```
add a, b, c     # The sum of b and c is placed in a
add a, a, d     # The sum of b, c, and d is now in a
add a, a, e     # The sum of b, c, d, and e is now in a
```

```
add t0,g,h # temporary variable t0 contains g + h
```

```
add t1,i,j # temporary variable t1 contains i + j
```

```
sub f,t0,t1 # f gets t0 - t1, which is (g + h) - (i + j)
```

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three register operands
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three register operands
	add immediate	`addi $s1,$s2,20`	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	`lw $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	`lh $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	`lhu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	`lb $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	`lbu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	`sb $s1,20($s2)`	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	`ll $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	`sc $s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	`lui $s1,20`	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	`andi $s1,$s2,20`	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,20`	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Shift left by constant
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to $ra	For switch, procedure return
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	For procedure call

p.64

```
lw $t0, AddrConstant4($s1)   # $t0 = constant 4
add $s3,$s3,$t0              # $s3 = $s3 + $t0 ($t0 == 4)
```

What is the decimal value of this 32-bit two's complement number?

$$1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1100_{two}$$

exemple Complement 2

Substituting the number's bit values into the formula above:

$$(1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \ldots + (1 \times 2^{1}) + (0 \times 2^{1}) + (0 \times 2^{0})$$
$$= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^{2} + 0 + 0$$
$$= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten}$$
$$= -4_{ten}$$

$zero 0	$at Reservé pour pseudo-instructions	$v0 Valeur de retour	$v1 Valeur de retour
$a0 Argument	$a1 Argument	$a2 Argument	$a3 Argument
$t0 Temporaire	$t1 Temporaire	$t2 Temporaire	$t3 Temporaire
$t4 Temporaire	$t5 Temporaire	$t6 Temporaire	$t7 Temporaire
$s0 Enregistré	$s1 Enregistré	$s2 Enregistré	$s3 Enregistré
$s4 Enregistré	$s5 Enregistré	$s6 Enregistré	$s7 Enregistré
$t8 Temporaire	$t9 Temporaire	$k0 Réservé au système d'opération	$k1 Réservé au système d'opération
$gp Pointeur des données globales	$sp Pointeur de la pile	$fp Pointeur du cadre	$ra Adresse de retour

```
1  .data   0x10010000
2                          # donnees
3  x: .byte 1, 2
4          .align 2
5  r: .word 0              # resultat
6
7  .text   0x400000
8  .globl main
9
10 main:
11         la      $t0, x
12         lb      $s0, 0($t0)
13         lb      $s1, 1($t0)
14         add     $t0, $s0, $s1
15         la      $t1, r
16         sw      $t0, 0($t1)
17
18         ori     $v0, $0,   0xA
19         syscall
```