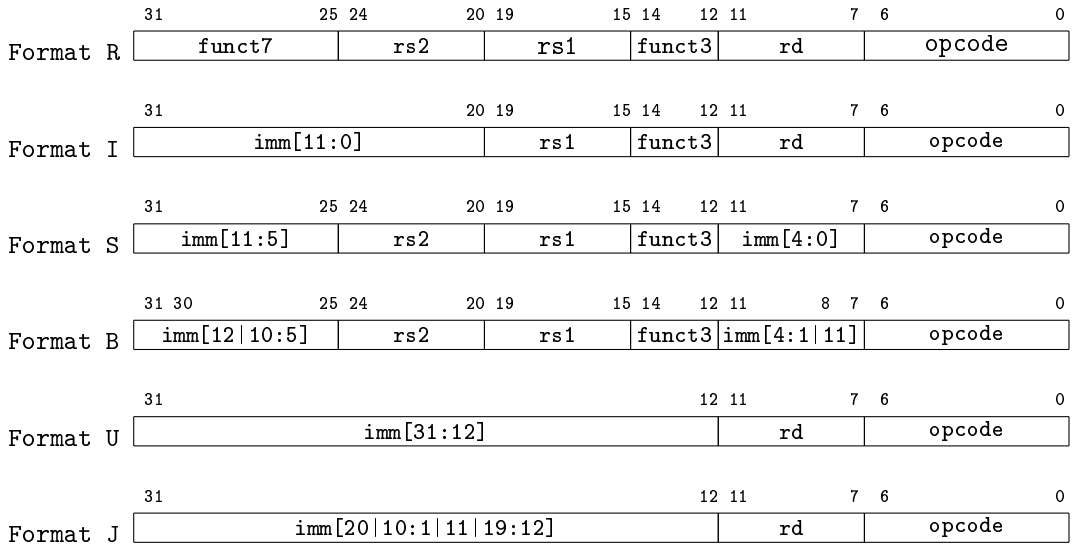
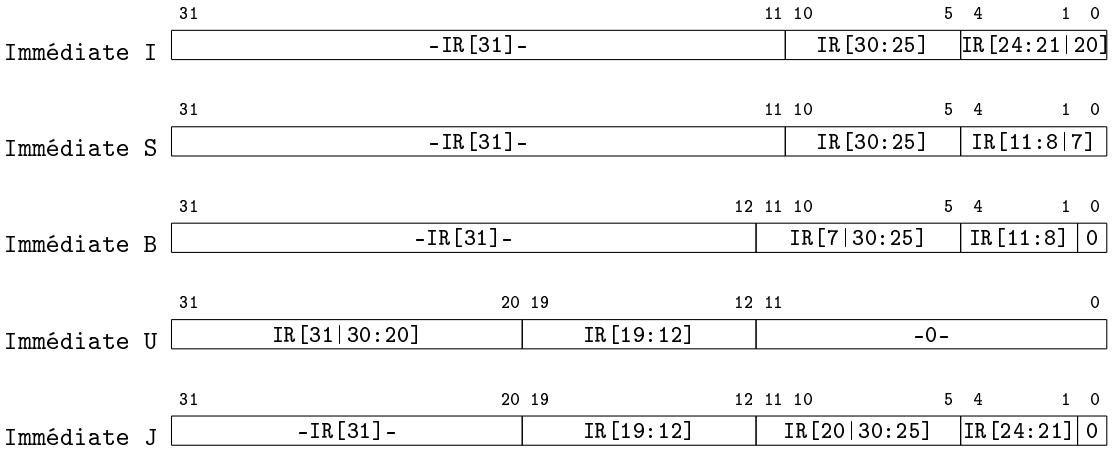
Encodage des instructions rv32im

Les tables suivantes présentent sous forme compacte les 6 formats des instructions rv32im codées sur 32 bits : le format R pour les opérations registre-registre, le format I pour les opérations immédiates courtes et les « loads », le format S pour les « stores », le format B pour les opérations de saut conditionnel, le format U pour les opérations immédiates longues, et le format J pour les opérations de saut inconditionnel.



Construction des constantes immédiates rv32im

Les tables suivantes présentent sous forme compacte les 5 types de constantes immédiates produites à partir des instructions rv32im. Dans le RISC-V les constantes immédiates sont toujours étendues de signe. Certains formats construisent la valeur de la constante à partir de la valeur immédiate contenu dans l'instruction de manière assez peu conventionnelle. On introduit donc une constante intermédiaire (`cst`) dans la notation pour faciliter la description.

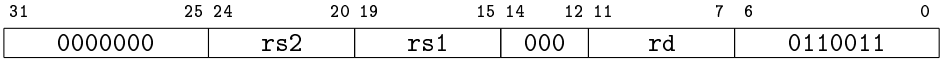


Description des instructions rv32im

— le processeur possède 32 registres de 32 bits chacun, notés `x0` à `x31` ;
— aucune instruction n'utilise de registre implicite ;
— le registre `x0` peut être écrit, mais il vaut néanmoins toujours '0'.

