Instructions MIPS

Légende :
- R[] = registre
- M[] = mémoire

add
Add
Addition de deux registres et place la somme dans un registre.

Type R
opcode : 0x00
funct : 0x20

R[rd] = R[rs] + R[rt]

```
add $t0,$s1,$s2
# t0 = s1 + s2
```

addi
Add Immediate
Addition de la valeur d'un registre avec une valeur constante et place la somme dans un registre.

Type I
opcode : 0x08

R[rt] = R[rs] + Imm

```
addi $s3,$s3,1
# s3 = s3 + 1
ou
# s3++
```

addiu
Add Immediate Unsigned
Addition non-signée de la valeur d'un registre et une valeur constante. La somme est placée dans un registre.

Type I
opcode : 0x09

R[rt] = R[rs] + Imm

```
addiu $s3,$s3,1
# s3 = s3 + 1
ou
# s3++
```

addu
Add Unsigned
Addition non-signée de deux registres et la somme est placée dans un registre.

Type R
opcode : 0x00
funct : 0x21

R[rd] = R[rs] + R[rt]

addu $t0,$s1,$s2
# t0 = s1 + s2

and
And
ET logique, bit par bit, entre 2 registres. Le résultat est placé dans un registre.

Type R
opcode : 0
funct : 0x24

R[rd] = R[rs] & R[rt]

and $t0,$t1,$t2
t0 = t1 & t2

andi
And Immediate
ET logique, bit par bit, entre un registre et une constante. Le résultat est placé dans un registre.

Type I
opcode : 0x0C

R[rt] = R[rs] & Imm

andi $s2,$t0,7
s2 = t0 & 7 (masquage des 3 derniers bits)

beq
Branch on Equal
On effectue un branchement à l'étiquette donnée si les deux registres sont égaux.

Type I
opcode : 0x04

if (R[rs] == R[rt]) then PC = PC + 4 + Imm

beq $s1,$zero,Fin
si s1 == 0, on branche à l'étiquette Fin

bne
Branch on not Equal
On effectue un branchement à l'étiquette donnée si les deux registres sont différents.

Type I
opcode : 0x05

if (R[rs] != R[rt]) then PC = PC + 4 + Imm

bne $s1,$zero,Fin
si s1 != 0, on branche à l'étiquette Fin

j
Jump
Saute à une adresse dans le programme.

Type-J
opcode : 0x02

PC = address

j Exit
on saute à l'étiquette Exit

jal
Jump and Link
Saute à une fonction. S'utilise de pair avec l'instruction jr.

Type J
opcode : 0x03

R[31] = PC + 8; PC = address

jal Calculate
on saute à l'étiquette Calcute avec l'information pour revenir où l'on était

jr
Jump Register
Saute à une adresse dans un registre. S'utilise de pair avec l'instruction jal.

Type R
opcode : 0
funct : 0x08

PC = R[rs]

```
jr $ra
# on saute à l'adresse dans le registre ra
```

lb
Load Byte
Charge un octet en mémoire dans un registre en considérant le signe (par extension de signe).

Type I
opcode : 0x20

R[rt] = 24*signe, M[R[rs] + Imm]

```
lb $s2,0($t2)
```

lbu
Load Byte Unsigned
Charge un octet non-signé en mémoire dans un registre.

Type I
opcode : 0x24

R[rt] = 24*0, M[R[rs]] (moins significatif), des 0 dans le reste

```
lbu $s2,0($t2)
```

lh
Load Halfword
Charge deux octets en mémoire dans un registre en considérant le signe (par extension de signe).

Type I
opcode : 0x21

R[rt] = 16*signe, M[R[rs] + Imm]

```
lh $s2,0($t2)
```

lhu
Load Halfword Unsigned
Charge deux octets non-signés en mémoire dans un registre.

Type I
opcode : 0x25

R[rt] = 16*0, M[R[rs]] (moins significatifs), des 0 dans le reste

lhu $s2,0($t2)

lui
Load Upper Immediate
Charge une valeur constante dans les deux octets les plus significatifs d'un registre

Type I
opcode : 0x0F

R[rt] = Imm (dans les 2 octets les plus significatifs), des 0 dans le reste

lui $s2,8
s2 = 8 << 16

lw
Load Word
Charge 4 octets d'une adresse mémoire dans un registre.

