Registres. **[11 janvier 2024]**

— Chaque registre $\mathtt{x}_n$ possède 64 bits: $b_{63}b_{62}\cdots b_1b_0$

— Notation: $\mathtt{x}_n\langle i\rangle \coloneqq b_i$, $\mathtt{x}_n\langle i,j\rangle \coloneqq b_ib_{i-1}\cdots b_j$, $\mathtt{r}_n$ réfère au registre $\mathtt{x}_n$ ou $\mathtt{w}_n$

— Chaque sous-registre $\mathtt{w}_n$ possède 32 bits et correspond à $\mathtt{x}_n\langle 31,0\rangle$

— Le compteur d'instruction `pc` n'est pas accessible

— Conventions:

Registres	Nom	Utilisation
$\mathtt{x}_0-\mathtt{x}_7$	—	registres d'arguments et de retour de sous-programmes
$\mathtt{x}_8$	`xr`	registre pour retourner l'adresse d'une structure
$\mathtt{x}_9-\mathtt{x}_{15}$	—	registres temporaires sauvegardés par l'appelant
$\mathtt{x}_{16}-\mathtt{x}_{17}$	$\mathtt{ip}_0-\mathtt{ip}_1$	registres temporaires intra-procéduraux
$\mathtt{x}_{18}$	`pr`	registre temporaire pouvant être réservé par le système
$\mathtt{x}_{19}-\mathtt{x}_{28}$	—	registres temporaires sauvegardés par l'appelé
$\mathtt{x}_{29}$	`fp`	pointeur vers l'ancien sommet de pile (*frame pointer*)
$\mathtt{x}_{30}$	`lr`	registre d'adresse de retour (*link register*)
$\mathtt{x}_{zr}$	`sp`	registre contenant la valeur 0, ou pointeur de pile (*stack pointer*)

Arithmétique (entiers).

— Les codes de condition sont modifiés par `cmp`, `adds`, `adcs`, `subs`, `sbcs` et `negs`

— À cette différence près, `adds`, `adcs`, `subs`, `sbcs` et `negs` se comportent respectivement comme `add`, `adc`, `sub`, `sbc` 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 $\mathtt{r}_d$ et $\mathtt{r}_m$	`cmp x19, x21`
	`cmp rd, i`	compare $\mathtt{r}_d$ et i	`cmp x19, 42`
	`cmp rd, rm, decal j`	compare $\mathtt{r}_d$ et $\mathtt{r}_m$ *decal* j	`cmp x19, x21, lsl 1`
`add`	`add rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n + \mathtt{r}_m$	`add x19, x20, x21`
	`add rd, rn, i`	$\mathtt{r}_d \leftarrow \mathtt{r}_n + i$	`add x19, x20, 42`
	`add rd, rn, rm, decal j`	$\mathtt{r}_d \leftarrow \mathtt{r}_n + (\mathtt{r}_m$ *decal* $j)$	`add x19, x20, x21, lsl 1`
`adc`	`adc rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n + \mathtt{r}_m + \mathtt{C}$	`adc x19, x20, x21`
`sub`	`sub rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n - \mathtt{r}_m$	`sub x19, x20, x21`
	`sub rd, rn, i`	$\mathtt{r}_d \leftarrow \mathtt{r}_n - i$	`sub x19, x20, 42`
	`sub rd, rn, rm, decal j`	$\mathtt{r}_d \leftarrow \mathtt{r}_n - (\mathtt{r}_m$ *decal* $j)$	`sub x19, x20, x21, lsl 1`
`sbc`	`sbc rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n - \mathtt{r}_m - 1 + \mathtt{C}$	`sbc x19, x20, x21`
`neg`	`neg rd, rm`	$\mathtt{r}_d \leftarrow -\mathtt{r}_m$	`neg x19, x21`
	`neg rd, rm, decal j`	$\mathtt{r}_d \leftarrow -(\mathtt{r}_m$ *decal* $j)$	`neg x19, x21, lsl 1`
`mul`	`mul rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n \cdot \mathtt{r}_m$	`mul x19, x20, x21`
`udiv`	`udiv rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n \div \mathtt{r}_m$ (non signé)	`udiv x19, x20, x21`
`sdiv`	`sdiv rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n \div \mathtt{r}_m$ (signé)	`sdiv x19, x20, x21`
`madd`	`madd rd, rn, rm, ra`	$\mathtt{r}_d \leftarrow \mathtt{r}_a + (\mathtt{r}_n \cdot \mathtt{r}_m)$	`madd x19, x20, x21, x22`
`msub`	`msub rd, rn, rm, ra`	$\mathtt{r}_d \leftarrow \mathtt{r}_a - (\mathtt{r}_n \cdot \mathtt{r}_m)$	`msub x19, x20, x21, x22`

Accès mémoire.