— add —

encodage



action
Addition registre registre signée.

syntaxe
`add rd, rs1, rs2`

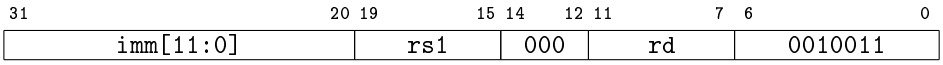
description
Les contenus des registres `rs1` et `rs2` sont ajoutés pour former un résultat sur 32 bits qui est placé dans le registre `rd`.

opération
$\texttt{rd} \leftarrow \texttt{rs1} + \texttt{rs2}$

format R

— addi —

encodage



action
Addition registre immédiat signée.

syntaxe
`addi rd, rs1, imm`

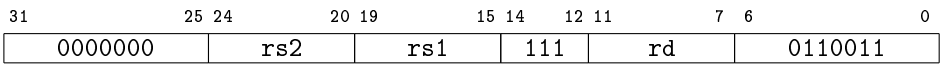
description
Le contenu du registre `rs1` est ajouté à l'immédiat sur 12 bits étendu de signe pour former un résultat sur 32 bits qui est placé dans le registre `rd`.

opération
$\texttt{rd} \leftarrow \texttt{rs1} + (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20})$

format I

— and —

encodage



action
Et bit-à-bit registre registre

syntaxe
`and rd, rs1, rs2`

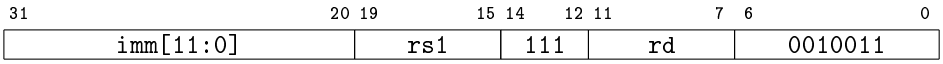
description
Un `et` bit-à-bit est effectué entre les contenus des registres `rs1` et `rs2`. Le résultat est placé dans le registre `rd`.

opération
`rd` ← `rs1` and `rs2`

format R

— andi —

encodage



action
Et bit-à-bit registre immédiat

syntaxe
`andi rd, rs1, imm`

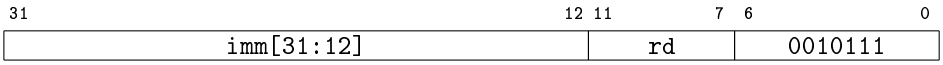
description
La valeur immédiate sur 16 bits subit une extension de zéros. Un `et` bit-à-bit est effectué entre cette valeur étendue et le contenu du registre `rs1` pour former un résultat placé dans le registre `rs2`.

opération
`rd` ← $(\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20})$ and `rs1`

format I

— auipc —

encodage



action
Addition d'un immédiat aux bits de poids fort de `pc`.

syntaxe
`auipc rd, imm`

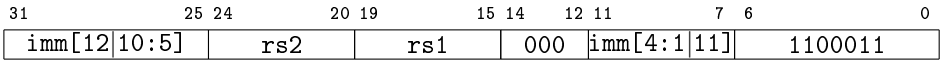
description
La valeur immédiate sur 20 bits décalée à gauche de 12 bits, avec injection de zéros. La constante ainsi obtenue est ajoutée à `pc` et le résultat est stocké dans `rd`.

opération
`rd` ← $(\texttt{IR}_{31\ldots12} \parallel 0^{12}) + \texttt{pc}$

format U

— beq —

encodage



action
Branchement si registre égal registre

syntaxe
`beq rs1, rs2, label`

description
Les contenus des registres `rs1` et `rs2` sont comparés. S'ils sont égaux, le programme saute à l'adresse correspondant à l'étiquette. La constante `cst` construite (de manière assez exotique) à partir de l'`imm` represente la distance, en avant ou en arrière, à laquelle il faut sauter. Cette distance est calculée par l'assembleur.

opération

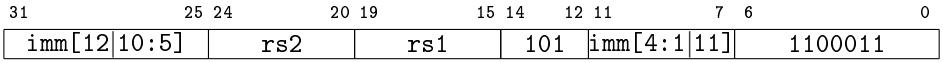
$\texttt{cst} = (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_7 \parallel \texttt{IR}_{30\ldots25} \parallel \texttt{IR}_{11\ldots8} \parallel 0)$

$\texttt{rs1} = \texttt{rs2} \Rightarrow \texttt{pc} \leftarrow \texttt{pc} + \texttt{cst}$

format B

— bge —

encodage



action

Branchement si supérieur ou égal, comparaison signée

syntaxe

`bge rs1, rs2, label`

description

Les valeurs contenues dans les registres `rs1` et `rs2` sont considérés comme signées. Si le contenu du registre `rs1` est supérieur ou égal à celui du registre `rs2`, le programme saute à l'adresse correspondant à l'étiquette. La constante `cst` construite (de manière assez exotique) à partir de l'`imm` represente la distance, en avant ou en arrière, à laquelle il faut sauter. Cette distance est calculée par l'assembleur.

opération

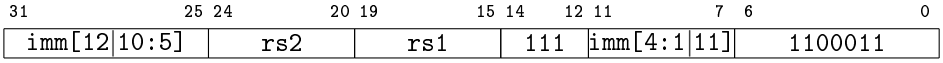
$\texttt{cst} = (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_7 \parallel \texttt{IR}_{30\ldots25} \parallel \texttt{IR}_{11\ldots8} \parallel 0)$

$\texttt{rs1} \geq \texttt{rs2} \Rightarrow \texttt{pc} \leftarrow \texttt{pc} + \texttt{cst}$

format B

— bgeu —

encodage



action

Branchement si supérieur ou égal, comparaison non-signée

syntaxe

`bge rs1, rs2, label`

description

Les valeurs contenues dans les registres `rs1` et `rs2` sont considérés comme non-signées. Si le contenu du registre `rs1` est supérieur ou égal à celui du registre `rs2`, le programme saute à l'adresse correspondant à l'étiquette. La constante `cst` construite (de manière assez exotique) à partir de l'`imm` represente la distance, en avant ou en arrière, à laquelle il faut sauter. Cette distance est calculée par l'assembleur.

