Jeu d'instructions Assembleur PowerPc

Notations

D	le déplacement
CR	le Registre de Condition
Rx	registre entre r0 et r31
	(RT pour "Target", RS pour "Source", RA et RB pour des opérandes)
(Rx)	le contenu du registre Rx
$(RA\|0)$	si RA=0, alors 0, sinon le contenu de RA
V, SI	quantité immédiate signée sur 16 bits
UI	quantité immédiate non signée sur 16 bits
$rep(N, B)$	champ formé de N répétitions du bit B
$mem(A, N)$	N octets en mémoire à partir de l'adresse A
$A\|\|B$	Le champ obtenu par concaténation de A et B
$exts(V)$	la valeur V avec extension de signe sur 32 bits
$A\{i \dots j\}$	les bits d'indice i à j de A
$A \leftarrow B$	affectation

Mouvements de données

Par octet

Chargement octet et zéro

`lbz RT,D(RA)` $RT \leftarrow rep(24, 0)||mem((RA|0) + D, 1)$

Chargement octet et zéro avec mise-à-jour

`lbzu RT,D(RA)` $RT \leftarrow rep(24, 0)||mem((RA) + D, 1)$;
$RA \leftarrow (RA) + D$

Chargement indexé octet et zéro

`lbzx RT,RA,RB` $RT \leftarrow rep(24, 0)||mem((RA|0) + (RB), 1)$

Chargement indexé octet et zéro avec mise-à-jour

`lbzux RT,RA,RB` $RT \leftarrow rep(24, 0)||mem((RA) + (RB), 1)$
$RA \leftarrow (RA) + (RB)$

Rangement octet

`stb RS,D(RA)` $mem((RA|0) + D, 1) \leftarrow (RS)24..31$

Rangement octet avec mise-à-jour

`stbu RS,D(RA)` $mem((RA) + D, 1) \leftarrow (RS)24..31$
$RA \leftarrow (RA) + D$

Rangement indexé octet

`stbx RS,RA,RB` $mem((RA|0) + (RB), 1) \leftarrow (RS)24..31$

Rangement indexé octet avec mise-à-jour

`stbux RS,RA,RB` $mem((RA) + (RB), 1) \leftarrow (RS)24..31$
$RA \leftarrow (RA) + (RB)$

Par demi-mot

Chargement demi-mot et zéro

`lhz RT,D(RA)` $RT \leftarrow rep(16,0)||mem((RA|0)+D,2)$

Chargement demi-mot et zéro avec mise-à-jour

`lhzu RT,D(RA)` $RT \leftarrow rep(16,0)||mem((RA)+D,2)$
$RA \leftarrow (RA)+D$

Chargement indexé demi-mot et zéro

`lhzx RT,RA,RB` $RT \leftarrow rep(16,0)||mem((RA|0)+(RB),2)$

Chargement indexé demi-mot et zéro avec mise-à-jour

`lhzux RT,RA,RB` $RT \leftarrow rep(16,0)||mem((RA)+(RB),2)$
$RA \leftarrow (RA)+(RB)$

Chargement algébrique demi-mot

`lha RT,D(RA)` $RT \leftarrow exts(mem((RA|0)+D,2))$

Chargement algébrique demi-mot avec mise-à-jour

`lhau RT,D(RA)` $RT \leftarrow exts(mem((RA)+D,2))$
$RA \leftarrow (RA)+D$

Chargement algébrique indexé demi-mot

`lhax RT,RA,RB` $RT \leftarrow exts(mem((RA|0)+(RB),2))$

Chargement algébrique indexé demi-mot avec mise-à-jour

`lhaux RT,RA,RB` $RT \leftarrow exts(mem((RA)+(RB),2))$
$RA \leftarrow (RA)+(RB)$

Rangement demi-mot

`sth RS,D(RA)` $mem((RA|0)+D),2) \leftarrow (RS)16..31$

Rangement demi-mot avec mise-à-jour

`sthu RS,D(RA)` $mem((RA)+D),2) \leftarrow (RS)16..31$
$RA \leftarrow (RA)+D$

Rangement indexé demi-mot

`sthx RS,RA,RB` $mem((RA|0)+(RB),2) \leftarrow (RS)16..31$

Rangement indexé demi-mot avec mise-à-jour

`sthux RS,RA,RB` $mem((RA)+(RB),2) \leftarrow (RS)16..31$
$RA \leftarrow (RA)+(RB)$

Par mot

Chargement mot

`lwz RT,D(RA)` $RT \leftarrow mem((RA|0)+D,4)$

Chargement mot avec mise-à-jour

`lwzu RT,D(RA)` $RT \leftarrow mem((RA)+D,4)$
$RA \leftarrow (RA)+D$

Chargement indexé mot

`lwzx RT,RA,RB` $RT \leftarrow mem((RA|0)+(RB),4)$

Chargement indexé mot avec mise-à-jour

`lwzux RT,RA,RB` $RT \leftarrow mem((RA)+(RB),4)$
$RA \leftarrow (RA)+(RB)$

Rangement mot

`stw RS,D(RA)` $mem((RA|0)+D,4) \leftarrow (RS)$

Rangement mot avec mise-à-jour

`stwu RS,D(RA)` $mem((RA)+D,4) \leftarrow (RS)$
$RA \leftarrow (RA)+D$

Rangement indexé mot

`stwx RS,RA,RB` $mem((RA|0)+(RB),4) \leftarrow (RS)$

Rangement indexé mot avec mise-à-jour

`stwux RS,RA,RB` $mem((RA)+(RB),4) \leftarrow (RS)$
$RA \leftarrow (RA)+(RB)$

Chargement de constantes

Chargement immédiat

`li RT,V` $RT \leftarrow exts(V)$

Chargement immédiat en partie haute

`lis RT,V` $RT(0..15) \leftarrow V$

En fait ces deux instructions sont des mnémoniques étendus qui représentent `addi RT,0,V` et `addis RT,0,V`

