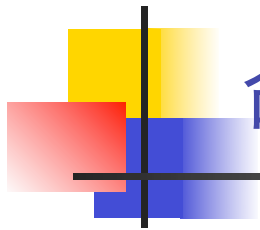


32ビットマイクロプロセッサp32の 基本命令セット

(version 1.4)

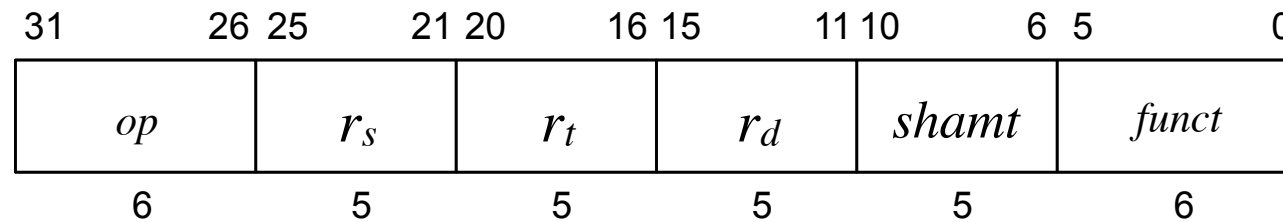
岡山大学大学院自然科学研究科

渡邊 誠也

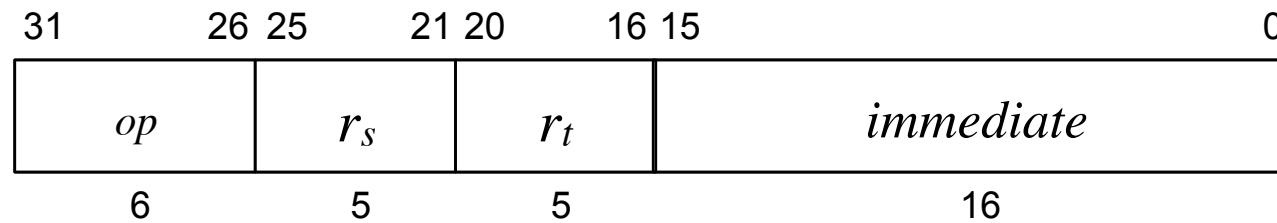


命令形式

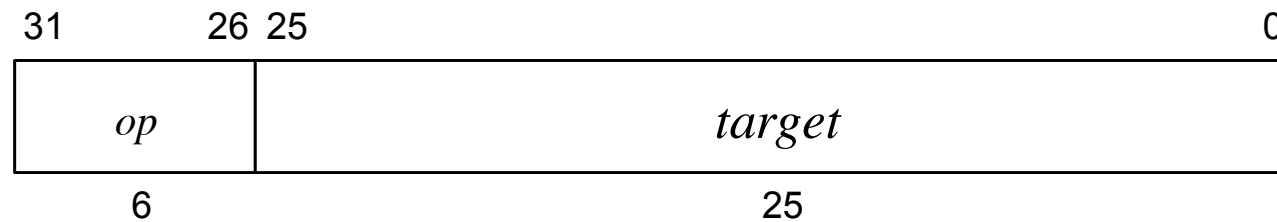
■ R形式



■ I形式



■ J形式





命令セットの概要

- ロード/ストア命令
 - メモリ⇄プロセッサ（レジスタ）間のデータ転送
 - lw, sw, lb, sb
- 演算命令
 - 算術演算, 論理演算, シフト演算, 比較演算など
 - add, sub, ..., add, ..., addi, andi, ...
- 条件分岐命令
 - 条件による分岐
 - beq, bne
- ジャンプ命令
 - 無条件分岐
 - j, jal, jr, jalr
- その他
 - 定数操作, システムコール
 - lui, syscall



主要な命令

■ 演算命令

- add add
- sub subtract
- addi add immediate
- and and

■ ロード/ストア命令

- lw load word
- sw store word

■ 条件分岐命令

- beq branch on equal
- bne branch on not equal

■ ジャンプ命令

- j jump
- jal jump and link
- jr jump register
- jalr jump and link reg.

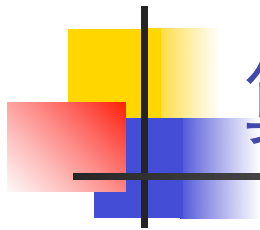
■ その他

- lui load upper imm.
- syscall system call



オペレーションの表記法

表記	意味
GPR [x]	レジスタ番号 x の汎用レジスタ
MEM [x]	バイトアドレス x のメモリ（ワード）
$y \leftarrow x$	データ x を記憶素子 y へ格納（転送）
$x_{a..b}$	ビットストリング x の a ビットから b ビットまでの部分ビットストリング（リトル・エンディアンビット表記, $a > b$ ）
x_a	ビットストリング x の a ビット目のビット（1ビット）
x^a	ビットストリング x を a ビット繰り返したビットストリング（ x は常に1ビットの値）
$x \parallel y$	ビットストリング x と y のビットストリング連結



算術・論理演算命令

算術演算命令

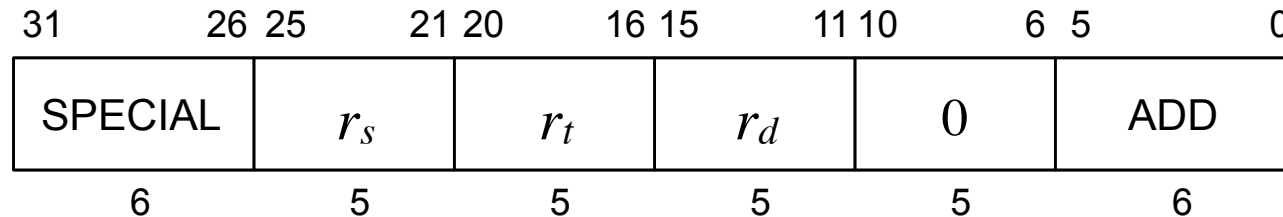
- ADD
- ADDI
- ADDU
- ADDIU
- SUB
- SUBU
- MULT
- MULTU
- DIV
- DIVU

論理演算命令

- AND
- ANDI
- OR
- ORI
- XOR
- XORI
- NOR



add



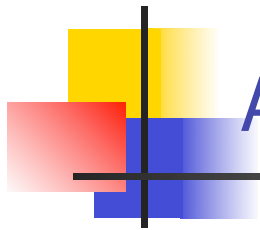
命令書式 `add r_d , r_s , r_t`

説明 汎用レジスタ r_s の内容と汎用レジスタ r_t の内容を加算した結果を生成. 結果を汎用レジスタ r_d に格納.

キャリーアウトの上位2ビットが異なる場合 (2の補数のオーバフロー) には, オーバフロー例外が発生.

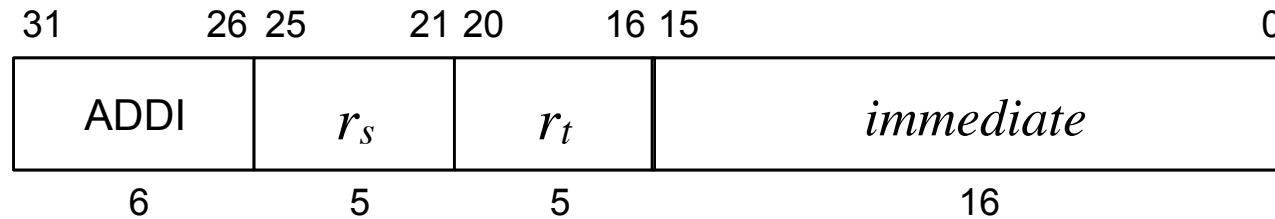
オペレーション

T: **GPR**[r_d] \leftarrow **GPR**[r_s] + **GPR**[r_t]



ADDI

add immediate



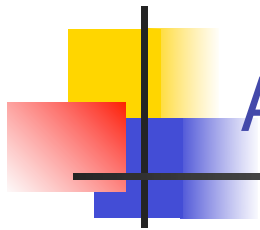
命令書式 `addi r_t , r_s , immediate`

説明 汎用レジスタ r_s の内容と16ビット即値 *immediate* を32ビットに符号拡張した値を加算した結果を生成. 結果を汎用レジスタ r_t に格納.

キャリーアウトの上位2ビットが異なる場合 (2の補数のオーバフロー) には, オーバフロー例外が発生.