opération

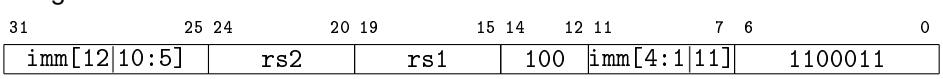
$\texttt{cst} = (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_7 \parallel \texttt{IR}_{30\ldots25} \parallel \texttt{IR}_{11\ldots8} \parallel 0)$

$\texttt{rs1} \overset{usg}{\geq} \texttt{rs2} \Rightarrow \texttt{pc} \leftarrow \texttt{pc} + \texttt{cst}$

format B

— blt —

encodage



action
Branchement si strictement inférieur, comparaison signée

syntaxe
`blt rs1, rs2, label`

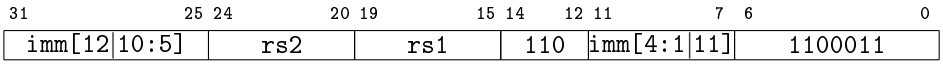
description
Les valeurs contenues dans les registres `rs1` et `rs2` sont considérés comme signées. Si le contenu du registre `rs1` est strictement inférieur à celui du registre `rs2`, le programme saute à l'adresse correspondant à l'étiquette. La constante `cst` construite (de manière assez exotique) à partir de l'`imm` represente la distance, en avant ou en arrière, à laquelle il faut sauter. Cette distance est calculée par l'assembleur.

opération
$\texttt{cst} = (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_7 \parallel \texttt{IR}_{30\ldots25} \parallel \texttt{IR}_{11\ldots8} \parallel 0)$
$\texttt{rs1} < \texttt{rs2} \Rightarrow \texttt{pc} \leftarrow \texttt{pc} + \texttt{cst}$

format B

— bltu —

encodage



action
Branchement si strictement inférieur, comparaison non-signée

syntaxe
`bltu rs1, rs2, label`

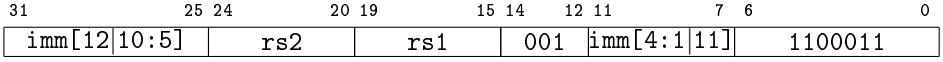
description
Les valeurs contenues dans les registres `rs1` et `rs2` sont considérés comme non-signées. Si le contenu du registre `rs1` est strictement inférieur à celui du registre `rs2`, le programme saute à l'adresse correspondant à l'étiquette. La constante `cst` construite (de manière assez exotique) à partir de l'`imm` represente la distance, en avant ou en arrière, à laquelle il faut sauter. Cette distance est calculée par l'assembleur.

opération
$\texttt{cst} = (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_7 \parallel \texttt{IR}_{30\ldots25} \parallel \texttt{IR}_{11\ldots8} \parallel 0)$
$\texttt{rs1} \overset{usg}{<} \texttt{rs2} \Rightarrow \texttt{pc} \leftarrow \texttt{pc} + \texttt{cst}$

format B

— bne —

encodage



action
Branchement si registre différent de registre

syntaxe
`bne rs1, rs2, label`

description
Les contenus des registres `rs1` et `rs2` sont comparés. S'ils sont différents, le programme saute à l'adresse correspondant à l'étiquette. La constante `cst` construite (de manière assez exotique) à partir de l'`imm` represente la distance, en avant ou en arrière, à laquelle il faut sauter. Cette distance est calculée par l'assembleur.

opération

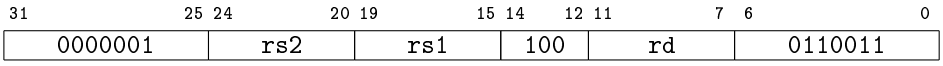
$\texttt{cst} = (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{7} \parallel \texttt{IR}_{30\ldots25} \parallel \texttt{IR}_{11\ldots8} \parallel 0)$

$\texttt{rs1} \neq \texttt{rs2} \Rightarrow \texttt{pc} \leftarrow \texttt{pc} + \texttt{cst}$

format B

— div —

encodage



action

Division entière signée

syntaxe

`div rd, rs1, rs2`

description

Le contenu du registre `rs1` est divisé par le contenu du registre `rs2`, le contenu des deux registres étant considéré comme des nombres en complément à deux (signés). Le quotien resultant de la division est placé dans le registre `rd`.

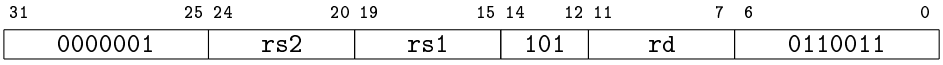
opération

$\texttt{rd} \leftarrow \frac{\texttt{rs1}}{\texttt{rs2}}$

format R

— divu —

encodage



action

Division entière non-signée

syntaxe

`divu rd, rs1, rs2`

description

Le contenu du registre `rs1` est divisé par le contenu du registre `rs2`, le contenu des deux registres étant considéré comme des nombres non-signés. Le quotien resultant de la division est placé dans le registre `rd`.