— `ldrsw`, `ldrsh` et `ldrsb` se comportent respectivement comme `ldr` (4 octets), `ldrh` et `ldrb` à l'exception du fait qu'ils effectuent un chargement dans $\texttt{x}_\texttt{d}$ 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`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{m}$	`mov x19, x21`
	`mov rd, i`	$\texttt{r}_\texttt{d} \leftarrow i$	`mov x19, 42`
`ldr`	`ldr xd, a`	charge 8 octets: $\texttt{x}_\texttt{d}\langle 63,0\rangle \leftarrow \text{mem}_8[a]$	`ldr x19, [x20]`
	`ldr wd, a`	charge 4 octets: $\texttt{x}_\texttt{d}\langle 31,0\rangle \leftarrow \text{mem}_4[a]$; $\texttt{x}_\texttt{d}\langle 63,32\rangle \leftarrow 0$	`ldr w19, [x20]`
`ldrh`	`ldrh wd, a`	charge 2 octets: $\texttt{x}_\texttt{d}\langle 15,0\rangle \leftarrow \text{mem}_2[a]$; $\texttt{x}_\texttt{d}\langle 63,16\rangle \leftarrow 0$	`ldrh w19, [x20]`
`ldrb`	`ldrb wd, a`	charge 1 octet: $\texttt{x}_\texttt{d}\langle 7,0\rangle \leftarrow \text{mem}_1[a]$; $\texttt{x}_\texttt{d}\langle 63,8\rangle \leftarrow 0$	`ldrb w19, [x20]`
`str`	`str xd, a`	stocke 8 octets: $\text{mem}_8[a] \leftarrow \texttt{x}_\texttt{d}\langle 63,0\rangle$	`str x19, [x20]`
	`str wd, a`	stocke 4 octets: $\text{mem}_4[a] \leftarrow \texttt{x}_\texttt{d}\langle 31,0\rangle$	`str w19, [x20]`
`strh`	`strh wd, a`	stocke 2 octets: $\text{mem}_2[a] \leftarrow \texttt{x}_\texttt{d}\langle 15,0\rangle$	`str w19, [x20]`
`strb`	`strb wd, a`	stocke 1 octet: $\text{mem}_1[a] \leftarrow \texttt{x}_\texttt{d}\langle 7,0\rangle$	`strb w19, [x20]`
`ldp`	`ldp xd, xn, a`	charge 16 octets: $\texttt{x}_\texttt{d}\langle 63,0\rangle \leftarrow \text{mem}_8[a]$, $\texttt{x}_\texttt{n}\langle 63,0\rangle \leftarrow \text{mem}_8[a+8]$	`ldp x19, x20, [sp]`
`stp`	`stp xd, xn, a`	stocke 16 octets: $\text{mem}_8[a] \leftarrow \texttt{x}_\texttt{d}\langle 63,0\rangle$, $\text{mem}_8[a+8] \leftarrow \texttt{x}_\texttt{n}\langle 63,0\rangle$	`stp x19, x20, [sp]`

Conditions de branchement.

— Codes de condition: `N` (négatif), `Z` (zéro), `C` (report), `V` (débordement)

— `C` indique aussi l'*absence* d'emprunt lors d'une soustraction

— Conditions de branchement:

Entiers non signés

Code	Signification	Codes de condition
`eq`	$=$	$\texttt{Z}$
`ne`	$\neq$	$\neg\texttt{Z}$
`hs`	$\geq$	$\texttt{C}$
`hi`	$>$	$\texttt{C} \wedge \neg\texttt{Z}$
`ls`	$\leq$	$\neg\texttt{C} \vee \texttt{Z}$
`lo`	$<$	$\neg\texttt{C}$