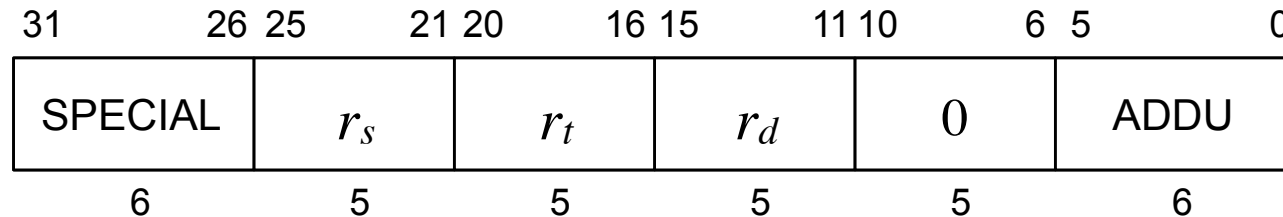
オペレーション

T: $\mathbf{GPR}[r_t] \leftarrow \mathbf{GPR}[r_s] + (immediate_{15})^{16} \parallel immediate_{15..0}$



ADDU

add unsigned



命令書式 `addu r_d , r_s , r_t`

説明 汎用レジスタ r_s の内容と汎用レジスタ r_t の内容を加算した結果を生成. 結果を汎用レジスタ r_d に格納

※ADDと同じ動作だが、オーバフロー例外は起こさない

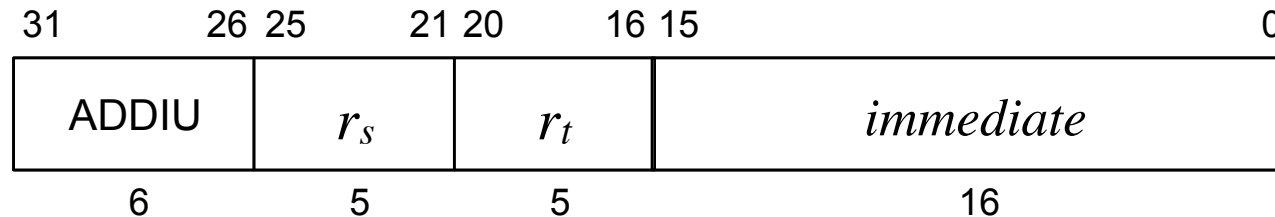
オペレーション

T: **GPR[r_d] ← GPR[r_s] + GPR[r_t]**



ADDIU

add immediate unsigned



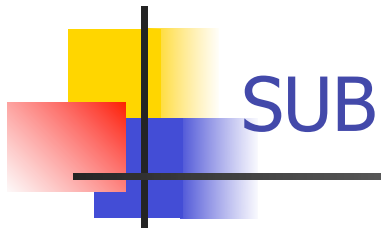
命令書式 `addiu r_t , r_s , immediate`

説明 汎用レジスタ r_s の内容と16ビット即値 $immediate$ を32ビットに符号拡張した値を加算した結果を生成. 結果を汎用レジスタ r_t に格納.

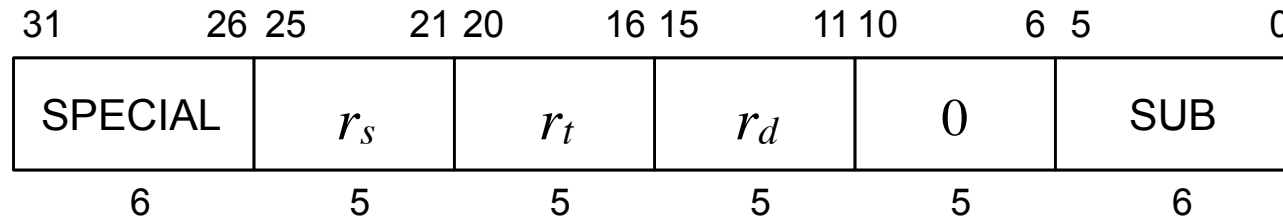
※ADDIと同じ動作だが、オーバフロー例外は起こさない

オペレーション

T: $\mathbf{GPR}[r_t] \leftarrow \mathbf{GPR}[r_s] + (immediate_{15})^{16} \parallel immediate_{15..0}$



subtract



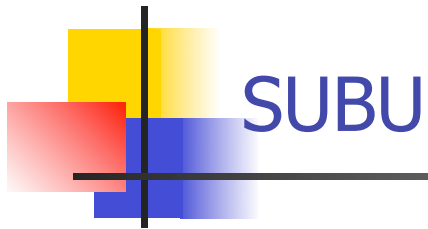
命令書式 **sub** r_d , r_s , r_t

説明 汎用レジスタ r_s の内容からレジスタ r_t の内容を減算した結果を生成. 結果を汎用レジスタ r_d に格納

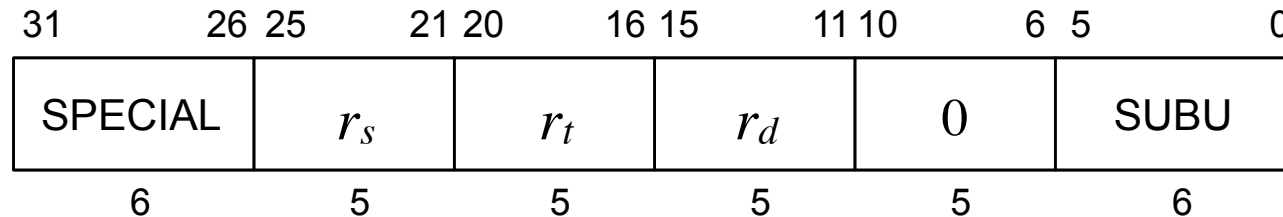
キャリーアウトの上位2ビットが異なる場合 (2の補数のオーバフロー) には, オーバフロー例外が発生.

オペレーション

T: **GPR[r_d] \leftarrow GPR[r_s] - GPR[r_t]**



subtract unsigned



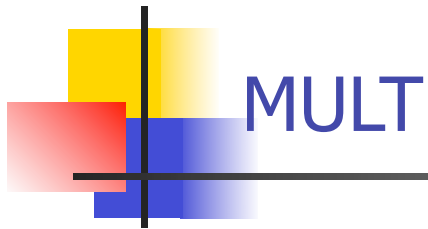
命令書式 `subu r_d , r_s , r_t`

説明 汎用レジスタ r_s の内容から汎用レジスタ r_t の内容を減算した結果を生成.
結果を汎用レジスタ r_d に格納

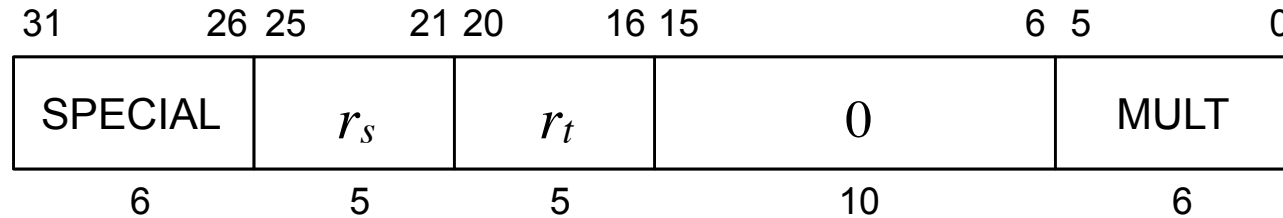
※SUBと同じ動作だが、オーバフロー例外は起こさない

オペレーション

T: **$\text{GPR}[r_d] \leftarrow \text{GPR}[r_s] - \text{GPR}[r_t]$**



multiply



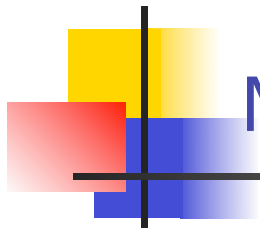
命令書式 `mult r_s , r_t`

説明 汎用レジスタ r_s の内容と汎用レジスタ r_t の内容を、両方のオペランドを32ビットの2の補数の値として扱い乗算。結果は特殊レジスタHI（上位ワード）およびLO（下位ワード）に格納。

オーバフロー例外が発生しない。

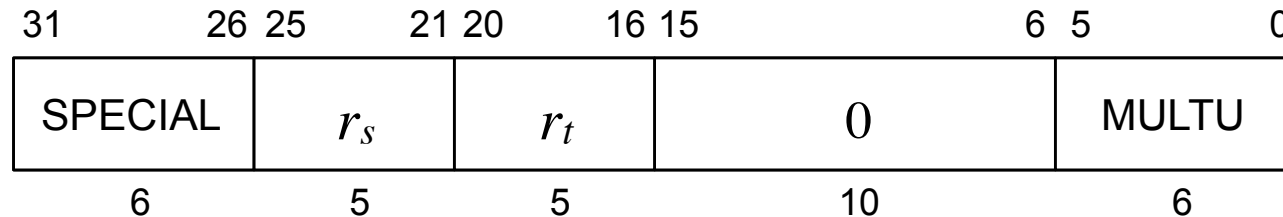
オペレーション

T: $t \leftarrow \mathbf{GPR}[r_s] \times \mathbf{GPR}[r_t]$
 LO $\leftarrow t_{31..0}$
 HI $\leftarrow t_{63..32}$



MULTU

multiply unsigned



命令書式 `multu r_s , r_t`

説明 汎用レジスタ r_s の内容と汎用レジスタ r_t の内容を、両方のオペランドを32ビットの無符号値として扱い乗算。結果は特殊レジスタHI（上位ワード）およびLO（下位ワード）に格納。

