

Registres.

- ▶ Chaque registre x_n possède 64 bits: $b_{63}b_{62} \dots b_1b_0$
- ▶ Notation: $x_n(i) \stackrel{\text{def}}{=} b_i$, $x_n(i, j) \stackrel{\text{def}}{=} b_i b_{i-1} \dots b_j$, r_n réfère au registre x_n ou w_n
- ▶ Chaque sous-registre w_n possède 32 bits et correspond à $x_n\langle 31, 0 \rangle$
- ▶ Le compteur d'instruction pc n'est pas accessible
- ▶ Conventions:

Registres	Nom	Utilisation
$x_0 - x_7$	—	registres d'arguments et de retour de sous-programmes
x_8	xr	registre pour retourner l'adresse d'une structure
$x_9 - x_{15}$	—	registres temporaires sauvegardés par l'appelant
$x_{16} - x_{17}$	ip ₀ - ip ₁	registres temporaires intra-procéduraux
x_{18}	pr	registre temporaire pouvant être réservé par le système
$x_{19} - x_{28}$	—	registres temporaires sauvegardés par l'appelé
x_{29}	fp	pointeur vers l'ancien sommet de pile (<i>frame pointer</i>)
x_{30}	lr	registre d'adresse de retour (<i>link register</i>)
x_{2r}	sp	registre contenant la valeur 0, ou pointeur de pile (<i>stack pointer</i>)

Arithmétique (entiers).

- ▶ Les codes de condition sont modifiés par **cmp**, **adds**, **subs** et **negs**
- ▶ À cette différence près, **adds**, **adcs**, **subs** et **negs** se comportent respectivement comme **add**, **adc**, **sub** et **neg**
- ▶ Instructions, où i est une valeur immédiate de 12 bits et j est une valeur immédiate de 6 bits:

Code d'op.	Syntaxe	Effet	Exemple
cmp	cmp rd, rm	compare r_d et r_m	cmp x19, x21
	cmp rd, i	compare r_d et i	cmp x19, 42
	cmp rd, rm, decal j	compare r_d et r_m <i>decal j</i>	cmp x19, x21, lsl 1
add	add rd, rn, rm	$r_d \leftarrow r_n + r_m$	add x19, x20, x21
	add rd, rn, i	$r_d \leftarrow r_n + i$	add x19, x20, 42
	add rd, rn, rm, decal j	$r_d \leftarrow r_n + (r_m \text{ decal } j)$	add x19, x20, x21, lsl 1
adc	adc rd, rn, rm	$r_d \leftarrow r_n + r_m + C$	adc x19, x20, x21
sub	sub rd, rn, rm	$r_d \leftarrow r_n - r_m$	sub x19, x20, x21
	sub rd, rn, i	$r_d \leftarrow r_n - i$	sub x19, x20, 42
	sub rd, rn, rm, decal j	$r_d \leftarrow r_n - (r_m \text{ decal } j)$	sub x19, x20, x21, lsl 1
neg	neg rd, rm	$r_d \leftarrow -r_m$	neg x19, x21
	neg rd, rm, decal j	$r_d \leftarrow -(r_m \text{ decal } j)$	neg x19, x21, lsl 1
mul	mul rd, rn, rm	$r_d \leftarrow r_n \cdot r_m$	mul x19, x20, x21
udiv	udiv rd, rn, rm	$r_d \leftarrow r_n \div r_m$ (non signé)	udiv x19, x20, x21
sdiv	sdiv rd, rn, rm	$r_d \leftarrow r_n \div r_m$ (signé)	sdiv x19, x20, x21
madd	madd rd, rn, rm, ra	$r_d \leftarrow r_a + (r_n \cdot r_m)$	madd x19, x20, x21, x22
msub	msub rd, rn, rm, ra	$r_d \leftarrow r_a - (r_n \cdot r_m)$	msub x19, x20, x21, x22

Accès mémoire.