Entiers signés

Code	Signification	Codes de condition
`eq`	$=$	$\texttt{Z}$
`ne`	$\neq$	$\neg\texttt{Z}$
`ge`	$\geq$	$\texttt{N} = \texttt{V}$
`gt`	$>$	$\neg\texttt{Z} \wedge (\texttt{N} = \texttt{V})$
`le`	$\leq$	$\texttt{Z} \vee (\texttt{N} \neq \texttt{V})$
`lt`	$<$	$\texttt{N} \neq \texttt{V}$
`vs`	débordement	$\texttt{V}$
`vc`	pas de débordement	$\neg\texttt{V}$
`mi`	négatif	$\texttt{N}$
`pl`	non négatif	$\neg\texttt{N}$

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 *cond*	`b.eq main100`
`b`	`b etiq`	branche à `etiq:`	`b main100`
`cbz`	`cbz rd, etiq`	branche à `etiq:` si $\texttt{r}_\texttt{d} = 0$	`cbz x19 main100`
`cbnz`	`cbnz rd, etiq`	branche à `etiq:` si $\texttt{r}_\texttt{d} \neq 0$	`cbnz x19 main100`
`tbz`	`tbz rd, j, etiq`	branche à `etiq:` si $\texttt{r}_\texttt{d}\langle j\rangle = 0$	`tbz x19, 1, main100`
`tbnz`	`tbnz rd, j, etiq`	branche à `etiq:` si $\texttt{r}_\texttt{d}\langle j\rangle \neq 0$	`tbnz x19, 1, main100`
`bl`	`bl etiq`	branche à `etiq:` et $\texttt{x}_{30} \leftarrow \texttt{pc} + 4$	`bl printf`
`blr`	`blr xd`	branche à $\texttt{x}_\texttt{d}$ et $\texttt{x}_{30} \leftarrow \texttt{pc} + 4$	`blr x20`
`br`	`br xd`	branche à $\texttt{x}_\texttt{d}$	`br x20`
`ret`	`ret`	branche à $\texttt{x}_{30}$ (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]`	$x_d + (x_n \; decal \; k)$	—	`[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	$x_d \leftarrow x_d + k$ après calcul	`[x20], 1`
relatif	`etiq`	adresse de `etiq`	—	`main100`

Autres instructions.

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

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`	$r_d \leftarrow r_n \wedge (r_m \; decal \; j)$	`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`	$r_d \leftarrow r_n \vee (r_m \; decal \; j)$	`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`	$r_d \leftarrow r_n \oplus (r_m \; decal \; j)$	`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`	$r_d \leftarrow r_n \wedge \neg(r_m \; decal \; j)$	`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`

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 dn, a`	charge un nombre en virgule flottante double précision de l'adresse a vers d_n (8 octets)	`ldr d8, [x19]`
	`ldr sn, a`	charge un nombre en virgule flottante simple précision de l'adresse a vers s_n (4 octets)	`ldr s8, [x19]`
`str`	`str dn, a`	stocke un nombre en virgule flottante double précision de d_n vers l'adresse a (8 octets)	`str d8, [x19]`
	`str sn, a`	stocke un nombre en virgule flottante simple précision de s_n vers l'adresse a (4 octets)	`str s8, [x19]`
`fmov`	`fmov vd, vm`	$v_d \leftarrow v_m$	`fmov d8, d9`
	`fmov vd, i`	$v_d \leftarrow i$	`fmov d8, 1.5`
`fcmp`	`fcmp vd, vm`	compare v_d et v_m	`fcmp d8, d9`
	`fcmp vd, i`	compare v_d et i	`fcmp d8, 0.0`
`fadd`	`fadd vd, vn, vm`	$v_d \leftarrow v_n + v_m$	`fadd d8, d9, d10`
`fsub`	`fsub vd, vn, vm`	$v_d \leftarrow v_n - v_m$	`fsub d8, d9, d10`
`fmul`	`fmul vd, vn, vm`	$v_d \leftarrow v_n \cdot v_m$	`fmul d8, d9, d10`
`fdiv`	`fdiv vd, vn, vm`	$v_d \leftarrow v_n / v_m$	`fdiv d8, d9, d10`
`fsqrt`	`fsqrt vd, vn`	$v_d \leftarrow \sqrt{v_n}$	`fsqrt d8, d9`
`fabs`	`fabs vd, vn`	$v_d \leftarrow \lvert v_n \rvert$	`fabs d8, d9`