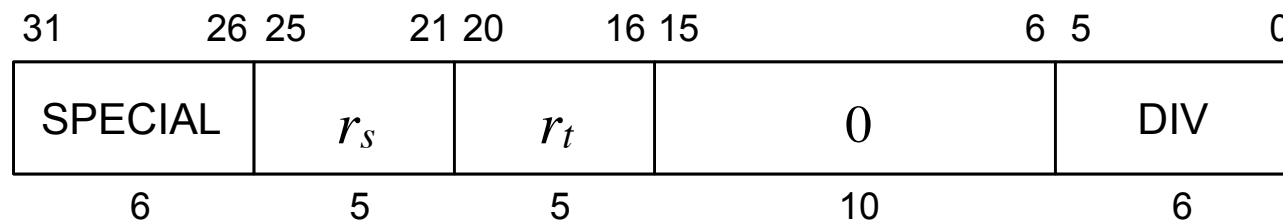
オーバフロー例外が発生しない。

オペレーション

T: $t \leftarrow (0 \parallel \mathbf{GPR}[r_s]) \times (0 \parallel \mathbf{GPR}[r_t])$
 LO $\leftarrow t_{31..0}$
 HI $\leftarrow t_{63..32}$



divide



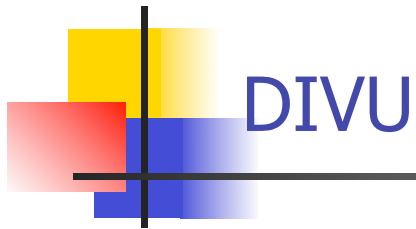
命令書式 `div r_s , r_t`

説明 汎用レジスタ r_s の内容を汎用レジスタ r_t の内容で除算。両方のオペランドを32ビットの2の補数の値として扱う。演算結果の商ワードは特殊レジスタLOに、剰余ワードは特殊レジスタHIにそれぞれ格納。

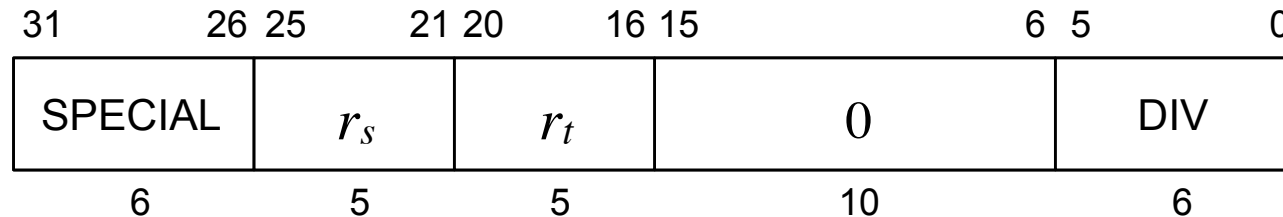
オーバフロー例外が発生しない。

オペレーション

T: **LO** \leftarrow **GPR**[r_s] div **GPR**[r_t]
 HI \leftarrow **GPR**[r_s] mod **GPR**[r_t]



divide unsigned



命令書式 `divu r_s , r_t`

説明 汎用レジスタ r_s の内容を汎用レジスタ r_t の内容で除算. 両方のオペランドを32ビットの無符号値として扱う. 演算結果の商ワードは特殊レジスタLOに, 剰余ワードは特殊レジスタHIにそれぞれ格納.

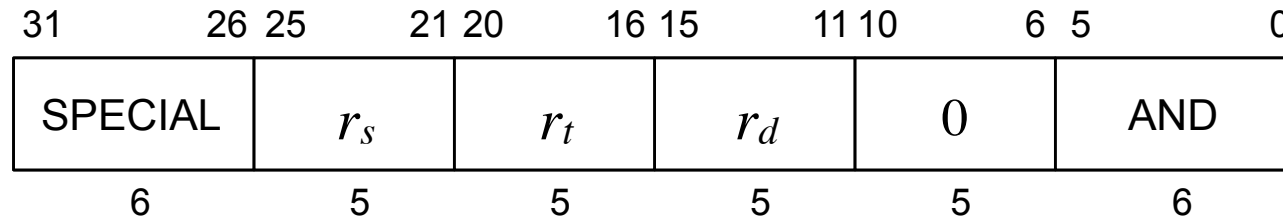
オーバフロー例外が発生しない.

オペレーション

T: **LO** \leftarrow (0 || **GPR**[r_s]) div (0 || **GPR**[r_t])
 HI \leftarrow (0 || **GPR**[r_s]) mod (0 || **GPR**[r_t])



and

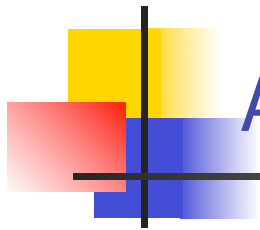


命令書式 **and** r_d , r_s , r_t

説明 汎用レジスタ r_s の内容と汎用レジスタ r_t の内容のビット毎の論理積を生成。結果を汎用レジスタ r_d に格納

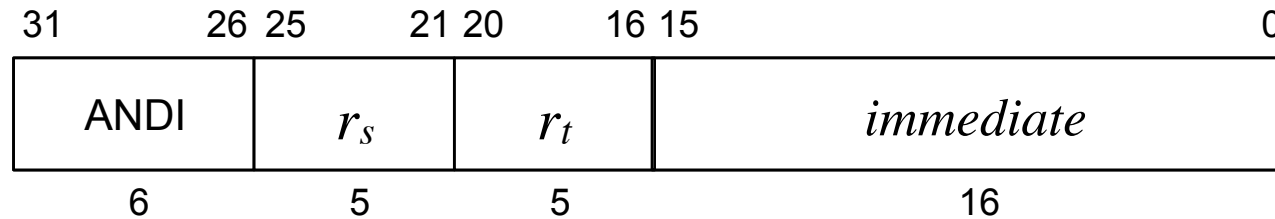
オペレーション

T: **GPR**[r_d] \leftarrow **GPR**[r_s] and **GPR**[r_t]



ANDI

and immediate

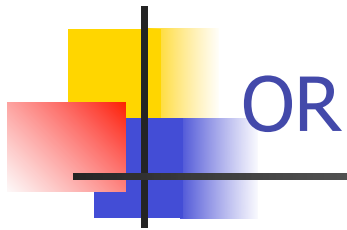


命令書式 `andi` r_t , r_s , *immediate*

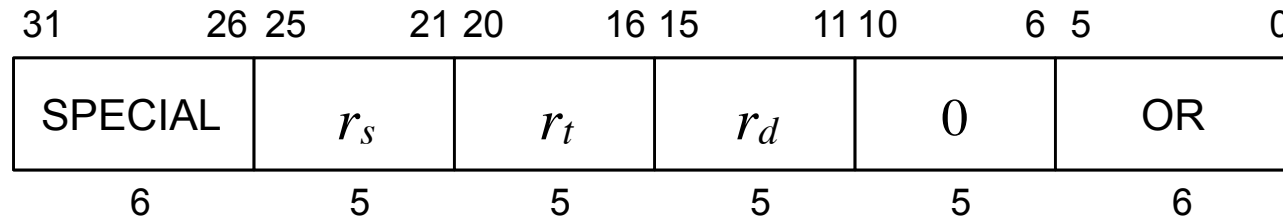
説明 汎用レジスタ r_s の内容と16ビット即値*immediate*を32ビットにゼロ拡張した値との論理積を生成. 結果を汎用レジスタ r_t に格納.

オペレーション

T: $\mathbf{GPR}[r_t] \leftarrow \mathbf{GPR}[r_s]_{31..16} \text{ and } (immediate \text{ or } \mathbf{GPR}[r_s]_{15..0})$



or



命令書式

or r_d, r_s, r_t

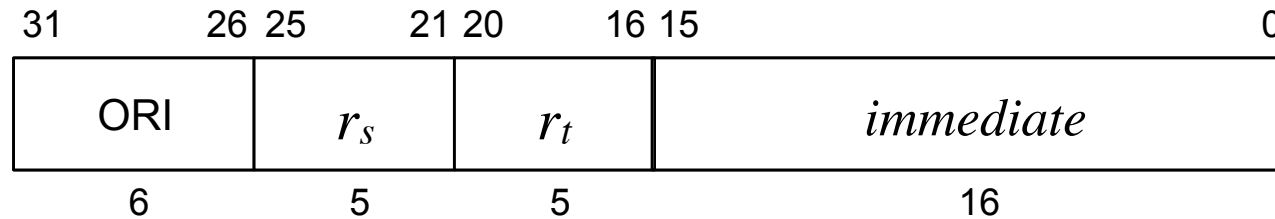
説明 汎用レジスタ r_s の内容と汎用レジスタ r_t の内容の論理和を生成. 結果を汎用レジスタ r_d に格納

オペレーション

T: **GPR** $[r_d] \leftarrow \mathbf{GPR}[r_s] \text{ or } \mathbf{GPR}[r_t]$



or immediate

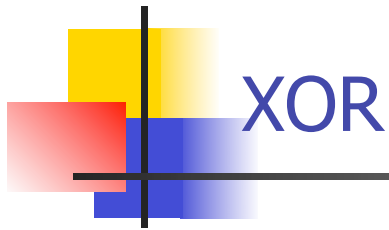


命令書式 `ori` r_t , r_s , *immediate*

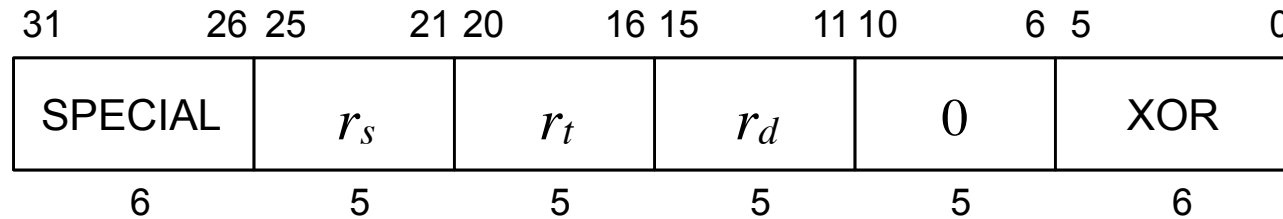
説明 汎用レジスタ r_s の内容と16ビット即値*immediate*を32ビットにゼロ拡張した値との論理和を生成. 結果を汎用レジスタ r_t に格納.

