Instructions

T	RIJ	Syntax			EC
		abs.d	$f2,$f4	Floating point absolute value double precision : Set $f2 to absolute value of $f4, double precision	
		abs.s	$f0,$f1	Floating point absolute value single precision : Set $f0 to absolute value of $f1, single precision	
x	R	add	$t1,$t2,$t3	Addition with overflow : set $t1 to ($t2 plus $t3)	
5	F,R	add.d	$f2,$f4,$f6	Floating point addition double precision : Set $f2 to double-precision floating point value of $f4 plus $f6	fd = fs + ft
5	F,R	add.s	$f0,$f1,$f3	Floating point addition single precision : Set $f0 to single-precision floating point value of $f1 plus $f3	fd = fs + ft
x	I	addi	$t1,$t2,-100	Addition immediate with overflow : set $t1 to ($t2 plus signed 16-bit immediate)	
2	I	addiu	$t1,$t2,-100	Addition immediate unsigned without overflow : set $t1 to ($t2 plus signed 16-bit immediate), no overflow	rd = rs + SignExt(imm16)
2	R	addu	$t1,$t2,$t3	Addition unsigned without overflow : set $t1 to ($t2 plus $t3), no overflow	rd = rs + rd
3	R	and	$t1,$t2,$t3	Bitwise AND : Set $t1 to bitwise AND of $t2 and $t3	rd = rs AND rd
3	I	andi	$t1,$t2,100	Bitwise AND immediate : Set $t1 to bitwise AND of $t2 and zero-extended 16-bit immediate	rd = rs AND ZeroExt(imm16)
		bc1f	1,label	Branch if specified FP condition flag false (BC1F, not BCLF) : If Coprocessor 1 condition flag specified by immediate is false (zero) then branch to statement at label's address	
5	F,I	bc1f	label	Branch if FP condition flag 0 false (BC1F, not BCLF) : If Coprocessor 1 condition flag 0 is false (zero) then branch to statement at label's address	
		bc1t	1,label	Branch if specified FP condition flag true (BC1T, not BCLT) : If Coprocessor 1 condition flag specified by immediate is true (one) then branch to statement at label's address	
5	F,I	bc1t	label	Branch if FP condition flag 0 true (BC1T, not BCLT) : If Coprocessor 1 condition flag 0 is true (one) then branch to statement at label's address	
3	I	beq	$t1,$t2,label	Branch if equal : Branch to statement at label's address if $t1 and $t2 are equal	
		bgez	$t1,label	Branch if greater than or equal to zero : Branch to statement at label's address if $t1 is greater than or equal to zero	
		bgezal	$t1,label	Branch if greater then or equal to zero and link : If $t1 is greater than or equal to zero, then set $ra to the Program Counter and branch to statement at label's address	
		bgtz	$t1,label	Branch if greater than zero : Branch to statement at label's address if $t1 is greater than zero	
		blez	$t1,label	Branch if less than or equal to zero : Branch to statement at label's address if $t1 is less than or equal to zero	
		bltz	$t1,label	Branch if less than zero : Branch to statement at label's address if $t1 is less than zero	
		bltzal	$t1,label	Branch if less than zero and link : If $t1 is less than or equal to zero, then set $ra to the Program Counter and branch to statement at label's address	
3	I	bne	$t1,$t2,label	Branch if not equal : Branch to statement at label's address if $t1 and $t2 are not equal	
8		break		Break execution : Terminate program execution with exception	
		break	100	Break execution with code : Terminate program execution with specified exception code	
5	F,R	c.eq.d	$f2,$f4	Compare equal double precision : If $f2 is equal to $f4 (double-precision), set Coprocessor 1 condition flag 0 true else set it false	
		c.eq.d	1,$f2,$f4	Compare equal double precision : If $f2 is equal to $f4 (double-precision), set Coprocessor 1 condition flag specified by immediate to true else set it to false	
5	F,R	c.eq.s	$f0,$f1	Compare equal single precision : If $f0 is equal to $f1, set Coprocessor 1 condition flag 0 true else set it false	
		c.eq.s	1,$f0,$f1	Compare equal single precision : If $f0 is equal to $f1, set Coprocessor 1 condition flag specied by immediate to true else set it to false	
5	FR	c.le.d	$f2,$f4	Compare less or equal double precision : If $f2 is less than or equal to $f4 (double-precision), set Coprocessor 1 condition flag 0 true else set it false	
		c.le.d	1,$f2,$f4	Compare less or equal double precision : If $f2 is less than or equal to $f4 (double-precision), set Coprocessor 1 condition flag specfied by immediate true else set it false	
5	F,R	c.le.s	$f0,$f1	Compare less or equal single precision : If $f0 is less than or equal to $f1, set Coprocessor 1 condition flag 0 true else set it false	