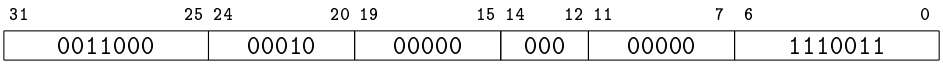
opération

$\texttt{rd} \leftarrow \frac{0 \parallel \texttt{rs1}}{0 \parallel \texttt{rs2}}$

format R

— mret —

encodage



action

Retour d'exception (ou d'interruption)

syntaxe

`mret`

description pour les extensions

Le programme saute à l'adresse stockée dans le registre `mepc`. Le bit `mie` du registre `mstatus` prend la valeur du bit `mpie` de ce même registre, et `mpie` est mis à '1'.

opération

$\texttt{pc} \leftarrow \texttt{mepc}$

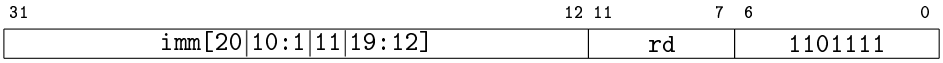
$\texttt{mstatus}_3 \leftarrow \texttt{mstatus}_7$

$\texttt{mstatus}_7 \leftarrow 1$

format X

— jal —

encodage



action

Saut ou appel de fonction inconditionnel immédiat

syntaxe

`jal rd, label`

description

L'adresse de l'instruction suivant le `jal` est sauvée dans le registre `rd`. On effectue un simple saut (sans sauvegarder l'adresse de retour) en choisissant `x0` pour `rd`. Le programme saute inconditionnellement à l'adresse correspondant à l'étiquette. La constante `cst` construite (de manière tout aussi exotique mais cependant différente de celle des branchements) à partir de l'`imm` represente la distance, en avant ou en arrière, à laquelle il faut sauter. Cette distance est calculée par l'assembleur.

opération

$\texttt{rd} \leftarrow \texttt{pc} + 4$

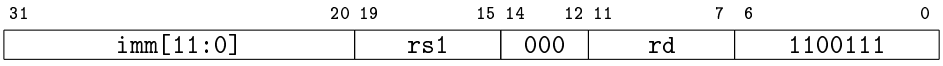
$\texttt{cst} = (\texttt{IR}_{31}^{12} \parallel \texttt{IR}_{19\ldots12} \parallel \texttt{IR}_{20} \parallel \texttt{IR}_{30\ldots25} \parallel \texttt{IR}_{24\ldots21} \parallel 0)$

$\texttt{pc} \leftarrow \texttt{pc} + \texttt{cst}$

format J

— jalr —

encodage



action

Saut ou appel de fonction inconditionnel registre plus immédiat

syntaxe

`jalr rd, imm(rs1)`

description

Le programme saute à l'adresse contenue dans le registre `rs1` auquel la constante sur 12 bits étandue de signe a été ajoutée, puis le bit de poids faible mis à zéro. L'adresse de l'instruction suivant le `jalr` est sauvée dans le registre `rd`. Si `rd` est `x0`, l'instruction est un simple saut.

opération

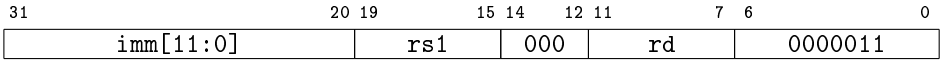
$\texttt{rd} \leftarrow \texttt{pc} + 4$

$\texttt{pc} \leftarrow (\texttt{rs1} + (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}))_{31\ldots1} \parallel 0$

format I

— lb —

encodage



action

Lecture d'un octet signé de la mémoire

syntaxe

`lb rd, imm(rs1)`

description

L'adresse de chargement est la somme de la valeur immédiate sur 12 bits étendue de signe, et du contenu du registre `rs1`. Le contenu de cette adresse subit une extension de signe et est ensuite placé dans le registre `rd`.

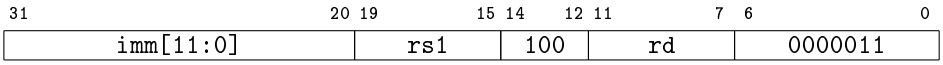
opération

$\texttt{rd} \leftarrow \text{mem}[(\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) + \texttt{rs1}]_{7}^{24} \parallel \text{mem}[(\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) + \texttt{rs1}]_{7\ldots0}$

format I

— lbu —

encodage



action

Lecture d'un octet non-signé de la mémoire

syntaxe

`lbu rd, imm(rs1)`

description

L'adresse de chargement est la somme de la valeur immédiate sur 12 bits étendue de signe, et du contenu du registre `rs1`. Le contenu de cette adresse est étendu avec des zéros et est ensuite placé dans le registre `rd`.

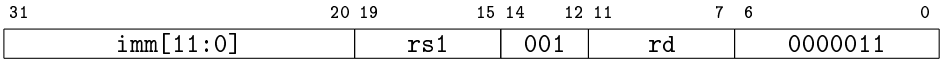
opération

$\texttt{rd} \leftarrow 0^{24} \parallel \text{mem}[(\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) + \texttt{rs1}]_{7\ldots0}$

format I

— lh —

encodage



action

Lecture d'un demi-mot signé de la mémoire

syntaxe

`lh rd, imm(rs1)`

description

L'adresse de chargement est la somme de la valeur immédiate sur 12 bits étendue de signe, et du contenu du registre `rs1`. Le demi-mot contenu à cette adresse subit une extension de signe et est ensuite placé dans le registre `rd`. Attention, le bit de poids faible de l'adresse résultante doit être à zéro.