オペレーション

T: **GPR**[r_t] \leftarrow **GPR**[r_s]_{31..16} or (*immediate* or **GPR**[r_s]_{15..0})



exclusive or

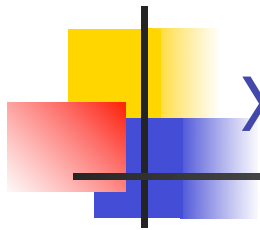


命令書式 **xor** r_d , r_s , r_t

説明 汎用レジスタ r_s の内容と汎用レジスタ r_t の内容の排他的論理和を生成. 結果を汎用レジスタ r_d に格納

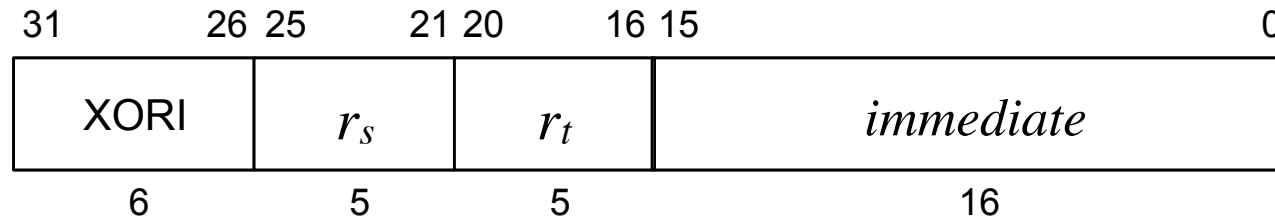
オペレーション

T: **GPR**[r_d] ← **GPR**[r_s] xor **GPR**[r_t]



XORI

exclusive or immediate

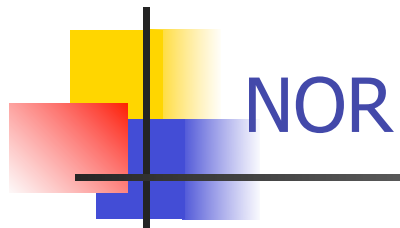


命令書式 **xori** r_t , r_s , *immediate*

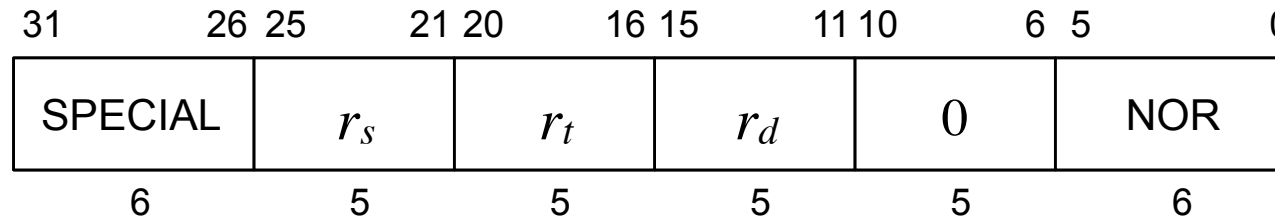
説明 汎用レジスタ r_s の内容と16ビット即値*immediate*を32ビットにゼロ拡張した値との排他的論理和を生成. 結果を汎用レジスタ r_t に格納.

オペレーション

T: **GPR**[r_t] \leftarrow **GPR**[r_s]_{31..16} xor (*immediate* xor **GPR**[r_s]_{15..0})



not or

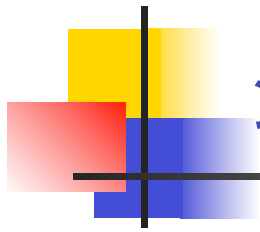


命令書式 **nor** r_d , r_s , r_t

説明 汎用レジスタ r_s の内容と汎用レジスタ r_t の内容のビット毎の否定論理和を生成。結果を汎用レジスタ r_d に格納

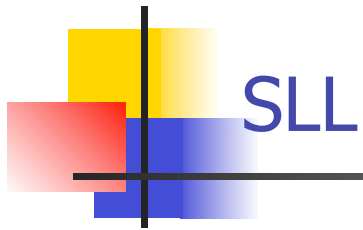
オペレーション

T: **GPR**[r_d] \leftarrow **GPR**[r_s] **nor** **GPR**[r_t]



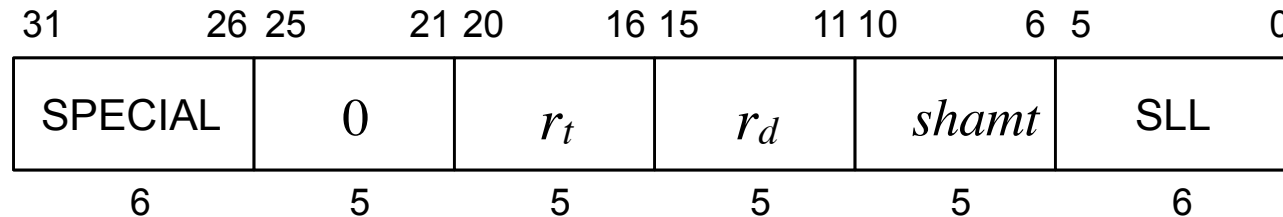
シフト演算命令

- SLL
- SRL
- SRA
- SLLV
- SRLV
- SRAV



SLL

shift left logical



命令書式 `sll` $r_d, r_t, shamt$

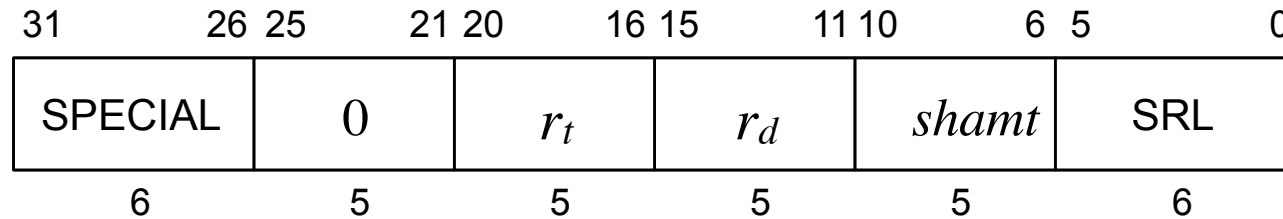
説明 0を下位ビットに挿入して、汎用レジスタ r_t の内容を $shamt$ ビットだけ左にシフト。32ビットの結果はレジスタ r_d に格納。

オペレーション

T: $\mathbf{GPR}[r_d] \leftarrow ((\mathbf{GPR}[r_t])_{(31 - shamt)..0}) \parallel 0^{shamt}$



shift right logical



命令書式 `srl` $r_d, r_t, shamt$

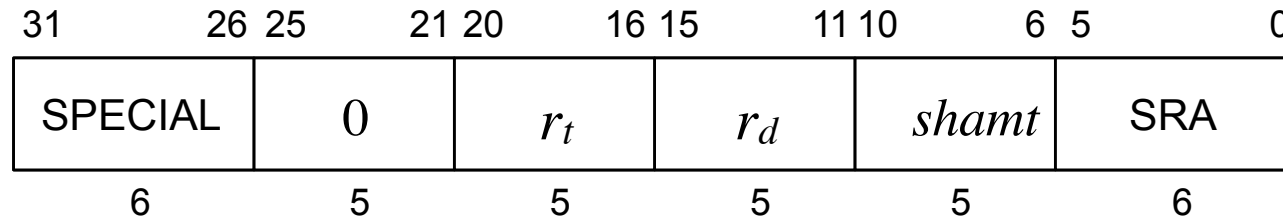
説明 0を上位ビットに挿入して、汎用レジスタ r_t の内容を $shamt$ ビットだけ右にシフト。32ビットの結果はレジスタ r_d に格納。

オペレーション

T: $\mathbf{GPR}[r_d] \leftarrow 0^{shamt} \parallel ((\mathbf{GPR}[r_t])_{31..shamt})$



shift right arithmetic

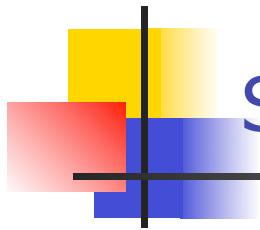


命令書式 **sra** $r_d, r_t, shamt$

説明 上位ビットを符号拡張して汎用レジスタ r_t の内容を $shamt$ ビットだけ右にシフト. 32ビットの結果はレジスタ r_d に格納.