- ▶ **ldrsb**, **ldrsh** et **ldrsb** se comportent respectivement comme **ldr** (4 octets), **ldrh** et **ldrb** à l'exception du fait qu'ils effectuent un chargement dans x_d où les bits excédentaires sont le bit de signe de la donnée chargée, plutôt que des zéros
- ▶ Instructions, où a est une adresse et $\text{mem}_b[a]$ réfère aux b octets à l'adresse a de la mémoire principale:

Code d'op.	Syntaxe	Effet	Exemple
mov	mov rd, rm	$r_d \leftarrow r_m$	mov x19, x21
	mov rd, i	$r_d \leftarrow i$	mov x19, 42
ldr	ldr xd, a	charge 8 octets: $x_d\langle 63, 0 \rangle \leftarrow \text{mem}_8[a]$	ldr x19, [x20]
	ldr wd, a	charge 4 octets: $x_d\langle 31, 0 \rangle \leftarrow \text{mem}_4[a]$; $x_d\langle 63, 32 \rangle \leftarrow 0$	ldr w19, [x20]
ldrh	ldrh wd, a	charge 2 octets: $x_d\langle 15, 0 \rangle \leftarrow \text{mem}_2[a]$; $x_d\langle 63, 16 \rangle \leftarrow 0$	ldrh w19, [x20]
ldrb	ldrb wd, a	charge 1 octet: $x_d\langle 7, 0 \rangle \leftarrow \text{mem}_1[a]$; $x_d\langle 63, 8 \rangle \leftarrow 0$	ldrb w19, [x20]
str	str xd, a	stocke 8 octets: $\text{mem}_8[a] \leftarrow x_d\langle 63, 0 \rangle$	str x19, [x20]
	str wd, a	stocke 4 octets: $\text{mem}_4[a] \leftarrow x_d\langle 31, 0 \rangle$	str w19, [x20]
strh	strh wd, a	stocke 2 octets: $\text{mem}_2[a] \leftarrow x_d\langle 15, 0 \rangle$	strh w19, [x20]
strb	strb wd, a	stocke 1 octet: $\text{mem}_1[a] \leftarrow x_d\langle 7, 0 \rangle$	strb w19, [x20]
ldp	ldp xd, xn, a	charge 16 octets: $x_d\langle 63, 0 \rangle \leftarrow \text{mem}_8[a]$, $x_n\langle 63, 0 \rangle \leftarrow \text{mem}_8[a + 8]$	ldp x19, x20, [sp]
stp	stp xd, xn, a	stocke 16 octets: $\text{mem}_8[a] \leftarrow x_d\langle 63, 0 \rangle$, $\text{mem}_8[a + 8] \leftarrow x_n\langle 63, 0 \rangle$	stp x19, x20, [sp]

Conditions de branchement.

- Codes de condition: N (négatif), Z (zéro), C (report), V (débordement)
- Conditions de branchement:

Entiers non signés		
Code	Signification	Codes de condition
eq	=	Z
ne	≠	¬Z
hs	≥	C
hi	>	C ∧ ¬Z
ls	≤	¬C ∨ Z
lo	<	¬C

Entiers signés		
Code	Signification	Codes de condition
eq	=	Z
ne	≠	¬Z
ge	≥	N = V
gt	>	¬Z ∧ (N = V)
le	≤	Z ∨ (N ≠ V)
lt	<	N ≠ V
vs	débordement	V
vc	pas de débordement	¬V
mi	négatif	N
pl	non négatif	¬N

Branchement.