Type I
opcode : 0x23

R[rt] = M[R[rs] + Imm]

lw $t0,32($s3)
# t0 = s3[8]

nor
Nor
Négation du résultat d'un OU entre deux registres. Le résultat est placé dans un registre.

Type R
opcode : 0
funct : 0x27

R[rd] = !(R[rs] | R[rt])

```
nor $t0,$s1,$s3
# t0 = !(s1 | s3)
```

or
Or
OU logique, bit par bit, entre deux registres. Le résultat est placé dans un registre.

Type R
opcode : 0
funct : 0x25

R[rd] = R[rs] | R[rt]

```
or $t0,$t1,$t2
# t0 = t1 | t2
```

ori
Or Immediate
OU logique, d'un registre et une constante. Le résultat est placé dans un registre.

Type I
opcode : 0x0D

R[rt] = R[rs] | Imm

```
ori $t0,$s5,7
# t0 = s5 | 7
```

sb
Store Byte
Sauvegarder un octet en mémoire.

Type I
opcode : 0x28

M[R[rs] + Imm] = R[rt] (octet le moins significatif)

sb $t1,0($s2)
s2[0] = t1

sh
Store Halfword
Sauvegarder deux octets en mémoire.

Type I
opcode : 0x29

M[R[rs] + Imm] = R[rt] (2 octets les moins significatifs)

sh $t1,0($s2)
s2[0] = t1

sll
Shift Left Logical
Décalement des bits vers la gauche. Un décalement d'un bit vers la gauche correspond à une multiplication par 2.

Type R
opcode : 0x00
funct : 0x00

R[rd] = R[rt] << shamt

sll $t1,$s3,2
t1 = s3 << 2 bits
ou
t1 = s3 * 4

slt
Set Less Than
Affecte un registre indiquant le résultat de la comparaison de deux registres.

Type R
opcode : 0
funct : 0x2A

R[rd] = (R[rs] < R[rt]) ? 1 : 0

slt $t0,$s3,$s4
t0 = (s3 < s4) ? 1 : 0

slti
Set Less Than Immediate
Affecte un registre indiquant le résultat de la comparaison d'un registre et d'une valeur immédiate.

Type I
opcode : 0x0A

R[rt] = (R[rs] < Imm) ? 1 : 0

slti $t0,$s2,10
t0 = (s2 < 10) ? 1 : 0

sltiu
Set Less Than Immediate Unsigned
Affecte un registre indiquant le résultat de la comparaison non-signée d'un registre et d'une valeur immédiate.

Type I
opcode : 0x0B

R[rt] = (R[rs] < Imm) ? 1 : 0

sltiu $t0,$s2,10
t0 = (s2 < 10) ? 1 : 0

sltu
Set Less Than Unsigned
Affecte un registre indiquant le résultat de la comparaison non-signée de deux registres.

Type R
opcode : 0
funct : 0x2B

R[rd] = (R[rs] < R[rt]) ? 1 : 0

sltu $t0,$s3,$s4
t0 = (s3 < s4) ? 1 : 0

srl
Shift Right Logical
Décalement des bits vers la droite. Un décalement d'un bit vers la droite correspond à une division par 2.

Type R
opcode : 0x00
funct : 0x02

R[rd] = R[rt] >> shamt

srl $t0,$s2,1
t0 = s2 >> 1 bit
ou
t0 = s2 / 2

sub
Subtract
Soustraction de deux registres et la différence est placée dans un registre.

Type R
opcode : 0x00
funct : 0x22

R[rd] = R[rs] – R[rt]

sub $t0,$s1,$s2
# t0 = s1 – s2

subu
Substract Unsigned
Soustraction non-signée de deux registres. La différence est placée dans un registre.

Type R
opcode : 0
funct : 0x23

R[rd] = R[rs] – R[rt]

subu $t0,$s1,$s2
# t0 = s1 – s2

sw
Store Word
Sauvegarder quatre octets en mémoire.

Type I
opcode : 0x2B

M[R[rs] + Imm] = R[rt]

sw $t0,1200($t1)
# t1[300] = t0