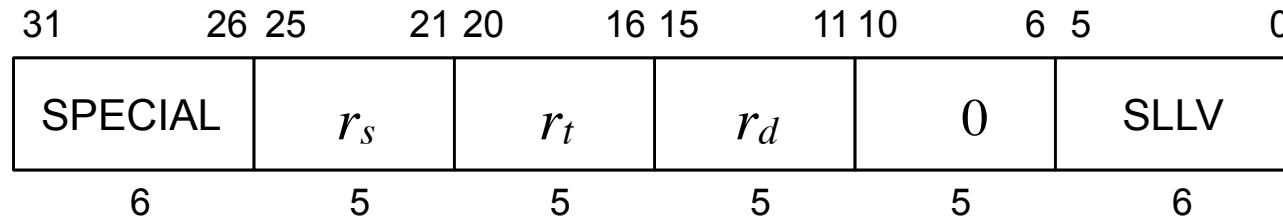
オペレーション

T: **GPR**[r_d] \leftarrow (**GPR**[r_t]) $_{31}^{shamt}$ || ((**GPR**[r_t]) $_{31..shamt}$)



SLLV

shift left logical variable

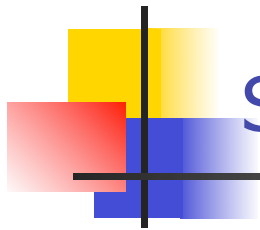


命令書式 `sllv` r_d , r_t , r_s

説明 0を下位ビットに挿入して、汎用レジスタ r_t の内容を汎用レジスタ r_s の内容の下位5ビットで指定されるビット数だけ左にシフト。32ビットの結果はレジスタ r_d に格納。

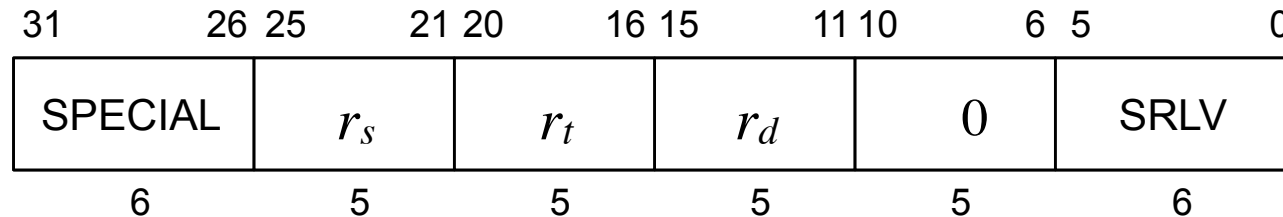
オペレーション

T: $\mathbf{GPR}[r_d] \leftarrow ((\mathbf{GPR}[r_t])_{(31 - \mathbf{GPR}[rs]_{4..0})..0}) \parallel 0 \mathbf{GPR}[rs]_{4..0}$



SRLV

shift right logical variable

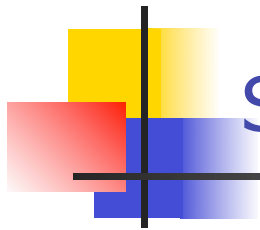


命令書式 `srlv` r_d, r_t, r_s

説明 0を上位ビットに挿入して、汎用レジスタ r_t の内容を汎用レジスタ r_s の内容の下位5ビットで指定されるビット数だけ右にシフト。32ビットの結果はレジスタ r_d に格納。

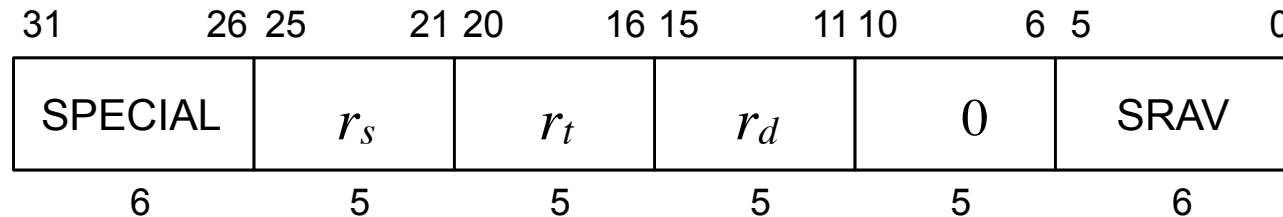
オペレーション

T: $\mathbf{GPR}[r_d] \leftarrow 0 \mathbf{GPR}[r_s]_{4..0} \parallel ((\mathbf{GPR}[r_t])_{31..} \mathbf{GPR}[r_s]_{4..0})$



SRAV

shift right arithmetic variable



命令書式 **srav** r_d , r_t , r_s

説明 上位ビットを符号拡張して汎用レジスタ r_t の内容を汎用レジスタ r_s の内容の下位5ビットで指定されるビット数だけ右にシフト. 32ビットの結果はレジスタ r_d に格納.

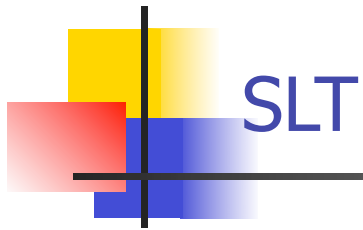
オペレーション

T: **GPR**[r_d] \leftarrow (**GPR**[r_t])₃₁ **GPR**[r_s]_{4..0} || ((**GPR**[r_t])_{31..} **GPR**[r_s]_{4..0})

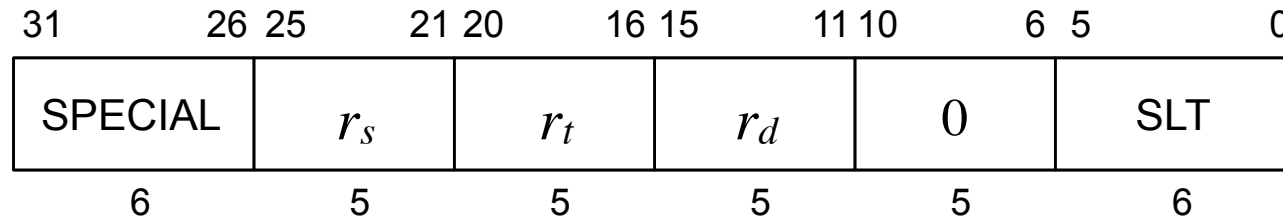


比較演算命令

- SLT
- SLTU
- SLTI
- SLTIU



set on less than

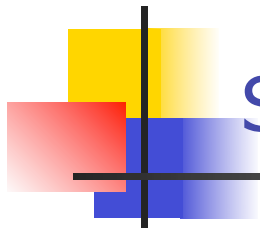


命令書式 `slt` r_d , r_s , r_t

説明 汎用レジスタ r_t の内容と汎用レジスタ r_s の内容と比較. 符号付き32ビット整数として, 汎用レジスタ r_s の内容が汎用レジスタ r_t の内容より小さい場合結果は1, そうでない場合0. 比較結果は汎用レジスタ r_d に格納.

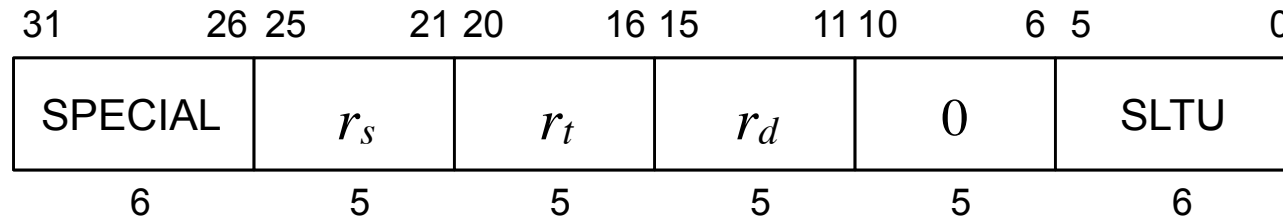
オペレーション

T: if **GPR**[r_s] < **GPR**[r_t] then
 GPR[r_d] \leftarrow 0³¹ || 1
 else
 GPR[r_d] \leftarrow 0³²
 endif



SLTU

set on less than unsigned

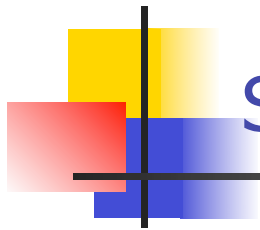


命令書式 `sltu` r_d , r_s , r_t

説明 汎用レジスタ r_t の内容と汎用レジスタ r_s の内容と比較. 無符号32ビット整数として, 汎用レジスタ r_s の内容が汎用レジスタ r_t の内容より小さい場合結果は1, そうでない場合0. 比較結果は汎用レジスタ r_d に格納.