- Instructions de branchement, où j est une valeur immédiate de 6 bits:

Code d'op.	Syntaxe	Effet	Exemple
b.	b.cond etiq	branche à etiq : si <i>cond</i>	b.eq main100
b	b etiq	branche à etiq :	b main100
cbz	cbz rd, etiq	branche à etiq : si $r_d = 0$	cbz x19 main100
cbnz	cbnz rd, etiq	branche à etiq : si $r_d \neq 0$	cbnz x19 main100
tbz	tbz rd, j, etiq	branche à etiq : si $r_d(j) = 0$	tbz x19, 1, main100
tbnz	tbnz rd, j, etiq	branche à etiq : si $r_d(j) \neq 0$	tbnz x19, 1, main100
bl	bl etiq	branche à etiq : et $x_{30} \leftarrow pc + 4$	bl printf
blr	blr xd	branche à x_d et $x_{30} \leftarrow pc + 4$	blr x20
br	br xd	branche à x_d	br x20
ret	ret	branche à x_{30} (retour de sous-prog.)	ret

Adressage.

- Modes d'adressages, où k est une valeur immédiate de 7 bits:

Nom	Syntaxe	Adresse	Effet	Exemple
adresse d'une étiquette	adr xd, etiq	—	$x_d \leftarrow$ adresse de etiq :	adr x19, main100
indirect par registre	[xd]	x_d	—	[x20]
indirect par registre indexé	[xd, xn]	$x_d + x_n$	—	[x20, x21]
	[xd, k]	$x_d + k$	—	[x20, 1]
	[xd, xn, decal k]	$x_d + (x_n \text{ decal } k)$	—	[x20, x21, lsl 1]
ind. par reg. indexé pré-inc.	[xd, k]!	$x_d + k$	$x_d \leftarrow x_d + k$ avant calcul	[x20, 1]!
ind. par reg. indexé post-inc.	[xd], k	$x_d + k$	$x_d \leftarrow x_d + k$ après calcul	[x20], 1
relatif	etiq	adresse de etiq	—	main100

Logique et manipulation de bits.

- Les instructions **lsl**, **lsr**, **asr** et **ror** possèdent également une variante de 32 bits utilisant les registres w_d , w_n et w_m (dans ce cas, les 32 bits de poids fort sont mis à 0)
- Instructions, où i est une valeur immédiate de 12 bits et j est une valeur immédiate de 6 bits:

Code d'op.	Syntaxe	Effet	Exemple
mvn	mvn rd, rn	$r_d \leftarrow \neg r_n$	mvn x19, x20
and	and rd, rn, rm	$r_d \leftarrow r_n \wedge r_m$	and x19, x20, x21
	and rd, rn, i	$r_d \leftarrow r_n \wedge i$	and x19, x20, 4
	and rd, rn, rm, decal j	$r_d \leftarrow r_n \wedge (r_m \text{ decal } j)$	and x19, x20, x21, lsl 1
orr	orr rd, rn, rm	$r_d \leftarrow r_n \vee r_m$	orr x19, x20, x21
	orr rd, rn, i	$r_d \leftarrow r_n \vee i$	orr x19, x20, 4
	orr rd, rn, rm, decal j	$r_d \leftarrow r_n \vee (r_m \text{ decal } j)$	orr x19, x20, x21, lsl 1
eor	eor rd, rn, rm	$r_d \leftarrow r_n \oplus r_m$	eor x19, x20, x21
	eor rd, rn, i	$r_d \leftarrow r_n \oplus i$	eor x19, x20, 4
	eor rd, rn, rm, decal j	$r_d \leftarrow r_n \oplus (r_m \text{ decal } j)$	eor x19, x20, x21, lsl 1
bic	bic rd, rn, rm	$r_d \leftarrow r_n \wedge \neg r_m$	bic x19, x20, x21
	bic rd, rn, i	$r_d \leftarrow r_n \wedge \neg i$	bic x19, x20, 4
	bic rd, rn, rm, decal j	$r_d \leftarrow r_n \wedge \neg (r_m \text{ decal } j)$	bic x19, x20, x21, lsl 1
lsl	lsl xd, xn, j	décalage de j bits vers la gauche: $x_d \langle 63, j \rangle \leftarrow x_n \langle 63 - j, 0 \rangle$; $x_d \langle j - 1, 0 \rangle \leftarrow 0$	lsl x19, x20, 1
lsr	lsr xd, xn, j	décalage de j bits vers la droite: $x_d \langle 63 - j, 0 \rangle \leftarrow x_n \langle 63, j \rangle$; $x_d \langle 63, 64 - j \rangle \leftarrow 0$	lsr x19, x20, 1
asr	asr xd, xn, j	décalage arithmétique de j bits vers la droite: $x_d \langle 63 - j, 0 \rangle \leftarrow x_n \langle 63, j \rangle$; $x_d \langle 63, 64 - j \rangle \leftarrow x_n \langle 63 \rangle$	asr x19, x20, 1
ror	ror xd, xn, j	décalage circulaire de j bits vers la droite: $x_d \leftarrow x_n \langle j - 1, 0 \rangle x_n \langle 63, j \rangle$	ror x19, xn, 1