		c.le.s	1,$f0,$f1	Compare less or equal single precision : If $f0 is less than or equal to $f1, set Coprocessor 1 condition flag specified by immediate to true else set it to false	
5	F,R	c.lt.d	$f2,$f4	Compare less than double precision : If $f2 is less than $f4 (double-precision), set Coprocessor 1 condition flag 0 true else set it false	
		c.lt.d	1,$f2,$f4	Compare less than double precision : If $f2 is less than $f4 (double-precision), set Coprocessor 1 condition flag specified by immediate to true else set it to false	

5	F,R	c.lt.s	$f0,$f1	Compare less than single precision : If $f0 is less than $f1, set Coprocessor 1 condition flag 0 true else set it false
		c.lt.s	1,$f0,$f1	Compare less than single precision : If $f0 is less than $f1, set Coprocessor 1 condition flag specified by immediate to true else set it to false
		ceil.w.d	$f1,$f2	Ceiling double precision to word : Set $f1 to 32-bit integer ceiling of double-precision float in $f2
		ceil.w.s	$f0,$f1	Ceiling single precision to word : Set $f0 to 32-bit integer ceiling of single-precision float in $f1
		clo	$t1,$t2	Count number of leading ones : Set $t1 to the count of leading one bits in $t2 starting at most significant bit position
		clz	$t1,$t2	Count number of leading zeroes : Set $t1 to the count of leading zero bits in $t2 starting at most significant bit positio
		cvt.d.s	$f2,$f1	Convert from single precision to double precision : Set $f2 to double precision equivalent of single precision value in $f1
		cvt.d.w	$f2,$f1	Convert from word to double precision : Set $f2 to double precision equivalent of 32-bit integer value in $f1
		cvt.s.d	$f1,$f2	Convert from double precision to single precision : Set $f1 to single precision equivalent of double precision value in $f2
		cvt.s.w	$f0,$f1	Convert from word to single precision : Set $f0 to single precision equivalent of 32-bit integer value in $f2 [$f1]
		cvt.w.d	$f1,$f2	Convert from double precision to word : Set $f1 to 32-bit integer equivalent of double precision value in $f2
		cvt.w.s	$f0,$f1	Convert from single precision to word : Set $f0 to 32-bit integer equivalent of single precision value in $f1
5	R	div	$t1,$t2	Division with overflow : Divide $t1 by $t2 then set LO to quotient and HI to remainder (use mfhi to access HI, mflo to access LO)
5	F,R	div.d	$f2,$f4,$f6	Floating point division double precision : Set $f2 to double-precision floating point value of $f4 divided by $f6
5	F,R	div.s	$f0,$f1,$f3	Floating point division single precision : Set $f0 to single-precision floating point value of $f1 divided by $f3
5	R	divu	$t1,$t2	Division unsigned without overflow : Divide unsigned $t1 by $t2 then set LO to quotient and HI to remainder (use mfhi to access HI, mflo to access LO)
8		eret		Exception return : Set Program Counter to Coprocessor 0 EPC register value, set Coprocessor Status register bit 1 (exception level) to zero
		floor.w.d	$f1,$f2	Floor double precision to word : Set $f1 to 32-bit integer floor of double-precision float in $f2
		floor.w.s	$f0,$f1	Floor single precision to word : Set $f0 to 32-bit integer floor of single-precision float in $f1
3	J	j	target	Jump unconditionally : Jump to statement at target address
3	J	jal	target	Jump and link : Set $ra to Program Counter (return address) then jump to statement at target address
		jalr	$t1	Jump and link register : Set $ra to Program Counter (return address) then jump to statement whose address is in $t1
3		jalr	$t1,$t2	Jump and link register : Set $t1 to Program Counter (return address) then jump to statement whose address is in $t2
3	J	jr	$t1	Jump register unconditionally : Jump to statement whose address is in $t1
2	I	lb	$t1,-100($t2)	Load byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
2	I	lbu	$t1,-100($t2)	Load byte unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
	I	ldc1	$f2,-100($t2)	Load double word Coprocessor 1 (FPU)) : Set $f2 to 64-bit value from effective memory doubleword address
2	I	lh	$t1,-100($t2)	Load halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
2	I	lhu	$t1,-100($t2)	Load halfword unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
		ll	$t1,-100($t2)	Load linked : Paired with Store Conditional (sc) to perform atomic read-modify-write. Treated as equivalent to Load Word (lw) because MARS does not simulate multiple processors.