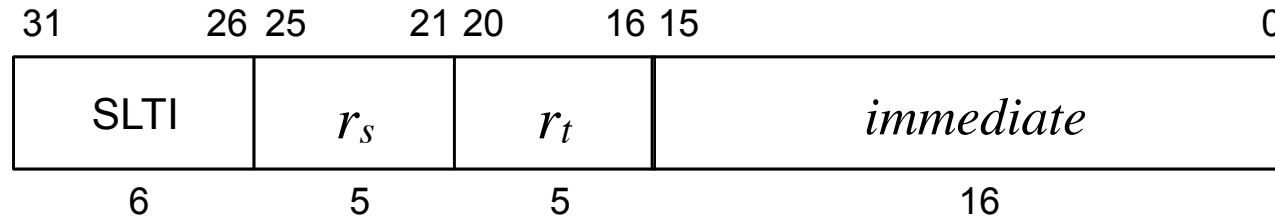
オペレーション

T: if (0 || **GPR**[r_s]) < (0 || **GPR**[r_t]) then
 GPR[r_d] \leftarrow 0³¹ || 1
 else
 GPR[r_d] \leftarrow 0³²
 endif



SLTI

set on less than immediate

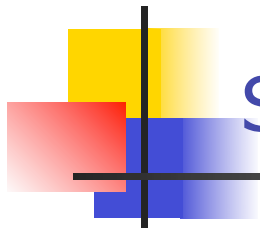


命令書式 `slti` r_t , r_s , *immediate*

説明 16ビット即値を符号拡張して汎用レジスタ r_s の内容と比較。符号付き32ビット整数として、汎用レジスタ r_s の内容が符号拡張した即値より小さい場合結果は1，そうでない場合0。比較結果は汎用レジスタ r_t に格納。

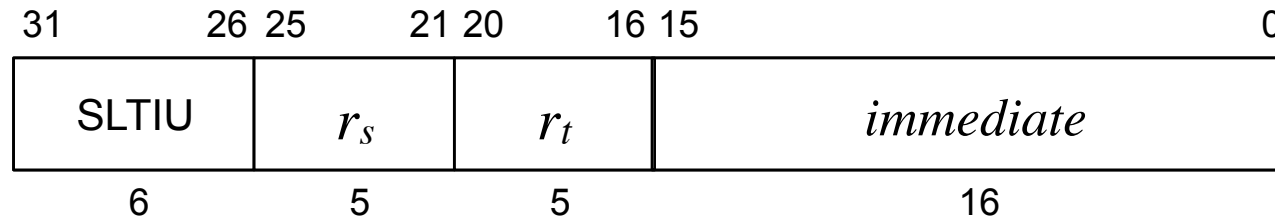
オペレーション

T: if **GPR**[r_s) < (*immediate*₁₅)¹⁶ || *immediate*_{15..0}) then
 GPR[r_t] \leftarrow 0³¹ || 1
 else
 GPR[r_t] \leftarrow 0³²
 endif



SLTIU

set on less than immediate unsigned

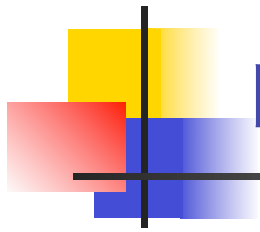


命令書式 `sltui` r_t , r_s , *immediate*

説明 16ビット即値を符号拡張して汎用レジスタ r_s の内容と比較。無符号32ビット整数として、汎用レジスタ r_s の内容が符号拡張した即値より小さい場合結果は1、そうでない場合0。比較結果は汎用レジスタ r_t に格納。

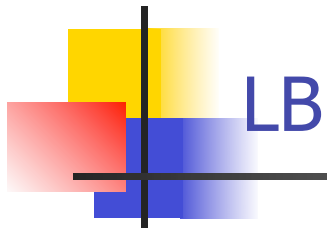
オペレーション

T: if $(0 \parallel \mathbf{GPR}[r_s]) < (0 \parallel (\mathit{immediate}_{15})^{16} \parallel \mathit{immediate}_{15..0})$ then
 $\mathbf{GPR}[r_t] \leftarrow 0^{31} \parallel 1$
 else
 $\mathbf{GPR}[r_t] \leftarrow 0^{32}$
 endif

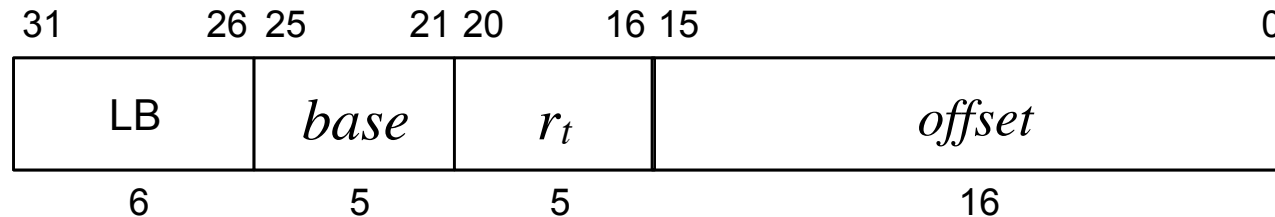


ロード・ストア命令

- LB
- LW
- SB
- SW



load byte

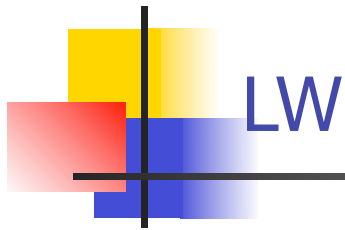


命令書式 **lb** *r_t*, *offset*(*base*)

説明 16ビットオフセットを符号拡張して汎用レジスタ*base*の内容に加算し、
32ビット無符号有効アドレスを生成。有効アドレスで指定するメモリ位置のバイトの内容を符号拡張し汎用レジスタ*r_t*にロード

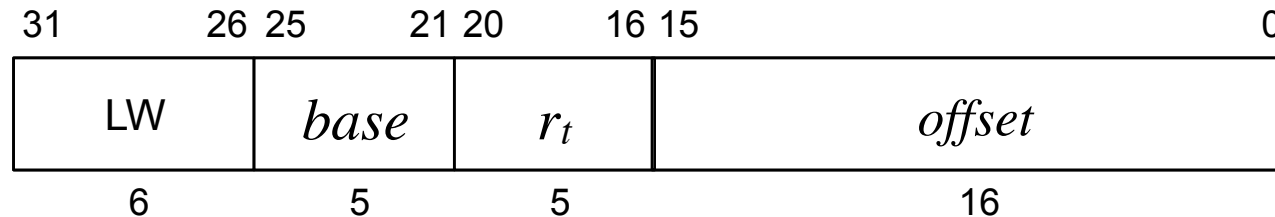
オペレーション

T: $\text{address} \leftarrow ((\text{offset}_{15})^{16} \parallel \text{offset}_{15..0}) + \mathbf{GPR}[\text{base}]$
 $\text{data} \leftarrow \mathbf{MEM}[\text{address}_{31..2} \parallel 0^2]$
 $\text{loc} \leftarrow \text{address}_{1..0}$
T + 1: $\mathbf{GPR}[\text{r}_t] \leftarrow (\text{data}_{31-(8 * \text{loc})}^{24} \parallel \text{data}_{(31-(8 * \text{loc}))..(24-(8 * \text{loc}))})$



LW

load word

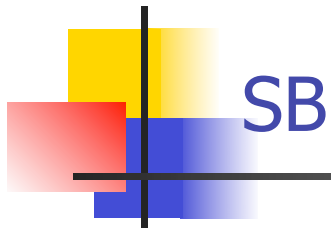


命令書式 `lw rt, offset(base)`

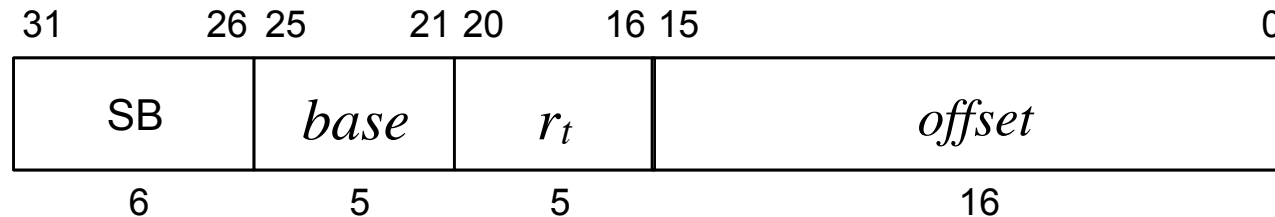
説明 16ビットオフセットを符号拡張して汎用レジスタ*base*の内容に加算し、
32ビット無符号有効アドレスを生成。有効アドレスで指定するメモリ位置のワードの内容を汎用レジスタ*r_t*にロード

オペレーション

T: $\text{address} \leftarrow ((\text{offset}_{15})^{16} \parallel \text{offset}_{15..0}) + \mathbf{GPR}[\text{base}]$
 $\text{data} \leftarrow \mathbf{MEM}[\text{address}_{31..2} \parallel 0^2]$
T + 1: $\mathbf{GPR}[\text{r}_t] \leftarrow \text{data}$



store byte

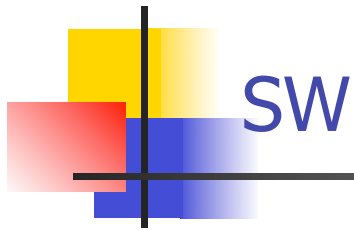


命令書式 **sb** *r_t*, *offset*(*base*)

説明 16ビットオフセットを符号拡張して汎用レジスタ*base*の内容に加算し、
32ビット無符号有効アドレスを生成。汎用レジスタ*r_t*の最下位バイトを有効アドレスで指定するメモリ位置に格納

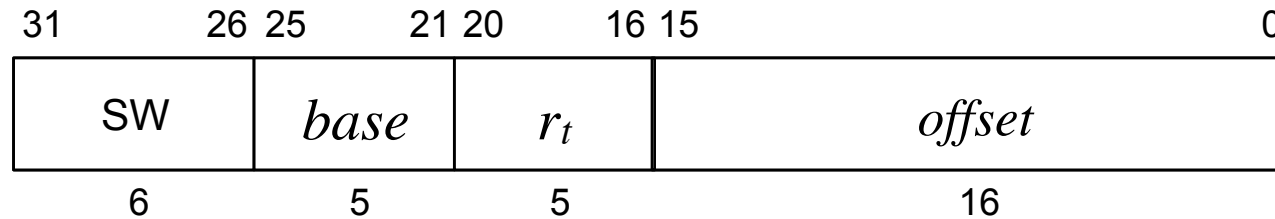
オペレーション

T: $\text{address} \leftarrow ((\text{offset}_{15})^{16} \parallel \text{offset}_{15..0}) + \mathbf{GPR}[\text{base}]$
 $\text{loc} \leftarrow \text{address}_{1..0}$
 $\text{data} \leftarrow \mathbf{GPR}[\text{r}_t]_{(7+8*\text{loc})..0} \parallel 0^{24-(8*\text{loc})}$
T + 1: $\mathbf{MEM}[\text{address}_{31..2} \parallel 0^2] \leftarrow \text{data}$



