_{s1} < 7 >$ $x_d = x_{s1} < 16:0 >, x_d < 64:16 > = s_{s1} < 15 >$ $x_d = x_{s1} < 16:0 >, x_d < 32:16 > = s_{s1} < 15 >$ $x_d = x_{s1} < 32:0 >, x_d < 64:32 > = s_{s1} < 31 >$

uxtb	uxtb $x_d, w_{s1}$	uxtb $x_{21}, w_{20}$	$x_d = x_{s1}<8:0>, x_d<64:8>=0$
uxth	uxth $w_d, w_{s1}$	uxth $x_{21}, w_{20}$	$x_d = x_{s1}<8:0>, x_d<32:8>=0$
uxtw	uxth $x_d, w_{s1}$	uxth $w_{21}, w_{20}$	$x_d = x_{s1}<16:0>, x_d<64:16>=0$
	uxth $w_d, w_{s1}$	uxth $x_{21}, w_{20}$	$x_d = x_{s1}<16:0>, x_d<32:16>=0$
	uxtw $x_d, w_{s1}$	uxtw $x_{21}, w_{20}$	$x_d = x_{s1}<32:0>, x_d<64:16>=0$
bic bics	bic $x_d, x_{s1}, x_{s2}, \text{decal } \#imm_6$	bic $x_{22}, x_{20}, x_{21} \text{lsl } \#3$	$x_d = x_{s1} \&& !(x_{s2} \text{ decal } imm_6)$
eon	eon $x_d, x_{s1}, x_{s2}, \text{decal } \#imm_6$	eon $x_{22}, x_{20}, x_{21} \text{lsl } \#3$	$x_d = x_{s1} ^\wedge !(x_{s2} \text{ decal } imm_6)$
orn	orn $x_d, x_{s1}, x_{s2}, \text{decal } \#imm_6$	orn $x_{22}, x_{20}, x_{21} \text{lsl } \#3$	$x_d = x_{s1}    !(x_{s2} \text{ decal } imm_6)$
mvn	mvn $x_d, x_{s1}$ mvn $x_d, x_{s1} \text{ decal } \#imm_6$	mvn $x_{21}, x_{20}$ mvn $x_{21}, x_{20} \text{lsl } \#12$	$x_d = !(x_{s1})$ $x_d = x_{zr}    !(x_{s1} \text{ decal } imm_6)$
cls	cls $x_d, x_{s1}$	cls $x_{21}, x_{20}$	$x_d = \text{nombre consécutif de bits à droite de } x_{s1}<63> \text{ qui sont égaux à } x_{s1}<63>$
clz	clz $x_d, x_{s1}$	clz $x_{21}, x_{20}$	$x_d = \text{nombre consécutif de bits à droite de } x_{s1}<63> \text{ qui sont à 0}$
rbit	rbit $x_d, x_{s1}$	rbit $x_{21}, x_{20}$	$x_d = \text{les bits de } x_{s1} \text{ en ordre inverse}$
rev	rev $x_d, x_{s1}$	rev $x_{21}, x_{20}$	$x_d = x_{s1} \text{ avec l'ordre des octets inversé}$
rev16	rev16 $x_d, x_{s1}$	rev16 $x_{21}, x_{20}$	$x_d = x_{s1} \text{ avec l'ordre des octets dans chaque demi-mot inversé}$
rev32	rev32 $x_d, x_{s1}$	rev32 $x_{21}, x_{20}$	$x_d = x_{s1} \text{ avec l'ordre des octets dans chaque mot inversé}$

Note: Les instructions **ands**, **bics** et **tst** modifient les codes condition. L'instruction **extr** récupère un groupe de bits sur deux registres juxtaposés.