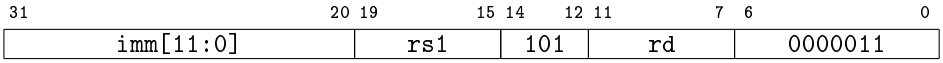
opération

$\texttt{rd} \leftarrow \text{mem}[(\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) + \texttt{rs1}]_{15}^{16} \parallel \text{mem}[(\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) + \texttt{rs1}]_{15\ldots0}$

format I

— lhu —

encodage



action

Lecture d'un demi-mot non-signé de la mémoire

syntaxe

`lhu rd, imm(rs1)`

description

L'adresse de chargement est la somme de la valeur immédiate sur 12 bits étendue de signe, et du contenu du registre `rs1`. Le demi-mot contenu à cette adresse est étendu de zéros et est ensuite placé dans le registre `rd`. Attention, le bit de poids faible de l'adresse résultante doit être à zéro.

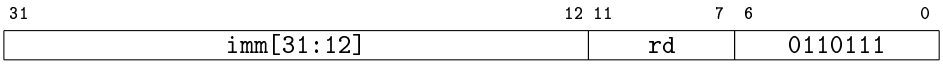
opération

$\texttt{rd} \leftarrow 0^{16} \parallel \text{mem}[(\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) + \texttt{rs1}]_{15\ldots0}$

format I

— lui —

encodage



action

Lecture d'une constante dans les poids forts

syntaxe

`lui rd, imm`

description

La constante immédiate de 20 bits est décalée de 12 bits à gauche, et complétée de zéros. La valeur ainsi obtenue est placée dans `rd`.

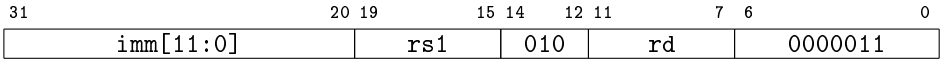
opération

$\texttt{rd} \leftarrow (\texttt{IR}_{31\ldots12} \parallel 0^{12})$

format U

— lw —

encodage



action

Lecture d'un mot de la mémoire

syntaxe

`lw rd, imm(rs1)`

description

L'adresse de chargement est la somme de la valeur immédiate sur 12 bits étendue de signe, et du contenu du registre `rs1`. Le contenu de cette adresse est placé dans le registre `rd`. Attention, les deux bits de poids faible de l'adresse résultante doivent être à zéro.

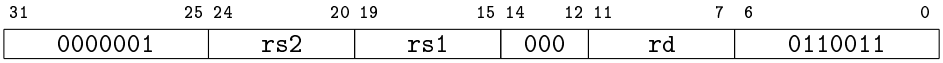
opération

$\texttt{rd} \leftarrow \text{mem}[(\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) + \texttt{rs1}]$

format I

— mul —

encodage



action
Multiplication registre registre, poids faibles

syntaxe
`mul rd, rs1, rs2`

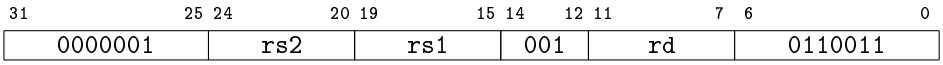
description
Le contenu du registre `rs1` est multiplié par le contenu du registre `rs2`, et les 32 bits de poids faible du résultat de l'opération sont placés dans `rd`.

opération
$\texttt{rd} \leftarrow [\texttt{rs1} \times \texttt{rs2}]_{31...0}$

format R

— mulh —

encodage



action
Multiplication registre registre, opérandes signés, poids forts

syntaxe
`mulh rd, rs1, rs2`

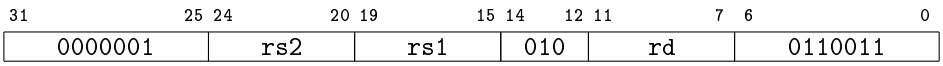
description
Le contenu du registre `rs1` est multiplié par le contenu du registre `rs2`, tous deux considérés comme signés, et les 32 bits de poids fort du résultat de l'opération sont placés dans `rd`.

opération
$\texttt{rd} \leftarrow [\texttt{rs1} \times \texttt{rs2}]_{63...32}$

format R

— mulhsu —

encodage



action
Multiplication registre registre, premier opérande signé, second non signé, poids forts

syntaxe
`mulhsu rd, rs1, rs2`

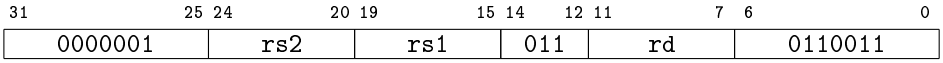
description
Le contenu du registre `rs1` considéré comme signé est multiplié par le contenu du registre `rs2` considéré comme non-signé, et les 32 bits de poids fort du résultat de l'opération sont placés dans `rd`.

opération
$\texttt{rd} \leftarrow [\texttt{rs1} \times (0 \parallel \texttt{rs2})]_{63...32}$

format R

— mulhu —

encodage



action
Multiplication registre registre, opérandes non-signés, poids forts

syntaxe
`mulhu rd, rs1, rs2`

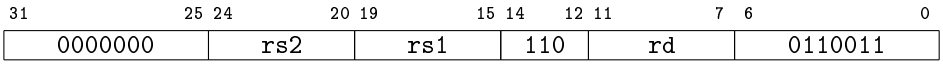
description
Le contenu du registre `rs1` est multiplié par le contenu du registre `rs2`, tous deux considérés comme non-signés, et les 32 bits de poids fort du résultat de l'opération sont placés dans `rd`.