SW

store word



命令書式 SW $r_t, \text{offset}(base)$

説明 16ビットオフセットを符号拡張して汎用レジスタ $base$ の内容に加算し、
32ビット無符号有効アドレスを生成。汎用レジスタ r_t の内容を有効アドレスで指定するメモリ位置に格納

オペレーション

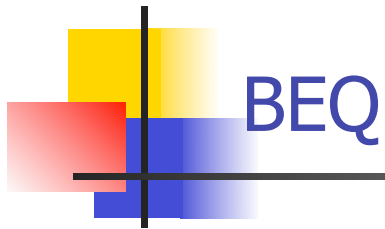
T: $\text{address} \leftarrow ((\text{offset}_{15})^{16} \parallel \text{offset}_{15..0}) + \mathbf{GPR}[base]$
 $\text{data} \leftarrow \mathbf{GPR}[r_t]$

T + 1: $\mathbf{MEM}[\text{address}_{31..2} \parallel 0^2] \leftarrow \text{data}$

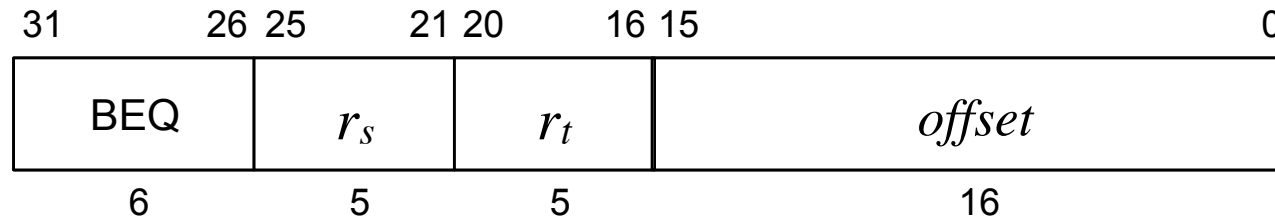


条件分岐命令

- BEQ
- BNE



branch on equal

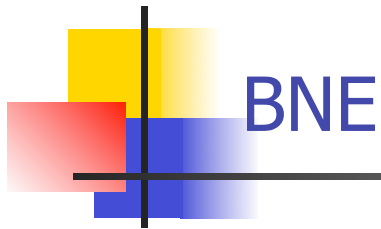


命令書式 `beq` $r_t, r_s, offset$

説明 16ビットのオフセットを2ビット左詰めして32ビットに符号拡張した値を遅延スロット内の命令のアドレスに加算して分岐ターゲット・アドレスを計算。汎用レジスタ r_s の内容と r_t の内容を比較し、2つの値が等しい場合には、1命令遅れでターゲット・アドレスに分岐

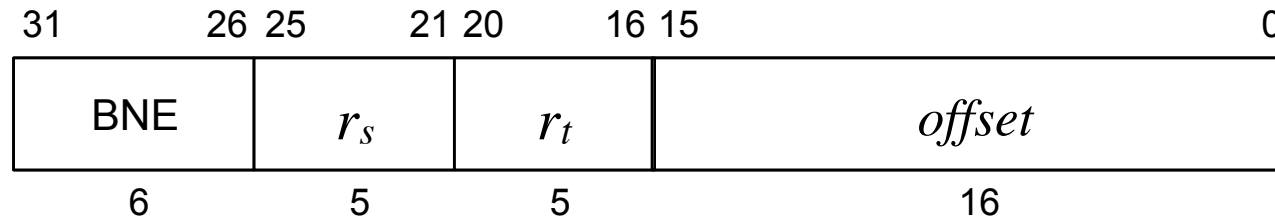
オペレーション

T: $target \leftarrow (offset_{15})^{14} \parallel offset \parallel 0^2$
 $condition \leftarrow (GPR[r_s] = GPR[r_t]) ? true : false$
T + 1: if condition then
 PC \leftarrow **PC** + target
 endif



BNE

branch on not equal



命令書式 **bne** $r_t, r_s, offset$

説明 16ビットのオフセットを2ビット左詰めして32ビットに符号拡張した値を遅延スロット内の命令のアドレスに加算して分岐ターゲット・アドレスを計算。汎用レジスタ r_s の内容と r_t の内容を比較し、2つの値が等しくない場合には、1命令遅れでターゲット・アドレスに分岐

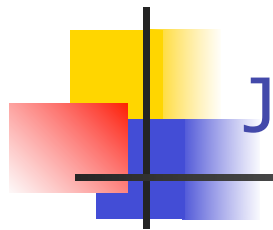
オペレーション

T: $target \leftarrow (offset_{15})^{14} \parallel offset \parallel 0^2$
 $condition \leftarrow (\mathbf{GPR}[r_s] \neq \mathbf{GPR}[r_t]) ? \text{true} : \text{false}$
T + 1: if condition then
 PC \leftarrow **PC** + target
 endif



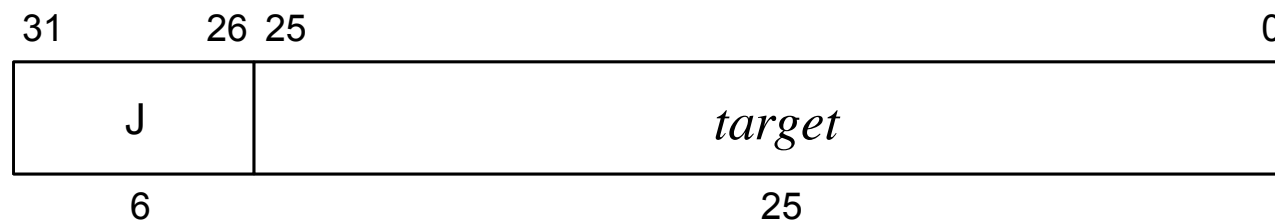
ジャンプ命令

- J
- JAL
- JR
- JALR



J

jump



命令書式 **j** *target*

説明 26ビットターゲット・アドレスを2ビット左詰めして、現在のプログラム・カウンタの上位4ビットと合成しジャンプ先のアドレスを計算。計算されたアドレスに1命令遅れで無条件に分岐。

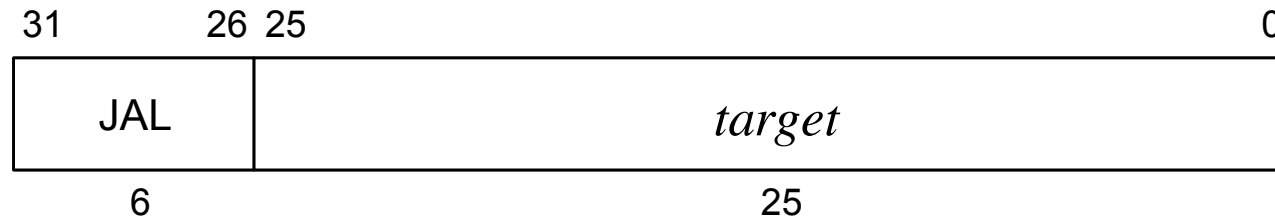
オペレーション

T: $\text{temp} \leftarrow \mathbf{PC}_{31..28} \parallel \text{target} \parallel 0^2$

T + 1: $\mathbf{PC} \leftarrow \text{temp}$



jump and link

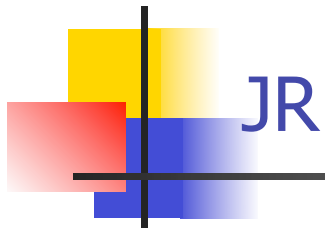


命令書式 `jal` *target*

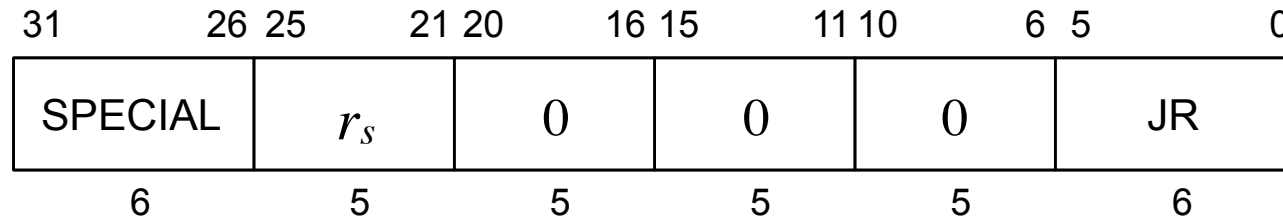
説明 26ビットターゲット・アドレスを2ビット左詰めして、現在のプログラム・カウンタの上位4ビットと合成しジャンプ先のアドレスを計算。計算されたアドレスに1命令遅れで無条件に分岐。遅延スロットの後の命令のアドレスはリンクレジスタ（\$31, \$ra）に格納。

オペレーション

T: $\text{temp} \leftarrow \text{PC}_{31..28} \parallel \text{target} \parallel 0^2$
 $\text{GPR}[31] \leftarrow \text{PC} + 8$
T + 1: $\text{PC} \leftarrow \text{temp}$



jump register



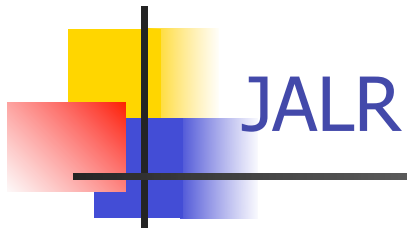
命令書式 `jr` r_s

説明 1命令遅れで汎用レジスタ r_s に格納されたアドレスへ無条件で分岐.