**Conditionnel cond** = condition de l'opération, **Xd** = registre destination; **Xs1 ,Xs2** = registres source pour les opérandes; **#uimm5** = immédiat de 5 bits ; **#uimm4** = immédiat de 4 bits contenant les codes condition dans l'ordre: N Z C V; **codes** = le registre codes condition contenant les bits N Z C V

Nom	Forme	Exemples	Description
adc adcs	adc $x_d, x_{s1}, x_{s2}$	adc $x_{22}, x_{20}, x_{21}$	$x_d = x_{s1} + x_{s2} + \mathbf{C}$
csel	csel $x_d, x_{s1}, x_{s2}, \text{cond}$	csel $x_{22}, x_{20}, x_{21}, \text{gt}$	Si cond alors $x_d = x_{s1}$ , sinon $x_d = x_{s2}$
csinc	csinc $x_d, x_{s1}, x_{s2}, \text{cond}$	csinc $x_{22}, x_{20}, x_{21}, \text{eq}$	Si cond alors $x_d = x_{s1}$ , sinon $x_d = x_{s2} + 1$
csinv	csinv $x_d, x_{s1}, x_{s2}, \text{cond}$	csinv $x_{22}, x_{20}, x_{21}, \text{ne}$	Si cond alors $x_d = x_{s1}$ , sinon $x_d = !(x_{s2})$
csneg	csneg $x_d, x_{s1}, x_{s2}, \text{cond}$	csinv $x_{22}, x_{20}, x_{21}, \text{lt}$	Si cond alors $x_d = x_{s1}$ , sinon $x_d = -x_{s2}$
sbc sbcs	sbc $x_d, x_{s1}, x_{s2}$	sbc $x_{22}, x_{20}, x_{21}$	$x_d = x_{s1} - x_{s2} - 1 + \mathbf{C}$
ccmn	ccmn $x_{s1}, x_{s2}, \#imm_4, \text{cond}$ ccmn $x_{s1}, \#imm_5, \#imm_4, \text{cond}$	ccmn $x_{20}, x_{21}, \#3, \text{gt}$ ccmn $x_{20}, \#25, \text{gt}$	Si cond alors codes = cmp $x_{s1}, -(x_{s2})$ sinon codes = imm <sub>4</sub> Si cond alors codes = cmp $x_{s1}, -(imm_5)$ sinon codes = imm <sub>4</sub>
ccmp	ccmp $x_{s1}, x_{s2}, \#imm_4, \text{cond}$ ccmp $x_{s1}, \#imm_5, \#imm_4, \text{cond}$	ccmp $x_{20}, x_{21}, \#3, \text{gt}$ ccmp $x_{20}, \#25, \text{gt}$	Si cond alors codes = cmp $x_{s1}, x_{s2}$ sinon codes = imm <sub>4</sub> Si cond alors codes = cmp $x_{s1}, imm_5$ sinon codes = imm <sub>4</sub>

Note: Les instructions **adcs**, **sbcs** **ccmn** et **ccmp** modifient les codes condition

**Virgule flottante**  $x_d, w_d$  = registre destination [entiers],  $x_{s1}, w_{s1}$  = registre source [entiers]; **fpimm<sub>8</sub>** = constante quart-précision sur 8 bits; **imm<sub>6</sub>** = constante positive indiquant le décalage, **adr** = adresse exprimée selon l'un des modes d'adressage; **b<sub>d</sub>, h<sub>d</sub>, s<sub>d</sub>, d<sub>d</sub>, q<sub>d</sub>** = registre destination [virgule flottante] en quart, demi, simple, double ou quadruple précision (8,16,32,64 ou 128 bits); **b<sub>s1</sub>, h<sub>s1</sub>, s<sub>s1</sub>, d<sub>s1</sub>, q<sub>s1</sub>** = registres source [virgule flottante]. **FPCR** = registre de configuration de l'unité virgule flottante; **fbits** = Position de la virgule dans un champ de bits à virgule fixe; **#uimm<sub>4</sub>** = immédiat de 4 bits contenant les codes condition dans l'ordre: N Z C V; **cond** = condition de l'opération; **codes** = le registre codes condition contenant les bits N Z C V.