Autres instructions.

Code d'op.	Syntaxe	Effet	Exemple
csel	csel rd, rn, rm, cond	si <i>cond</i> : $r_d \leftarrow r_n$, sinon: $r_d \leftarrow r_m$	csel x19, x20, x21, eq

Données statiques.

Segments de données		Données	
Pseudo-instruction	Contenu		
.section ".text"	instructions	.align k	donnée suivante stockée à une adresse divisible par k
.section ".rodata"	données en lecture seule	.skip k	réserve k octets
.section ".data"	données initialisées	.ascii s	chaîne de caractères initialisée à s
.section ".bss"	données non-initialisées	.asciz s	chaîne de caractères initialisée à s suivi du caract. nul
		.byte v	octet initialisé à v
		.hword v	demi-mot initialisé à v
		.word v	mot initialisé à v
		.xword v	double mot initialisé à v
		.single f	nombre en virg. flottante simple précision initialisé à f
		.double f	nombre en virg. flottante double précision initialisé à f

Registres (nombres en virgule flottante).

- ▶ Possède 32 registres double précision (64 bits) de la forme d_n
- ▶ Chaque registre d_n possède un sous-registre simple précision (32 bits) s_n
- ▶ v_n réfère au registre d_n ou s_n
- ▶ Conventions:

Registres	Utilisation
$d_0 - d_7$	registres d'arguments et de retour de sous-programmes
$d_8 - d_{15}$	registres sauvegardés par l'appelé
$d_{16} - d_{31}$	registres sauvegardés par l'appelant

Manipulation et arithmétique (nombres en virgule flottante).

- ▶ Les conditions de branchement sont les mêmes que pour les entiers et sont déterminées à partir de codes de condition mis à jour par **fcmp**

Code d'op.	Syntaxe	Effet	Exemple
ldr	ldr d_n, a	charge un nombre en virgule flottante double précision de l'adresse a vers d_n (8 octets)	ldr $d8, [x19]$
	ldr s_n, a	charge un nombre en virgule flottante simple précision de l'adresse a vers s_n (4 octets)	ldr $s8, [x19]$
str	str d_n, a	stocke un nombre en virgule flottante double précision de d_n vers l'adresse a (8 octets)	str $d8, [x19]$
	str s_n, a	stocke un nombre en virgule flottante simple précision de s_n vers l'adresse a (4 octets)	str $s8, [x19]$
fmov	fmov v_d, v_m	$v_d \leftarrow v_m$	fmov $d8, d9$
	fmov v_d, i	$v_d \leftarrow i$	fmov $d8, 1.5$
fcmp	fcmp v_d, v_m	compare v_d et v_m	fcmp $d8, d9$
	fcmp v_d, i	compare v_d et i	fcmp $d8, 0.0$
fadd	fadd v_d, v_n, v_m	$v_d \leftarrow v_n + v_m$	fadd $d8, d9, d10$
fsub	fsub v_d, v_n, v_m	$v_d \leftarrow v_n - v_m$	fsub $d8, d9, d10$
fmul	fmul v_d, v_n, v_m	$v_d \leftarrow v_n \cdot v_m$	fmul $d8, d9, d10$
fdiv	fdiv v_d, v_n, v_m	$v_d \leftarrow v_n / v_m$	fdiv $d8, d9, d10$
fsqrt	fsqrt v_d, v_n	$v_d \leftarrow \sqrt{v_n}$	fsqrt $d8, d9$
fabs	fabs v_d, v_n	$v_d \leftarrow v_n $	fabs $d8, d9$