2	I	lui	$t1,100	Load upper immediate : Set high-order 16 bits of $t1 to 16-bit immediate and low-order 16 bits to 0
2	I	lw	$t1,-100($t2)	Load word : Set $t1 to contents of effective memory word address
5	I	lwc1	$f1,-100($t2)	Load word into Coprocessor 1 (FPU) : Set $f1 to 32-bit value from effective memory word address
		lwl	$t1,-100($t2)	Load word left : Load from 1 to 4 bytes left-justified into $t1, starting with effective memory byte address and continuing through the low-order byte of its word
		lwr	$t1,-100($t2)	Load word right : Load from 1 to 4 bytes right-justified into $t1, starting with effective memory byte address and continuing through the high-order byte of its word
		madd	$t1,$t2	Multiply add : Multiply $t1 by $t2 then increment HI by high-order 32 bits of product, increment LO by low-order 32 bits of product (use mfhi to access HI, mflo to access LO)

		maddu	$t1,$t2	Multiply add unsigned : Multiply $t1 by $t2 then increment HI by high-order 32 bits of product, increment LO by low-order 32 bits of product, unsigned (use mfhi to access HI, mflo to access LO)
8	R	mfc0	$t1,$8	Move from Coprocessor 0 : Set $t1 to the value stored in Coprocessor 0 register $8
5	R	mfc1	$t1,$f1	Move from Coprocessor 1 (FPU) : Set $t1 to value in Coprocessor 1 register $f1
4	R	mfhi	$t1	Move from HI register : Set $t1 to contents of HI (see multiply and divide operations)
4	R	mflo	$t1	Move from LO register : Set $t1 to contents of LO (see multiply and divide operations)
		mov.d	$f2,$f4	Move floating point double precision : Set double precision $f2 to double precision value in $f4
5		mov.s	$f0,$f1	Move floating point single precision : Set single precision $f0 to single precision value in $f1
		movf	$t1,$t2	Move if FP condition flag 0 false : Set $t1 to $t2 if FPU (Coprocessor 1) condition flag 0 is false (zero)
		movf	$t1,$t2,1	Move if specified FP condition flag false : Set $t1 to $t2 if FPU (Coprocessor 1) condition flag specified by the immediate is false (zero)
		movf.d	$f2,$f4	Move floating point double precision : If condition flag 0 false, set double precision $f2 to double precision value in $f4
		movf.d	$f2,$f4,1	Move floating point double precision : If condition flag specified by immediate is false, set double precision $f2 to double precision value in $f4
		movf.s	$f0,$f1	Move floating point single precision : If condition flag 0 is false, set single precision $f0 to single precision value in $f1
		movf.s	$f0,$f1,1	Move floating point single precision : If condition flag specified by immediate is false, set single precision $f0 to single precision value in $f1e
		movn	$t1,$t2,$t3	Move conditional not zero : Set $t1 to $t2 if $t3 is not zero
		movn.d	$f2,$f4,$t3	Move floating point double precision : If $t3 is not zero, set double precision $f2 to double precision value in $f4
		movn.s	$f0,$f1,$t3	Move floating point single precision : If $t3 is not zero, set single precision $f0 to single precision value in $f1
		movt	$t1,$t2	Move if FP condition flag 0 true : Set $t1 to $t2 if FPU (Coprocessor 1) condition flag 0 is true (one)
		movt	$t1,$t2,1	Move if specfied FP condition flag true : Set $t1 to $t2 if FPU (Coprocessor 1) condition flag specified by the immediate is true (one)
		movt.d	$f2,$f4	Move floating point double precision : If condition flag 0 true, set double precision $f2 to double precision value in $f4