Nom	Formes	Exemples	Description
ldr	ldr b <sub>d</sub> , adr ldr h <sub>d</sub> , adr ldr s <sub>d</sub> , adr ldr d <sub>d</sub> , adr ldr q <sub>d</sub> , adr	ldr b12,[x20] ldr h12,[x20] ldr s12,[x20] ldr d12,[x20] ldr q12,[x20]	Lecture de 8 bits dans <b>b<sub>d</sub></b> Lecture de 16 bits dans <b>h<sub>d</sub></b> Lecture de 32 bits dans <b>s<sub>d</sub></b> Lecture de 64 bits dans <b>d<sub>d</sub></b> Lecture de 128 bits dans <b>q<sub>d</sub></b>
str	str b <sub>d</sub> , adr str h <sub>d</sub> , adr str s <sub>d</sub> , adr str d <sub>d</sub> , adr str q <sub>d</sub> , adr	str b12,[x20] str h12,[x20] str s12,[x20] str d12,[x20] str q12,[x20]	Écriture des 8 bits de <b>b<sub>d</sub></b> Écriture des 16 bits de <b>h<sub>d</sub></b> Écriture des 32 bits de <b>s<sub>d</sub></b> Écriture des 64 bits de <b>d<sub>d</sub></b> Écriture des 128 bits de <b>q<sub>d</sub></b>
fmov	fmov s <sub>d</sub> , s <sub>s1</sub> fmov d <sub>d</sub> , d <sub>s1</sub> fmov w <sub>d</sub> , s <sub>s1</sub> fmov x <sub>d</sub> , d <sub>s1</sub> fmov s <sub>d</sub> , w <sub>s1</sub> fmov d <sub>d</sub> , x <sub>s1</sub> fmov s <sub>d</sub> , fpimm <sub>8</sub> fmov d <sub>d</sub> , fpimm <sub>8</sub>	fmov s4,s5 fmov d4,d5 fmov w20,s5 fmov x20,d5 fmov s4,w20 fmov d4,x20 fmov s4,1.5 fmov d4,1.5	Copie une valeur d'un registre à un autre, ou copie un immédiat quart précis (8 bits) dans un registre. Pour initialiser à 0 un registre virgule flottante, il faut utiliser <b>wzr</b> ou <b>xzr</b> comme registre source. Les immédiats seront convertis vers le format de destination.
fcvt	fcvt s <sub>d</sub> , h <sub>s1</sub> fcvt h <sub>d</sub> , s <sub>s1</sub> fcvt d <sub>d</sub> , h <sub>s1</sub> fcvt h <sub>d</sub> , d <sub>s1</sub> fcvt d <sub>d</sub> , s <sub>s1</sub> fcvt s <sub>d</sub> , d <sub>s1</sub>	fcvt s0,h1 fcvt h0,s1 fcvt d0,h1 fcvt h0,d1 fcvt d0,s1 fcvt s0,d1	Conversion d'une valeur virgule flottante, d'un niveau de précision à un autre.
scvtf	scvtf s <sub>d</sub> , w <sub>s1</sub> scvtf s <sub>d</sub> , x <sub>s1</sub> scvtf d <sub>d</sub> , w <sub>s1</sub> scvtf d <sub>d</sub> , x <sub>s1</sub>	scvtf s0,w20 scvtf s0,x20 scvtf d0,w20 scvtf d0,x20	Conversion d'entier <b>signé</b> vers virgule flottante, en utilisant le mode d'arrondi configuré actuellement dans <b>FPCR</b> .
ucvtf	ucvtf s <sub>d</sub> , w <sub>s1</sub> ucvtf s <sub>d</sub> , x <sub>s1</sub> ucvtf d <sub>d</sub> , w <sub>s1</sub> ucvtf d <sub>d</sub> , x <sub>s1</sub>	ucvtf s0,w20 ucvtf s0,x20 ucvtf d0,w20 ucvtf d0,x20	Conversion d'entier <b>non-signé</b> vers virgule flottante, en utilisant le mode d'arrondi configuré actuellement dans <b>FPCR</b> .
fcvtzs	fcvtzs w <sub>d</sub> , s <sub>s1</sub> fcvtzs w <sub>d</sub> , d <sub>s1</sub> fcvtzs x <sub>d</sub> , s <sub>s1</sub> fcvtzs x <sub>d</sub> , d <sub>s1</sub>	fcvtzs w20,s5 fcvtzs x20,s5 fcvtzs w20,d5 fcvtzs x20,d5	Conversion de virgule flottante à <b>entier signé</b> , en arrondissant vers zéro.
fcvtzu	fcvtzu w <sub>d</sub> , s <sub>s1</sub> fcvtzu w <sub>d</sub> , d <sub>s1</sub> fcvtzu x <sub>d</sub> , s <sub>s1</sub> fcvtzu x <sub>d</sub> , d <sub>s1</sub>	fcvtzu w20,s5 fcvtzu x20,s5 fcvtzu w20,d5 fcvtzu x20,d5	Conversion de virgule flottante à <b>entier non-signé</b> , en arrondissant vers zéro.
frintz	frintx s <sub>d</sub> , s <sub>s1</sub> frintx d <sub>d</sub> , d <sub>s1</sub>	frintx s1,s0 frintx d1,d0	Arrondi à la partie entière d'un nombre virgule flottante.