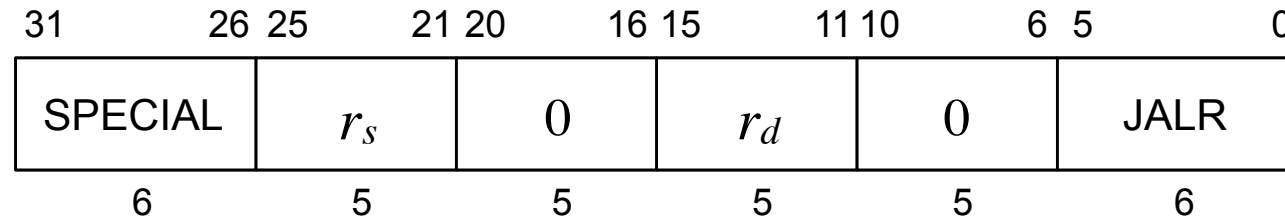
オペレーション

T: `temp` \leftarrow **GPR** $[r_s]$

T + 1: **PC** \leftarrow `temp`



jump and link register



命令書式 `jalr` r_d, r_s

説明 1命令遅れで汎用レジスタ r_s に格納されたアドレスへ無条件で分岐. 遅延スロットの後の命令のアドレスを汎用レジスタ r_d に格納. r_d が省略されている場合, r_d のデフォルト値は31.

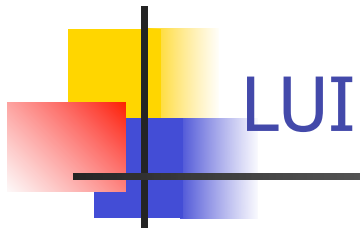
オペレーション

T: $\text{temp} \leftarrow \mathbf{GPR}[r_s]$
 $\mathbf{GPR}[r_d] \leftarrow \mathbf{PC} + 8$
T + 1: $\mathbf{PC} \leftarrow \text{temp}$

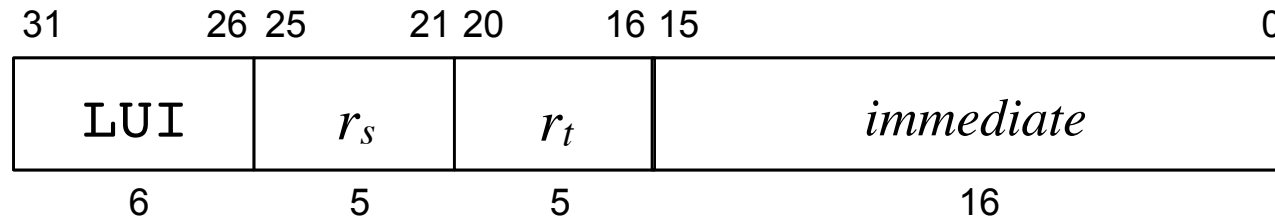


定数操作命令

- LUI



load upper immediate



命令書式 `lui` r_t , *immediate*

説明 16ビット即値を16ビット左詰めにして0の16ビットと合成. この結果を汎用レジスタ r_t に格納.

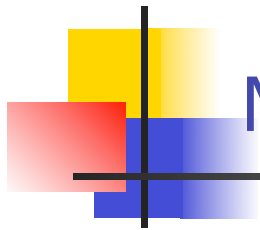
オペレーション

T: **GPR**[r_t] \leftarrow *immediate* || 0¹⁶



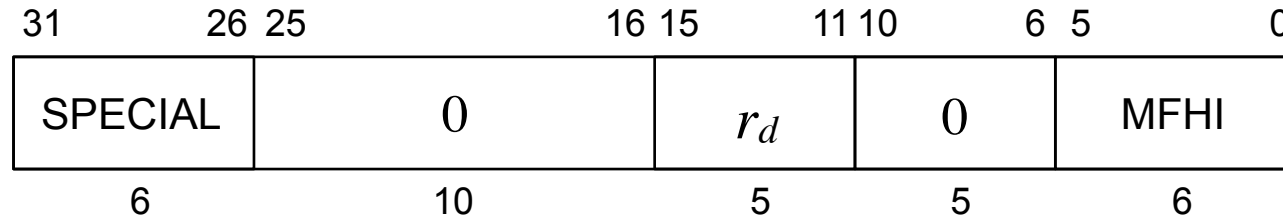
転送命令

- MFHI
- MFLO



MFHI

move from HI



命令書式 `mfhi` r_d

説明 特殊レジスタHIの内容を汎用レジスタ r_d にロード.

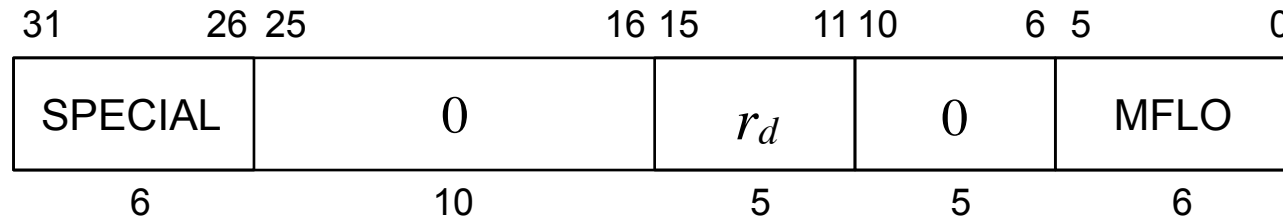
割込後で正しいオペレーションを保証するには, MFHI命令に続く2つの命令でHIレジスタを修正する命令 (MULT, MULTU, DIV など) を使用しないようにする.

オペレーション

T: **GPR**[r_d] \leftarrow **HI**



move from LO



命令書式 `mflo` r_d

説明 特殊レジスタLOの内容を汎用レジスタ r_d にロード.

割込後で正しいオペレーションを保証するには, MFLO命令に続く2つの命令でLOレジスタを修正する命令 (MULT, MULTU, DIV など) を使用しないようにする.

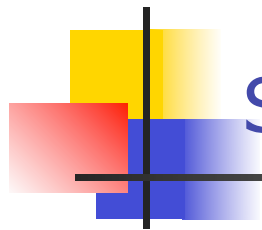
オペレーション

T: **GPR**[r_d] \leftarrow **LO**



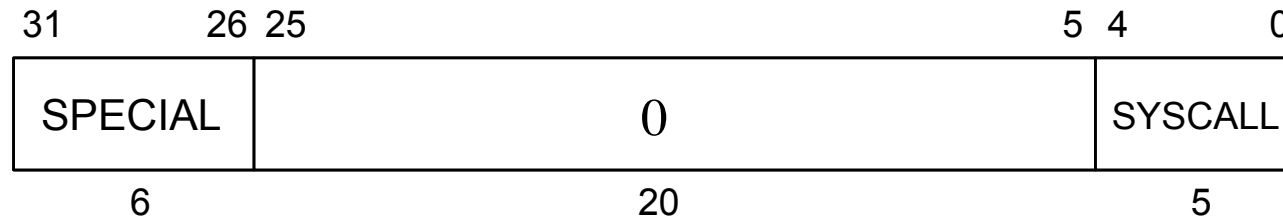
その他

- SYSCALL



SYSCALL

system call

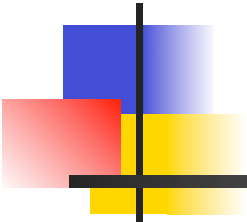


命令書式 `syscall`

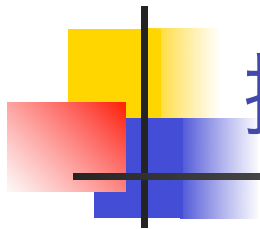
説明 システムコール・トラップを発生させ、制御を無条件で例外ハンドラに移す。

オペレーション

T: **PC** ← ExceptionHandler



コードの割当



操作コード(Opcode)

Opcode

28 .. 26 (下位3ビット)

31 .. 29 (上位3ビット)

	0	1	2	3	4	5	6	7
0	SPECIAL		J	JAL	BEQ	BNE		
1	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2								
3								
4	LB			LW				
5	SB			SW				
6								
7								



SPECIALコード

SPECIAL

2 .. 0 (下位3ビット)

5 .. 3 (上位3ビット)

	0	1	2	3	4	5	6	7
0	SLL		SRL	SRA	SLLV		SRLV	SRAV
1	JR	JALR			SYSCALL			
2	MFHI		MFLO					
3	MULT	MULTU	DIV	DIVU				
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5			SLT	SLTU				
6								
7								