		movt.d	$f2,$f4,1	Move floating point double precision : If condition flag specified by immediate is true, set double precision $f2 to double precision value in $f4e
		movt.s	$f0,$f1	Move floating point single precision : If condition flag 0 is true, set single precision $f0 to single precision value in $f1e
		movt.s	$f0,$f1,1	Move floating point single precision : If condition flag specified by immediate is true, set single precision $f0 to single precision value in $f1e
		movz	$t1,$t2,$t3	Move conditional zero : Set $t1 to $t2 if $t3 is zero
		movz.d	$f2,$f4,$t3	Move floating point double precision : If $t3 is zero, set double precision $f2 to double precision value in $f4
		movz.s	$f0,$f1,$t3	Move floating point single precision : If $t3 is zero, set single precision $f0 to single precision value in $f1
		msub	$t1,$t2	Multiply subtract : Multiply $t1 by $t2 then decrement HI by high-order 32 bits of product, decrement LO by low-order 32 bits of product (use mfhi to access HI, mflo to access LO)
		msubu	$t1,$t2	Multiply subtract unsigned : Multiply $t1 by $t2 then decrement HI by high-order 32 bits of product, decement LO by low-order 32 bits of product, unsigned (use mfhi to access HI, mflo to access LO)
8		mtc0	$t1,$8	Move to Coprocessor 0 : Set Coprocessor 0 register $8 to value stored in $t1
5		mtc1	$t1,$f1	Move to Coprocessor 1 (FPU) : Set Coprocessor 1 register $f1 to value in $t1
		mthi	$t1	Move to HI registerr : Set HI to contents of $t1 (see multiply and divide operations)
		mtlo	$t1	Move to LO register : Set LO to contents of $t1 (see multiply and divide operations)
5	F,R	mul.d	$f2,$f4,$f6	Floating point multiplication double precision : Set $f2 to double-precision floating point value of $f4 times $f6
5	F,R	mul.s	$f0,$f1,$f3	Floating point multiplication single precision : Set $f0 to single-precision floating point value of $f1 times $f3
5	R	mult	$t1,$t2	Multiplication : Set hi to high-order 32 bits, lo to low-order 32 bits of the product of $t1 and $t2 (use mfhi to access hi, mflo to access lo)
5	R	multu	$t1,$t2	Multiplication unsigned : Set HI to high-order 32 bits, LO to low-order 32 bits of the product of unsigned $t1 and $t2 (use mfhi to access HI, mflo to access LO)

		neg.d	$f2,$f4	Floating point negate double precision : Set double precision $f2 to negation of double precision value in $f4	
		neg.s	$f0,$f1	Floating point negate single precision : Set single precision $f0 to negation of single precision value in $f1	
		nop		Null operation : machine code is all zeroes	
3	R	nor	$t1,$t2,$t3	Bitwise NOR : Set $t1 to bitwise NOR of $t2 and $t3	rd = rs NOR rd = NOT (rs OR rt)
3	R	or	$t1,$t2,$t3	Bitwise OR : Set $t1 to bitwise OR of $t2 and $t3	rd = rs OR rd
2	I	ori	$t1,$t2,100	Bitwise OR immediate : Set $t1 to bitwise OR of $t2 and zero-extended 16-bit immediate	rd = rs OR ZeroExt(imm16)
		round.w.d	$f1,$f2	Round double precision to word : Set $f1 to 32-bit integer round of double-precision float in $f2	
		round.w.s	$f0,$f1	Round single precision to word : Set $f0 to 32-bit integer round of single-precision float in $f1	
2	I	sb	$t1,-100($t2)	Store byte : Store the low-order 8 bits of $t1 into the effective memory byte address	
		sc	$t1,-100($t2)	Store conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Stores $t1 value into effective address, then sets $t1 to 1 for success. Always succeeds because MARS does not simulate multiple processors.	