<b>fabs</b>	<b>fabs <math>s_d, s_{s1}</math></b> <b>fabs <math>d_d, d_{s1}</math></b>	<b>fabs <math>s1, s0</math></b> <b>fabs <math>d1, d0</math></b>	Valeur absolue d'un nombre à virgule flottante.
<b>fneg</b>	<b>fneg <math>s_d, s_{s1}</math></b> <b>fneg <math>d_d, d_{s1}</math></b>	<b>fneg <math>s1, s0</math></b> <b>fneg <math>d1, d0</math></b>	$s_d = -s_{s1}$ $d_d = -d_{s1}$
<b>fsqrt</b>	<b>fsqrt <math>s_d, s_{s1}</math></b> <b>fsqrt <math>d_d, d_{s1}</math></b>	<b>fsqrt <math>s1, s0</math></b> <b>fsqrt <math>d1, d0</math></b>	Racine carrée d'un nombre à virgule flottante.
<b>fadd</b>	<b>fadd <math>s_d, s_{s1}, s_{s2}</math></b> <b>fadd <math>d_d, d_{s1}, d_{s2}</math></b>	<b>fadd <math>s3, s1, s2</math></b> <b>fadd <math>d3, d1, d2</math></b>	$s_d = s_{s1} + s_{s2}$ $d_d = d_{s1} + d_{s2}$
<b>fmul</b>	<b>fmul <math>s_d, s_{s1}, s_{s2}</math></b> <b>fmul <math>d_d, d_{s1}, d_{s2}</math></b>	<b>fmul <math>s3, s1, s2</math></b> <b>fmul <math>d3, d1, d2</math></b>	$s_d = s_{s1} * s_{s2}$ $d_d = d_{s1} * d_{s2}$
<b>fnmul</b>	<b>fnmul <math>s_d, s_{s1}, s_{s2}</math></b> <b>fnmul <math>d_d, d_{s1}, d_{s2}</math></b>	<b>fnmul <math>s3, s1, s2</math></b> <b>fnmul <math>d3, d1, d2</math></b>	$s_d = -(s_{s1} * s_{s2})$ $d_d = -(d_{s1} * d_{s2})$
<b>fsub</b>	<b>fsub <math>s_d, s_{s1}, s_{s2}</math></b> <b>fsub <math>d_d, d_{s1}, d_{s2}</math></b>	<b>fsub <math>s3, s1, s2</math></b> <b>fsub <math>d3, d1, d2</math></b>	$s_d = s_{s1} - s_{s2}$ $d_d = d_{s1} - d_{s2}$
<b>fdiv</b>	<b>fdiv <math>s_d, s_{s1}, s_{s2}</math></b> <b>fdiv <math>d_d, d_{s1}, d_{s2}</math></b>	<b>fdiv <math>s3, s1, s2</math></b> <b>fdiv <math>d3, d1, d2</math></b>	$s_d = s_{s1} / s_{s2}$ $d_d = d_{s1} / d_{s2}$
<b>fmax</b>	<b>fmax <math>s_d, s_{s1}, s_{s2}</math></b> <b>fmax <math>d_d, d_{s1}, d_{s2}</math></b>	<b>fmax <math>s3, s1, s2</math></b> <b>fmax <math>d3, d1, d2</math></b>	$s_d = \max(s_{s1}, s_{s2})$ $d_d = \max(d_{s1}, d_{s2})$
<b>fmin</b>	<b>fmin <math>s_d, s_{s1}, s_{s2}</math></b> <b>fmin <math>d_d, d_{s1}, d_{s2}</math></b>	<b>fmin <math>s3, s1, s2</math></b> <b>fmin <math>d3, d1, d2</math></b>	$s_d = \min(s_{s1}, s_{s2})$ $d_d = \min(d_{s1}, d_{s2})$