opération
$\texttt{rd} \leftarrow [(0 \parallel \texttt{rs1}) \times (0 \parallel \texttt{rs2})]_{63...32}$

format R

— or —

encodage



action
Ou bit-à-bit registre registre

syntaxe
`or rd, rs1, rs2`

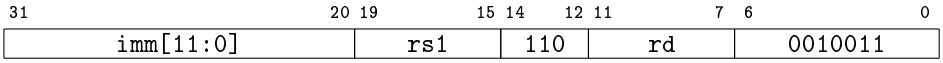
description
Un `ou` bit-à-bit est effectué entre les contenus des registres `rs1` et `rs2`. Le résultat est placé dans le registre `rd`.

opération
$\texttt{rd} \leftarrow \texttt{rs1}$ or $\texttt{rs2}$

format R

— ori —

encodage



action
Ou bit-à-bit registre immédiat

syntaxe
`ori rd, rs1, imm`

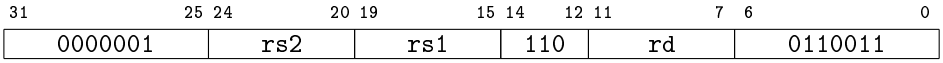
description
La valeur immédiate sur 12 bits est étendue de signe. Un `ou` bit-à-bit est effectué entre cette valeur étendue et le contenu du registre `rs1` pour former un résultat placé dans le registre `rd`.

opération
$\texttt{rd} \leftarrow (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31...20})$ or $\texttt{rs1}$

format I

— rem —

encodage



action

Reste de la division entière signée

syntaxe

`rem rd, rs1, rs2`

description

Le contenu du registre `rs1` est divisé par le contenu du registre `rs2`, le contenu des deux registres étant considéré comme des nombres en complément à deux (signés). Le reste de la division est placé dans le registre `rd`.

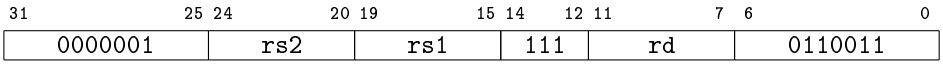
opération

$\texttt{rd} \leftarrow \texttt{rs1} \bmod \texttt{rs2}$

format R

— remu —

encodage



action

Reste de la division entière non-signée

syntaxe

`remu rd, rs1, rs2`

description

Le contenu du registre `rs1` est divisé par le contenu du registre `rs2`, le contenu des deux registres étant considéré comme des nombres non-signés. Le reste de la division est placé dans le registre `rd`.

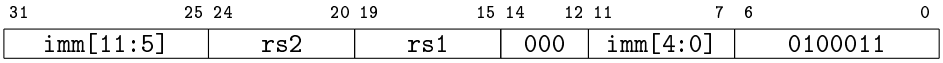
opération

$\texttt{rd} \leftarrow (0 \parallel \texttt{rs1}) \bmod (0 \parallel \texttt{rs2})$

format R

— sb —

encodage



action

Écriture d'un octet en mémoire

syntaxe

`sb rs2, imm(rs1)`

description

La constante `cst` construite à partir de l'`imm` est étendue de signe et sommé avec le contenu du registre `rs1`. L'octet de poids faible du registre `rs2` est écrit à l'adresse ainsi calculée.

opération

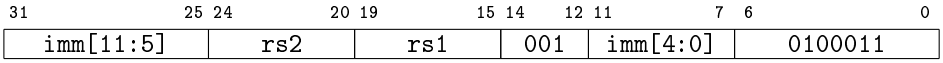
$\texttt{cst} = (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots25} \parallel \texttt{IR}_{11\ldots7})$

$\text{mem}[\texttt{cst} + \texttt{rs1}] \leftarrow \texttt{rs2}_{7\ldots0}$

format S

— sh —

encodage



action

Écriture d'un demi-mot en mémoire

syntaxe

`sh rs2, imm(rs1)`

description

La constante `cst` construite à partir de l'`imm` est étendue de signe et sommée avec le contenu du registre `rs1`. Les deux octets de poids faible du registre `rs2` sont écrit à l'adresse ainsi calculée. Le bit de poids faible de cette adresse doit être à zéro.

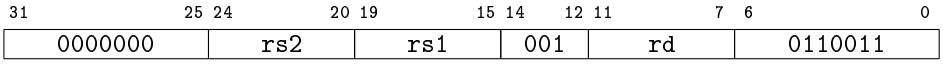
opération

$\texttt{cst} = (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots25} \parallel \texttt{IR}_{11\ldots7})$
$\text{mem}[\texttt{cst} + \texttt{rs1}] \leftarrow \texttt{rs2}_{15\ldots0}$

format S

— sll —

encodage



action

Décalage à gauche

syntaxe

`sll rd, rs1, rs2`

description

Le registre `rs1` est décalé à gauche de la valeur immédiate codée dans les 5 bits du `rs2`, des zéros étant introduits dans les bits de poids faibles. Le résultat est placé dans le registre `rd`.