	R	sdc1	$f2,-100($t2)	Store double word from Coprocessor 1 (FPU)) : Store 64 bit value in $f2 to effective memory doubleword address	
3	I	sh	$t1,-100($t2)	Store halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address	
3	R	sll	$t1,$t2,10	Shift left logical : Set $t1 to result of shifting $t2 left by number of bits specified by immediate	
3		sllv	$t1,$t2,$t3	of $t3	
3	R	slt	$t1,$t2,$t3	Set less than : If $t2 is less than $t3, then set $t1 to 1 else set $t1 to 0	
3	I	slti	$t1,$t2,-100	Set less than immediate : If $t2 is less than sign-extended 16-bit immediate, then set $t1 to 1 else set $t1 to 0	
3	I	sltiu	$t1,$t2,-100	Set less than immediate unsigned : If $t2 is less than sign-extended 16-bit immediate using unsigned comparison, then set $t1 to 1 else set $t1 to 0	
3	R	sltu	$t1,$t2,$t3	Set less than unsigned : If $t2 is less than $t3 using unsigned comparision, then set $t1 to 1 else set $t1 to 0	
		sqrt.d	$f2,$f4	Square root double precision : Set $f2 to double-precision floating point square root of $f4	
		sqrt.s	$f0,$f1	Square root single precision : Set $f0 to single-precision floating point square root of $f1	
3	R	sra	$t1,$t2,10	Shift right arithmetic : Set $t1 to result of sign-extended shifting $t2 right by number of bits specified by immediate	
3	R	srav	$t1,$t2,$t3	Shift right arithmetic variable : Set $t1 to result of sign-extended shifting $t2 right by number of bits specified by value in low-order 5 bits of $t3	
3	R	srl	$t1,$t2,10	Shift right logical : Set $t1 to result of shifting $t2 right by number of bits specified by immediate	
3	R	srlv	$t1,$t2,$t3	Shift right logical variable : Set $t1 to result of shifting $t2 right by number of bits specified by value in low-order 5 bits of $t3	
x		sub	$t1,$t2,$t3	Subtraction with overflow : set $t1 to ($t2 minus $t3)	
5	F,R	sub.d	$f2,$f4,$f6	Floating point subtraction double precision : Set $f2 to double-precision floating point value of $f4 minus $f6	
5	F,R	sub.s	$f0,$f1,$f3	Floating point subtraction single precision : Set $f0 to single-precision floating point value of $f1 minus $f3	
2	R	subu	$t1,$t2,$t3	Subtraction unsigned without overflow : set $t1 to ($t2 minus $t3), no overflow	
3	I	sw	$t1,-100($t2)	Store word : Store contents of $t1 into effective memory word address	
5	R	swc1	$f1,-100($t2)	Store word from Coprocesor 1 (FPU) : Store 32 bit value in $f1 to effective memory word address	
		swl	$t1,-100($t2)	Store word left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective byte address and continuing through the low-order byte of its word	
		swr	$t1,-100($t2)	Store word right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective byte address and continuing through that byte address	
8		syscall		Issue a system call : Execute the system call specified by value in $v0	
8		teq	$t1,$t2	Trap if equal : Trap if $t1 is equal to $t2	
		teqi	$t1,-100	Trap if equal to immediate : Trap if $t1 is equal to sign-extended 16 bit immediate	
		tge	$t1,$t2	Trap if greater or equal : Trap if $t1 is greater than or equal to $t2	
		tgei	$t1,-100	Trap if greater than or equal to immediate : Trap if $t1 greater than or equal to sign-extended 16 bit immediate	
		tgeiu	$t1,-100	Trap if greater or equal to immediate unsigned : Trap if $t1 greater than or equal to sign-extended 16 bit immediate, unsigned comparison	
		tgeu	$t1,$t2	Trap if greater or equal unsigned : Trap if $t1 is greater than or equal to $t2 using unsigned comparision	
8		tlbwr		TLB write random	

		tlt	$t1,$t2	Trap if less than: Trap if $t1 less than $t2	
		tlti	$t1,-100	Trap if less than immediate : Trap if $t1 less than sign-extended 16-bit immediate	
		tltiu	$t1,-100	Trap if less than immediate unsigned : Trap if $t1 less than sign-extended 16-bit immediate, unsigned comparison	
		tltu	$t1,$t2	Trap if less than unsigned : Trap if $t1 less than $t2, unsigned comparison	
		tne	$t1,$t2	Trap if not equal : Trap if $t1 is not equal to $t2	
		tnei	$t1,-100	Trap if not equal to immediate : Trap if $t1 is not equal to sign-extended 16 bit immediate	
		trunc.w.d	$f1,$f2	Truncate double precision to word : Set $f1 to 32-bit integer truncation of double-precision float in $f2	
		trunc.w.s	$f0,$f1	Truncate single precision to word : Set $f0 to 32-bit integer truncation of single-precision float in $f1	
3	R	xor	$t1,$t2,$t3	Bitwise XOR (exclusive OR) : Set $t1 to bitwise XOR of $t2 and $t3	rd = rs XOR rd
3	I	xori	$t1,$t2,100	Bitwise XOR immediate : Set $t1 to bitwise XOR of $t2 and zero-extended 16-bit immediate	rd = rs XOR ZeroExt(imm16)

Extended (pseudo) Instructions

```
b       label               Branch : Branch to statement at label unconditionally
                               bgez   $zero, label
bge     $t1,$t2,label       Branch if Greater or Equal : Branch to statement at label if $t1 is greater or equal to $t2
                               slt    $at, $t1, $t2           # $t1,$t2 + beq
                               beq    $at, $zero, label