<b>fmadd</b>	<b>fmadd <math>s_d, s_{s1}, s_{s2}, s_{s3}</math></b> <b>fmadd <math>d_d, d_{s1}, d_{s2}, d_{s3}</math></b>	<b>fmadd <math>s3, s0, s1, s2</math></b> <b>fmadd <math>d3, d0, d1, d2</math></b>	$s_d = s_{s3} + (s_{s1} * s_{s2})$ $d_d = d_{s3} + (d_{s1} * d_{s2})$
<b>fmsub</b>	<b>fmsub <math>s_d, s_{s1}, s_{s2}, s_{s3}</math></b> <b>fmsub <math>d_d, d_{s1}, d_{s2}, d_{s3}</math></b>	<b>fmsub <math>s3, s0, s1, s2</math></b> <b>fmsub <math>d3, d0, d1, d2</math></b>	$s_d = s_{s3} + (-s_{s1} * s_{s2})$ $d_d = d_{s3} + (-d_{s1} * d_{s2})$
<b>fnmadd</b>	<b>fnmadd <math>s_d, s_{s1}, s_{s2}, s_{s3}</math></b> <b>fnmadd <math>d_d, d_{s1}, d_{s2}, d_{s3}</math></b>	<b>fnmadd <math>s3, s0, s1, s2</math></b> <b>fnmadd <math>d3, d0, d1, d2</math></b>	$s_d = -s_{s3} + (-s_{s1} * s_{s2})$ $d_d = -d_{s3} + (-d_{s1} * d_{s2})$
<b>fnmsub</b>	<b>fnmsub <math>s_d, s_{s1}, s_{s2}, s_{s3}</math></b> <b>fnmsub <math>d_d, d_{s1}, d_{s2}, d_{s3}</math></b>	<b>fnmsub <math>s3, s0, s1, s2</math></b> <b>fnmsub <math>d3, d0, d1, d2</math></b>	$s_d = (-s_{s1} * s_{s2}) - s_{s3}$ $d_d = (-d_{s1} * d_{s2}) - d_{s3}$
<b>fcmp</b>	<b>fcmp <math>s_{s1}, s_{s2}</math></b> <b>fcmp <math>s_{s1}, 0.0</math></b> <b>fcmp <math>d_{s1}, d_{s2}</math></b> <b>fcmp <math>d_{s1}, 0.0</math></b>	<b>fcmp <math>s4, s5</math></b> <b>fcmp <math>s4, 0.0</math></b> <b>fcmp <math>d4, d5</math></b> <b>fcmp <math>d4, 0.0</math></b>	Comparaison de nombres virgule flottante, modifie les codes condition (N Z C V).
<b>fccmp</b>	<b>ccmp <math>s_{s1}, s_{s2}, #imm_4, cond</math></b> <b>ccmp <math>d_{s1}, d_{s2}, #imm_4, cond</math></b>	<b>ccmp <math>s20, s21, #3, gt</math></b> <b>ccmp <math>d20, d21, #3, gt</math></b>	Si cond alors codes = <b>fcmp <math>s_{s1}, s_{s2}</math></b> sinon codes = <b>imm<sub>4</sub></b> Si cond alors codes = <b>fcmp <math>d_{s1}, d_{s2}</math></b> sinon codes = <b>imm<sub>4</sub></b>
<b>fcsel</b>	<b>csel <math>s_d, s_{s1}, s_{s2}, cond</math></b> <b>csel <math>d_d, d_{s1}, d_{s2}, cond</math></b>	<b>csel <math>s22, s20, s21, gt</math></b> <b>csel <math>d22, d20, d21, gt</math></b>	Si cond alors $s_d = s_{s1}$ , sinon $s_d = s_{s2}$ Si cond alors $d_d = d_{s1}$ , sinon $d_d = d_{s2}$