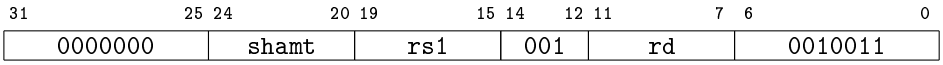
opération

$\texttt{rd} \leftarrow \texttt{rs1}_{31-\texttt{rs2}_{4\ldots0}\ldots0} \parallel 0^{\texttt{rs2}_{4\ldots0}}$

format R

— slli —

encodage



action

Décalage à gauche immédiat

syntaxe

`slli rd, rs1, imm`

description

Le registre `rs1` est décalé à gauche du nombre de bits spécifiés par l'immédiat `shamt`, des zéros étant introduits dans les bits de poids faibles. L'immédiat `shamt` occupe les bits habituellement utilisés pour coder `rs2`. Le résultat est placé dans le registre `rd`.

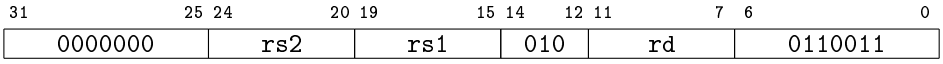
opération

$\texttt{rd} \leftarrow \texttt{rs2}_{31-shamt\ldots0} \parallel 0^{shamt}$

format R

— slt —

encodage



action

Comparaison signée registre registre

syntaxe

`slt rd, rs1, rs2`

description

Le contenu du registre `rs1` est comparé au contenu du registre `rs2`, les deux valeurs étant considérées comme des quantités signées. Si la valeur contenue dans `rs1` est inférieure à celle contenue dans `rs2`, alors `rd` prend la valeur '1', sinon il prend la valeur '0'.

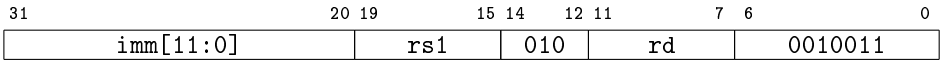
opération

$\texttt{rs1} < \texttt{rs2} \Rightarrow \texttt{rd} \leftarrow 0^{31} \parallel 1$
$\texttt{rs1} \geq \texttt{rs2} \Rightarrow \texttt{rd} \leftarrow 0^{32}$

format R

— slti —

encodage



action

Comparaison signée registre immédiat

syntaxe

`slti rd, rs1, imm`

description

Le contenu du registre `rs1` est comparé à la valeur immédiate sur 12 bits qui a subit une extension de signe, les deux valeurs étant considérées comme des quantités signées. Si la valeur contenue dans `rs1` est inférieure à celle de l'immédiat étendu, alors `rd` prend la valeur '1', sinon il prend la valeur '0'.

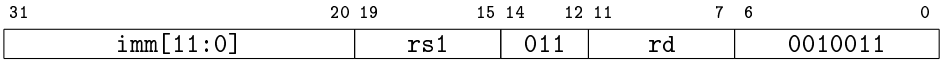
opération

$\texttt{rs1} < (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) \Rightarrow \texttt{rd} \leftarrow 0^{31} \parallel 1$
$\texttt{rs1} \geq (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20}) \Rightarrow \texttt{rd} \leftarrow 0^{32}$

format I

— sltiu —

encodage



action

Comparaison non-signée registre immédiat

syntaxe

`sltiu rd, rs1, imm`

description

Le contenu du registre `rs1` est comparé à la valeur immédiate sur 12 bits qui a subit une extension de signe. Les deux valeurs sont considérées comme des quantités non-signées. Si la valeur contenue dans `rs1` est inférieure à celle de l'immédiat étendu, alors `rd` prend la valeur '1', sinon il prend la valeur '0'.

opération

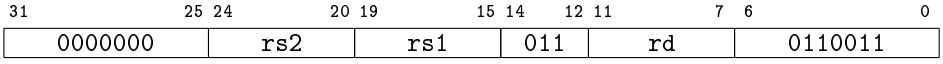
$(0 \parallel \texttt{rs1}) < (0 \parallel (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20})) \Rightarrow \texttt{rd} \leftarrow 0^{31} \parallel 1$

$(0 \parallel \texttt{rs1} \geq 0 \parallel (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20})) \Rightarrow \texttt{rd} \leftarrow 0^{32}$

format I

— sltu —

encodage



action

Comparaison non-signée registre registre

syntaxe

`sltu rd, rs1, rs2`

description

Le contenu du registre `rs1` est comparé au contenu du registre `rs2`, les deux valeurs étant considérés comme des quantités non-signées. Si la valeur contenue dans `rs1` est inférieure à celle contenue dans `rs2`, alors `rd` prend la valeur '1', sinon il prend la valeur '0'.

opération

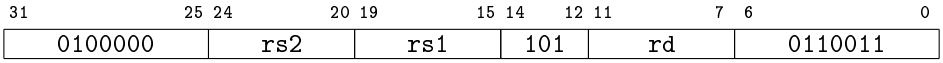
$(0 \parallel \texttt{rs1}) < (0 \parallel \texttt{rs2}) \Rightarrow \texttt{rd} \leftarrow 0^{31} \parallel 1$

$(0 \parallel \texttt{rs1}) \geq (0 \parallel \texttt{rs2}) \Rightarrow \texttt{rd} \leftarrow 0^{32}$

format R

— sra —

encodage



action

Décalage à droite arithmétique registre

syntaxe

`sra rd, rs2, rs1`

description

Le registre `rs1` est décalé à droite du nombre de bits spécifiés dans les 5 bits de poids faible du registre `rs2`, le signe de `rs1` étant introduit dans les bits de poids fort ainsi libérés. Le résultat est placé dans le registre `rd`.