Entrées/sorties (haut niveau).

- ▶ Affichage: `printf(&format, val1, val2, ...)`
- ▶ Lecture: `scanf(&format, &var1, &var2, ...)`
- ▶ Spécificateurs de format:

Famille	Format	Type
Nombres sur 32 bits	<code>%d</code>	entier décimal signé
	<code>%u</code>	entier décimal non signé
	<code>%X</code>	entier hexadécimal non signé
	<code>%f</code>	nombre en virgule flottante
Nombres sur 64 bits	<code>%ld</code>	entier décimal signé
	<code>%lu</code>	entier décimal non signé
	<code>%lX</code>	entier hexadécimal non signé
	<code>%lf</code>	nombre en virgule flottante
Caractères	<code>%c</code>	caractère (1 octet)
	<code>%s</code>	chaîne de caractères

Appels système.

- ▶ x_8 : code numérique du service
- ▶ x_0 à x_5 : arguments
- ▶ **svc** θ : appel du service

Débogage avec GDB.

Commande	Effet
Commandes de base	
<code>gdb exec</code>	Charge l'exécutable <code>./exec</code> en mode débogage
<code>break etiq</code>	Ajoute un point d'interruption à l'étiquette <code>etiq:</code>
<code>run</code>	Début l'exécution en mode débogage
<code>continue</code>	Continue l'exécution jusqu'au prochain point d'interruption
<code>stepi</code>	Exécute la prochaine instruction
<code>nexti</code>	Exécute la prochaine instruction (sans entrer dans les sous-programmes)
<code>info reg</code>	Affiche le contenu des registres
<code>x &etiq</code>	Affiche le contenu de la mémoire à l'adresse associée à l'étiquette <code>etiq:</code>
<code>quit</code>	Quitter le débogueur
Commandes avancées	
<code>run < fichier</code>	Début l'exécution en mode débogage avec l'entrée contenue dans <code>fichier</code>
<code>p/s \$xd</code>	Affiche le contenu du registre dans le format <code>s</code> parmi l'un de ces choix: <u>u</u> = entier non signé, <u>d</u> = entier signé, <u>x</u> = valeur hexadécimale, <u>t</u> = valeur binaire, <u>f</u> = nombre en virgule flottante, <u>c</u> = caractère. Par exemple, <code>p/t \$x19</code> affiche le contenu du registre <code>x19</code> en binaire
<code>p/s var</code>	Affiche le contenu de la variable <code>var</code> dans le format <code>s</code>
<code>set var = val</code>	Assigne la valeur <code>val</code> à <code>var</code> ; ce-dernier peut être un registre <code>\$xd</code> ou une variable
<code>x 0xABCDFE</code>	Affiche le contenu de la mémoire à l'adresse hexadécimale <code>ABCDEF</code>
<code>x/nsu adr</code>	Affiche le contenu de <code>n</code> unités de mémoire à partir de l'adresse <code>adr</code> dans le format <code>s</code> . L'unité de mémoire est défini par l'un des choix suivants de <code>u</code> : <u>b</u> = octet, <u>h</u> = demi-mot, <u>w</u> = mot, <u>g</u> = double mot. Par exemple, <code>x/10ug &tab</code> affiche les 10 premiers éléments de 64 bits non signés d'un tableau <code>tab</code>