Note: Les instructions **fcmp**, **fccmp**, et **fcsel** modifient les codes condition. Des instructions de conversion et d'arrondi avec des modes d'arrondi supplémentaires existent (voir la référence ARM complète pour plus de détails).

L'arithmétique quadruple précision n'est pas supportée sur cette architecture, bien qu'on puisse transférer de la mémoire des valeurs de 128 bits vers une paire de registres double-précision et vice-versa. Un registre quadruple précision identifie une paire de registre double précision consécutifs :  $q_0 = [d_2 : d_1]$ ,  $q_1 = [d_4 : d_3]$ , ...

# Informations complémentaires

**Formats** (Utilisés pour printf et scanf. Sont déclarés dans des chaînes de caractères à l'aide de **pseudo-instructions**)

Format	Lecture	Écriture
%c	Caractère → Code Ascii	Code Ascii → Caractère
%s	Chaîne → Suite de codes Ascii	Suite de codes Ascii → Chaîne
%d	Entier décimal → Binaire	Binaire → Entier décimal
%u	Entier non-signé décimal → Binaire	Binaire → Entier non-signé décimal
%o	Entier octal → Binaire	Binaire → Entier octal
%x	Entier hexadécimal → Binaire	Binaire → Entier hexadécimal
%f	Virgule flottante simple précision → Binaire	Binaire → Virgule flottante double précision
%ll	Entier décimal → Binaire 64 bits	Binaire 64 bits → Entier décimal

**Appels système** (x8: code de service, x0 à x7: paramètres, ensuite: **svc 0**)

Service	Code de service (x8)	Paramètres
exit	93	x0 : <b>code d'erreur</b> (0 pour fin normale)
read	63	x0 : 0 (stdin) x1 : <b>adresse</b> où écrire la chaîne de caractères à lire x2 : <b>nombre</b> de caractères à lire
write	64	x0 : 1 (stdout) x1 : <b>adresse</b> de la chaîne de caractères à imprimer x2 : <b>longueur</b> de la chaîne de caractères à imprimer
close flush	57	x0 : 0 (tous les flots)

## Table de conversion

Décimal	Hexadécimal	Binaire	Décimal	Hexadécimal	Binaire
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

## Pseudo-Instructions

(Directives pour l'assembleur. Ne sont pas des instructions machine).