`ucvtf`	`ucvtf vd, rn`	convertit l'entier non signé dans r_n vers un nombre en virgule flottante dans v_d (selon le mode d'approximation configuré dans le registre de contrôle FPCR)	`ucvtf d8, x19`
			`ucvtf d8, w19`
			`ucvtf s8, x19`
			`ucvtf s8, w19`

`scvtf`	`scvtf vd, rn`	convertit l'entier signé dans r_n vers un nombre en virgule flottante dans v_d (selon le mode d'approximation configuré dans le registre de contrôle FPCR)	`scvtf d8, x19`
			`scvtf d8, w19`
			`scvtf s8, x19`
			`scvtf s8, w19`

`fcvt`	`fcvt vd, vn`	convertit le nombre en virgule flottante dans v_n vers un nombre en virgule flottante d'une autre précision dans v_d	`fcvt d8, s9`

Appels système.

— x_8: code numérique du service

— x_0 à x_5: arguments

— `svc 0`: appel du service

Données statiques.

Segments de données

Pseudo-instruction	Contenu
`.section ".text"`	instructions
`.section ".rodata"`	données en lecture seule
`.section ".data"`	données initialisées
`.section ".bss"`	données non-initialisées

Données

`.align`	k	donnée suivante stockée à une adresse divisible par k
`.skip`	k	réserve k octets
`.ascii`	s	chaîne de caractères initialisée à s
`.asciz`	s	chaîne de caractères initialisée à s suivi du carac. 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

Entrées/sorties (haut niveau).

— Affichage: `printf(&format, val1, val2, ...)`

— Lecture: `scanf(&format, &var1, &var2, ...)`

— Spécificateurs de format:

Famille	Format	Type
Nombres sur 64 bits	`%ld`	entier décimal signé
	`%lu`	entier décimal non signé
	`%lX`	entier hexadécimal non signé
	`%lf`	nombre en virgule flottante
Nombres sur 32 bits	`%d`	entier décimal signé
	`%u`	entier décimal non signé
	`%X`	entier hexadécimal non signé
	`%f`	nombre en virgule flottante
Nombres sur 16 bits	`%hd`	entier décimal signé
	`%hu`	entier décimal non signé
	`%hX`	entier hexadécimal non signé
Caractères	`%c`	caractère (1 octet)
	`%s`	chaîne de caractères

Débogage avec GDB.

Commande	Effet
Commandes de base	
`gdb exec`	Charge l'exécutable `./exec` en mode débogage
`break etiq`	Ajoute un point d'interruption à l'étiquette `etiq:`
`run`	Débute l'exécution en mode débogage
`continue`	Continue l'exécution jusqu'au prochain point d'interruption
`stepi`	Exécute la prochaine instruction
`nexti`	Exécute la prochaine instruction (sans entrer dans les sous-programmes)
`info reg`	Affiche le contenu des registres
`x &etiq`	Affiche le contenu de la mémoire à l'adresse associée à l'étiquette `etiq:`
`quit`	Quitter le débogueur
Commandes avancées	
`run < fichier`	Débute l'exécution en mode débogage avec l'entrée contenue dans `fichier`
`p/s $xd`	Affiche le contenu du registre dans le format `s` parmi l'un de ces choix: `u` = entier non signé, `d` = entier signé, `x` = valeur hexadécimale, `t` = valeur binaire, `f` = nombre en virgule flottante, `c` = caractère. Par exemple, `p/t $x19` affiche le contenu du registre x_{19} en binaire
`p/s var`	Affiche le contenu de la variable `var` dans le format `s`
`set var = val`	Assigne la valeur `val` à `var`; ce-dernier peut être un registre `$xd` ou une variable
`x 0xABCDFE`	Affiche le contenu de la mémoire à l'adresse hexadécimale `ABCDEF`
`x/nsu adr`	Affiche le contenu de `n` unités de mémoire à partir de l'adresse `adr` dans le format `s`. L'unité de mémoire est défini par l'un des choix suivants de `u`: `b` = octet, `h` = demi-mot, `w` = mot, `g` = double mot. Par exemple, `x/10ug &tab` affiche les 10 premiers éléments de 64 bits non signés d'un tableau `tab`