bgeu    $t1,$t2,label       Branch if Greater or Equal Unsigned : Branch to statement at label if $t1 is greater or equal to $t2 (unsigned compare)
                               sltu   $at, $t1, $t2
                               beq    $at, $zero, label
bgt     $t1,$t2,label       Branch if Greater Than : Branch to statement at label if $t1 is greater than $t2
                               slt    $at, $t2, $t1           # $t2,$t1 + bne
                               bne    $at, $zero, label
bgtu    $t1,$t2,label       Branch if Greater Than Unsigned: Branch to statement at label if $t1 is greater than $t2 (unsigned compare)
                               sltu   $at, $t2, $t1
                               bne    $at, $zero, label
ble     $t1,$t2,label       Branch if Less or Equal : Branch to statement at label if $t1 is less than or equal to $t2
                               slt    $at, $t2, $t1           # $t2,$t1 + beq
                               beq    $at, $zero, label
bleu    $t1,$t2,label       Branch if Less or Equal Unsigned : Branch to statement at label if $t1 is less than or equal to $t2 (unsigned compare)
                               sltu   $at, $t2, $t1
                               beq    $at, $zero, label
blt     $t1,$t2,label       Branch if Less Than : Branch to statement at label if $t1 is less than $t2
                               slt    $at, $t1, $t2           # $t1,$t2 + bne
                               bne    $at, $zero, label
bltu    $t1,$t2,label       Branch if Less Than Unsigned : Branch to statement at label if $t1 is less than $t2
                               sltu   $at, $t1, $t2
                               bne    $at, $zero, label
la      $t1,-100            Load Address : Set $t1 to 16-bit immediate (sign-extended)
                               addiu  $t1, $zero, 0xffff ff9c
la      $t1,100             Load Address : Set $t1 to 16-bit immediate (zero-extended)
                               addiu  $t1, $zero, 0x0000 0064
la      $t1,100000          Load Address : Set $t1 to 32-bit immediate
                               lui    $at, 0x0000 0001        # hi(imm32)
                               ori    $t1, 0x0000 86a0        # lo(imm32)
la      $t1,label           Load Address : Set $t1 to label's address
                               lui    $t1, label
la      $t1,label+100000    Load Address : Set $t1 to sum (of label's address and 32-bit immediate)
                               lui    $at, 0x0000 0002        # hi(label+imm32)
                               ori    $t1, 0x0000 86a0        # lo(label+imm32)
li      $t1,-100            Load Immediate : Set $t1 to 16-bit immediate (sign-extended)
                               addiu  $t1, $zero, 0xffff ff9c # li == la
li      $t1,100             Load Immediate : Set $t1 to unsigned 16-bit immediate (zero-extended)
                               addiu  $t1, $zero, 0x0000 0064
li      $t1,100000          Load Immediate : Set $t1 to 32-bit immediate
                               lui    $at, 0x0000 0001
                               ori    $t1, 0x0000 86a0
move    $t1,$t2             MOVE : Set $t1 to contents of $t2
                               addu   $t1, $zero, $t2
not     $t1,$t2             Bitwise NOT (bit inversion)
                               nor    $t1, $zero, $t2
```

Directives

`.align`	Align next data item on specified byte boundary (0=byte, 1=half, 2=word, 3=double)
`.ascii`	Store the string in the Data segment but do not add null terminator
`.asciiz`	Store the string in the Data segment and add null terminator
`.byte`	Store the listed value(s) as 8 bit bytes
`.data`	Subsequent items stored in Data segment at next available address
`.double`	Store the listed value(s) as double precision floating point
`.dword`	Store the listed value(s) as 64 bit dwords on dword boundary
`.end_macro`	End macro definition. See .macro
`.eqv`	Substitute second operand for first. First operand is symbol, second operand is expression (like #define)
`.extern`	Declare the listed label and byte length to be a global data field
`.float`	Store the listed value(s) as single precision floating point
`.globl`	Declare the listed label(s) as global to enable referencing from other files
`.half`	Store the listed value(s) as 16 bit halfwords on halfword boundary
`.include`	Insert the contents of the specified file. Put filename in quotes.
`.kdata`	Subsequent items stored in Kernel Data segment at next available address
`.ktext`	Subsequent items (instructions) stored in Kernel Text segment at next available address
`.macro`	Begin macro definition. See .end_macro
`.set`	Set assembler variables. Currently ignored but included for SPIM compatability
`.space`	Reserve the next specified number of bytes in Data segment
`.text`	Subsequent items (instructions) stored in Text segment at next available address
`.word`	Store the listed value(s) as 32 bit words on word boundary