Nom	Formes	Exemples	Description
.section	.section "section"	.section ".text" .section ".bss" .section ".rodata" .section ".data"	Changement de section du programme. text = instructions, bss = données non-initialisées, rodata = constantes, data = données initialisées.
.global	.global étiquette	.global Main	Publie une étiquette pour accès par d'autres modules, ou l'éditeur de liens. Utile pour le point d'entrée du programme.
.align	.align valeur	.align 8 .align 16	S'assure que la prochaine donnée en mémoire soit à une adresse multiple de <b>valeur</b> .
.skip	.skip taille	.skip 100 .skip 100*4	Réserve la quantité d'octets indiquée par <b>taille</b> .
.ascii	.ascii chaîne	.ascii "bonjour"	Prépare une chaîne de caractères en mémoire.
.asciz	.asciz chaîne	.asciz "bonjour"	Prépare une chaîne de caractères en mémoire, puis ajoute le caractère de fin de chaîne à la fin.
.byte	.byte valeur	.byte 50 .byte 0x30 .byte -5 .byte 'a'	Prépare une valeur pouvant tenir sur un octet en mémoire.
.half	.half valeur	.half 16565	Prépare une valeur pouvant tenir sur deux octets en mémoire.
.word	.word valeur	.word 2095437	Prépare une valeur pouvant tenir sur quatre octets en mémoire.
.xword	.xword valeur	.xword 797867987987	Prépare une valeur pouvant tenir sur huit octets en mémoire.
.single	.single valeur	.single 0r3.14	Prépare une nombre virgule flottante simple précision en mémoire.
.double	.double valeur	.double 0r3.14	Prépare une nombre virgule flottante double précision en mémoire.

# Les instructions les plus utiles

Nom	Formes habituelles des paramètres	Catégorie	Description
mov	2 registres 1 registre et 1 immédiat	Arithmétique / Logique	Copie des données.
add	3 registres 2 registres et 1 immédiat	Arithmétique	Addition d'entiers.
sub	3 registres 2 registres et 1 immédiat	Arithmétique	Soustraction d'entiers.
mul	3 registres	Arithmétique	Multiplication d'entiers.
sdiv	3 registres	Arithmétique	Division d'entiers signés.
udiv	3 registres	Arithmétique	Division d'entiers non-signés.
cmp	2 registres 1 registre et 1 immédiat	Arithmétique	Comparaison de valeurs. Modifie les <b>codes condition</b> .
b.cond	1 condition et 1 étiquette identifiant une ligne de code	Contrôle	Si la condition est vraie, déplace le contrôle à la ligne de code identifiée par l'étiquette. Lit les <b>codes condition</b> .
bl	Étiquette identifiant le début d'un sous-programme.	Contrôle	Appel de sous-programme.
ret	Aucun paramètre	Contrôle	Retourne au programme appelant.
SAVE	Aucun paramètre	Macro	Sauvegarde l'état des registres x19 à x30.
RESTORE	Aucun paramètre	Macro	Récupère l'ancien état des registres x19 à x30.
ldr	1 registre, 1 adresse L'adresse est formée de 1 ou deux registres.	Accès mémoire	Lit 4 ou 8 octets à l'adresse spécifiée en mémoire.
str	1 registre, 1 adresse L'adresse est formée de 1 ou deux registres.	Accès mémoire	Écrit 4 ou 8 octets à l'adresse spécifiée en mémoire.
adr	1 registre et 1 étiquette identifiant une déclaration de données	Logique	Copie l'adresse des données identifiées par l'étiquette dans un registre.
and	3 registres 2 registres et 1 immédiat	Logique	ET logique bit à bit
eor	3 registres 2 registres et 1 immédiat	Logique	OU EXCLUSIF logique bit à bit
orr	3 registres 2 registres et 1 immédiat	Logique	OU logique bit à bit
lsl	3 registres 2 registres et 1 immédiat	Logique	Décalage logique à gauche
lsr	3 registres 2 registres et 1 immédiat	Logique	Décalage logique à droite