Mouvement de registre à registre

Chargement registre général

`mr RT,RS` $RT \leftarrow (RS)$
C'est un mnémonique étendu pour `or RT,RS,RS`

Chargement à partir du registre de lien (move from link register)

`mflr RT` $RT \leftarrow (LR)$

Rangement dans registre de lien (move to link register)

`mtlr RS` $LR \leftarrow (RS)$

Branchements et conditions

Branchement inconditionnel (goto ou jump)

`b adr`

Branchement inconditionnel avec lien (adresse suivante dans LR, Link Register)

`bl adr`

Branchement conditionnel

`bxx CR,adr`	Avec *xx* défini ci-dessous

Branchement conditionnel avec lien

`bxxl CR,adr`	Avec *xx* défini ci-dessous

Branchement conditionnel au registre de lien (LR)

`bxxlr CR`	Avec *xx* défini ci-dessous

Branchement conditionnel au registre de lien avec lien

`bxxlrl CR`	Avec *xx* défini ci-dessous

Code *xx* de branchement et signification

`lt`	*less than* $:<$
`eq`	*equal to* $:=$
`gt`	*greater than* $:>$
`le`	*less than or equal to* (*not greater than*) $:<=$
`ge`	*greater than or equal to* (*not less than*) $:>=$
`ne`	*not equal to* $:!=$
`nl`	*not less than* $:>=$
`ng`	*not greater* $:<=$
`z`	*zero*
`nz`	*not zero*

Les prédictions de branchements : les deux suffixes + et − peuvent être ajoutés à un code mnémonique de branchement conditionnel :

1. + indique que la prédiction de branchement est celle prévue
2. − indique que la prédiction de branchement n'est pas celle prévue

Opérations arithmétiques et logiques

La notation `[.]` signifie que le mnémonique peut être suivi d'un point. Dans ce cas, l'opération met à jour le sous-registre de condition CR0.

Opérations arithmétiques

`addi RT,RA,SI`	$RT \leftarrow (RA\|0) + exts(SI)$
`addis RT,RA,SI`	$RT \leftarrow (RA\|0) + (SI)\|\|rep(16,0)$
`add[.] RT,RA,RB`	$RT \leftarrow (RA) + (RB)$
`subf[.] RT,RA,RB`	$RT \leftarrow (RB) - (RA)$
`neg[.] RT,RA`	$RT \leftarrow -(RA)$
`mulli RT,RA,SI`	$RT \leftarrow (RA) * exts(SI)$
`mullw[.] RT,RA,RB`	$RT \leftarrow (RA) * (RB)$
`divw[.] RT,RA,RB`	$RT \leftarrow (RA)/(RB)$

Note : la séquence suivante permet de calculer le reste d'une division entière :


```
divw  RT,RA,RB
mullw RT,RT,RB
subf  RT,RT,RA
```

Les comparaisons positionnent les indicateurs d'un sous-registre de condition.

Comparaisons

`cmpwi CR,RA,SI`	comparaison de (RA) et $exts(SI)$, résultat ds CR
`cmpw CR,RA,RB`	comparaison de (RA) et (RB)

Opérations logiques

`andi RA,RS,UI`	$RA \leftarrow (RS)\ and\ rep(16,0)\|\|UI$
`ori RA,RS,UI`	$RA \leftarrow (RS)\ or\ rep(16,0)\|\|UI$
`xori RA,RS,UI`	$RA \leftarrow (RS)\ xor\ rep(16,0)\|\|UI$
`andis RA,RS,UI`	$RA \leftarrow (RS)\ and\ UI\|\|rep(16,0)$
`oris RA,RS,UI`	$RA \leftarrow (RS)\ or\ UI\|\|rep(16,0)$
`xoris RA,RS,UI`	$RA \leftarrow (RS)\ xor\ UI\|\|rep(16,0)$
`and[.] RA,RS,RB`	$RA \leftarrow (RS)\ and\ (RB)$
`or[.] RA,RS,RB`	$RA \leftarrow (RS\ )or\ (RB)$
`xor[.] RA,RS,RB`	$RA \leftarrow (RS)\ xor\ (RB)$
`nand[.] RA,RS,RB`	$RA \leftarrow (RS)\ nand\ (RB)$
`nor[.] RA,RS,RB`	$RA \leftarrow (RS)\ nor\ (RB)$
`eqv[.] RA,RS,RB`	$RA \leftarrow (RS)\ eqv\ (RB)(eqv = notxor)$
`andc[.] RA,RS,RB`	$RA \leftarrow (RS)\ and\ not(RB)$
`orc[.] RA,RS,RB`	$RA \leftarrow (RS)\ or\ not(RB)$

Extension de signe

`extsb[.] RA,RS`	$RA \leftarrow exts(RS24..31)$
`extsh[.] RA,RS`	$RA \leftarrow exts(RS16..31)$

Décalages

`slw[.] RA,RS,RB`	$RA \leftarrow (RS) << (RB)$
`srw[.] RA,RS,RB`	$RA \leftarrow (RS) >> (RB)$
`sraw[.] RA,RS,RB`	$RA \leftarrow exts(RS0..31 - (RB))$ (décalage algébrique)
`srawi[.] RA,RS,SI`	$RA \leftarrow exts(RS0..31 - SI)$