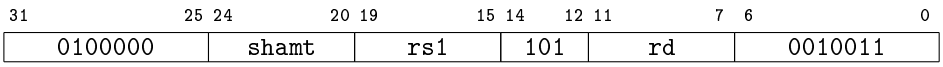
opération

$\texttt{rd} \leftarrow \texttt{rs1}_{31}^{\texttt{rs2}_{4\ldots0}} \parallel \texttt{rs1}_{31\ldots\texttt{rs2}_{4\ldots0}\ldots0}$

format R

— srai —

encodage



action

Décalage à droite arithmétique immédiat

syntaxe

`sra rd, rs1, shamt`

description

Le registre `rs1` est décalé à droite de la valeur immédiate codée dans les 5 bits du champ `shamt`, le bit de signe du registre étant introduit dans les bits de poids fort. Le résultat est placé dans le registre `rd`.

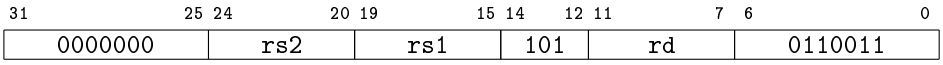
opération

$\texttt{rd} \leftarrow \texttt{rs1}_{31}^{shamt} \parallel \texttt{rs1}_{31\ldots shamt}$

format R

— srl —

encodage



action

Décalage à droite logique registre

syntaxe

```
srlv rd, rs2, rs1
```

description

Le registre `rs1` est décalé à droite du nombre de bits spécifiés dans les 5 bits de poids faible du registre `rs2`, des zéros étant introduits dans les bits de poids fort ainsi libérés. Le résultat est placé dans le registre `rd`.

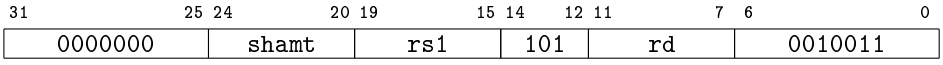
opération

$\texttt{rd} \leftarrow 0^{\texttt{rs2}_{4\ldots0}} \parallel \texttt{rs1}_{31\ldots\texttt{rs2}_{4\ldots0}}$

format R

— srli —

encodage



action

Décalage à droite logique immédiat

syntaxe

```
srl rd, rs1, shamt
```

description

Le registre `rs1` est décalé à droite de la valeur immédiate codée dans les 5 bits du champ `shamt`, des zéros étant introduits dans les bits de poids fort. Le résultat est placé dans le registre `rd`.

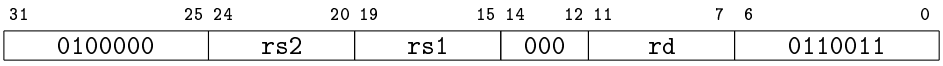
opération

$\texttt{rd} \leftarrow 0^{shamt} \parallel \texttt{rs1}_{31\ldots shamt}$

format R

— sub —

encodage



action

Soustraction registre registre signée

syntaxe

```
sub rd, rs1, rs2
```

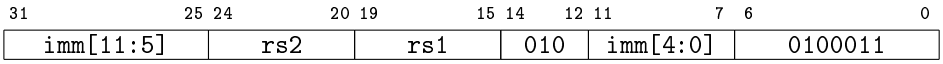
description
Le contenu du registre `rs2` est soustrait du contenu du registre `rs1` pour former un résultat sur 32 bits qui est placé dans le registre `rd`.

opération
`rd` $\leftarrow$ `rs1` $-$ `rs2`

format R

— sw —

encodage



action
Écriture d'un mot en mémoire

syntaxe
`sw rs2, imm(rs1)`

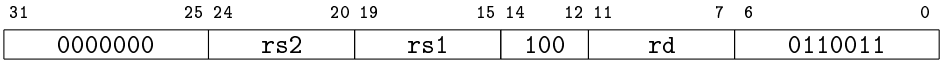
description
La constante `cst` construite à partir de l'`imm` est étendue de signe et sommé avec le contenu du registre `rs1`. La mot contenu dans le registre `rs2` est écrit à l'adresse ainsi calculée.

opération
`cst` $= (\mathtt{IR}_{31}^{20} \parallel \mathtt{IR}_{31\ldots25} \parallel \mathtt{IR}_{11\ldots7})$
mem$[$`cst` $+$ `rs1`$] \leftarrow$ `rs2`

format S

— xor —

encodage



action
Ou-exclusif bit-à-bit registre registre

syntaxe
`xor rd, rs1, rs2`

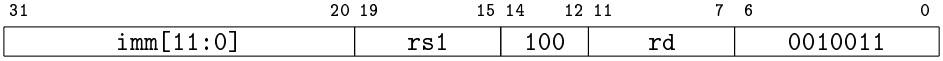
description
Un `ou-exclusif` bit-à-bit est effectué entre les contenus des registres `rs1` et `rs2`. Le résultat est placé dans le registre `rd`.

opération
`rd` $\leftarrow$ `rs1` xor `rs2`

format R

— xori —

encodage



action
Ou-exclusif bit-à-bit registre immédiat

syntaxe
`xori rd, rs1, imm`

description

La valeur immédiate sur 12 bits subit une extension de signe. Un `ou-exclusif` bit-à-bit est effectué entre cette valeur étendue et le contenu du registre `rs1` pour former un résultat placé dans le registre `rs2`.

opération

$\texttt{rd} \leftarrow (\texttt{IR}_{31}^{20} \parallel \texttt{IR}_{31\ldots20})$ xor `rs1`

format I