ARITHMETIC, LOGIC, SHIFTING

MIPS Assembly Programming

Arithmetic, Logic, Shifting Instructions

MIPS Instruction Set

1. Arithmetic, Logic, and Shifting Instructions
2. Conditional Branch Instructions
3. Load and Store Instructions
4. Function Call Instructions

1. Arithmetic, Logic, and Shifting Instructions

4.4.1 Arithmetic Instructions

	Op	Operands	Description
○	`abs`	*des, src1*	*des* gets the absolute value of *src1*.
	`add(u)`	*des, src1, src2*	*des* gets *src1* + *src2*.
	`and`	*des, src1, src2*	*des* gets the bitwise and of *src1* and *src2*.
	`div(u)`	*src1, reg2*	Divide *src1* by *reg2*, leaving the quotient in register `lo` and the remainder in register `hi`.
○	`div(u)`	*des, src1, src2*	*des* gets *src1* / *src2*.
○	`mul`	*des, src1, src2*	*des* gets *src1* × *src2*.
○	`mulo`	*des, src1, src2*	*des* gets *src1* × *src2*, with overflow.
	`mult(u)`	*src1, reg2*	Multiply *src1* and *reg2*, leaving the low-order word in register `lo` and the high-order word in register `hi`.
○	`neg(u)`	*des, src1*	*des* gets the negative of *src1*.
	`nor`	*des, src1, src2*	*des* gets the bitwise logical **nor** of *src1* and *src2*.
○	`not`	*des, src1*	*des* gets the bitwise logical negation of *src1*.
	`or`	*des, src1, src2*	*des* gets the bitwise logical **or** of *src1* and *src2*.
○	`rem(u)`	*des, src1, src2*	*des* gets the remainder of dividing *src1* by *src2*.
○	`rol`	*des, src1, src2*	*des* gets the result of rotating left the contents of *src1* by *src2* bits.
○	`ror`	*des, src1, src2*	*des* gets the result of rotating right the contents of *src1* by *src2* bits.
	`sll`	*des, src1, src2*	*des* gets *src1* shifted left by *src2* bits.
	`sra`	*des, src1, src2*	Right shift arithmetic.
	`srl`	*des, src1, src2*	Right shift logical.
	`sub(u)`	*des, src1, src2*	*des* gets *src1* - *src2*.
	`xor`	*des, src1, src2*	*des* gets the bitwise exclusive **or** of *src1* and *src2*.

pseudo-instructions

`l`oading an `i`mmediate value

- `li` is a pseudo-instruction that loads an immediate value into a register
- Pseudo-instructions are only understood by the MIPS assembler but not by the CPU (MIPS)
- More MIPS pseudo-instructions:
- **`blt, bgt, ble, neg, not, bge, li, la, move`**

We already used the 3 pseudo-instructions: **`li, la, move`**

`li` (load-immediate) pseudo-instruction

- `li $t0, 0x74A12`

- Which is equivalent to:
  - `lui $t0, 0x0007`
  - `ori $t0, $t0, 0x4A12`

- `lui` = load-upper-immediate (loads the upper 16 bits and the lower 16 bits with zeros)

- `ori` = or-immediate

page. 313

`li` pseudo-instruction

- `li $t0, c` → $\boxed{\texttt{ori \$t0, \$zero, c}}$
- Note that (0 ≤ c ≤ 65,535)

- `ori` = `or` `i`mmediate

`la` (**l**oad **a**ddress) pseudo-instruction

- `la $t0, message`
- Is similar to `li` pseudo-instruction.

`move` pseudo-instruction

- `move $a0, $t4`
- Is equivalent to:
- `add $a0, $t4, $0`

Addition and Subtraction

Add .. Subtract

What is the function that is implemented ?

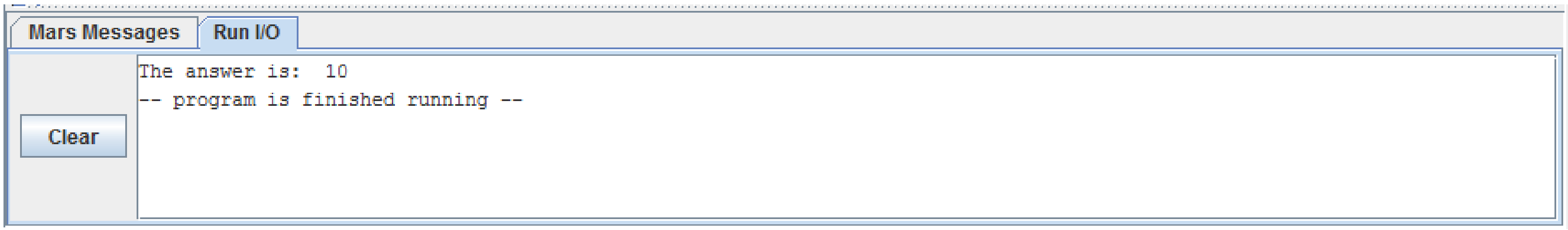
```
        .text
main:
        li      $t0, 9
        li      $t1, 6
        li      $t2, 7
        sub     $t3, $t0, $t1
        add     $t4, $t3, $t2
        la      $a0, message        # prints message
        li      $v0, 4
        syscall
        move    $a0, $t4            # move to a0 for printout
        li      $v0, 1
        syscall
        li      $v0, 10             # exit
        syscall

        .data

message:
        .asciiz "The answer is:  "
```

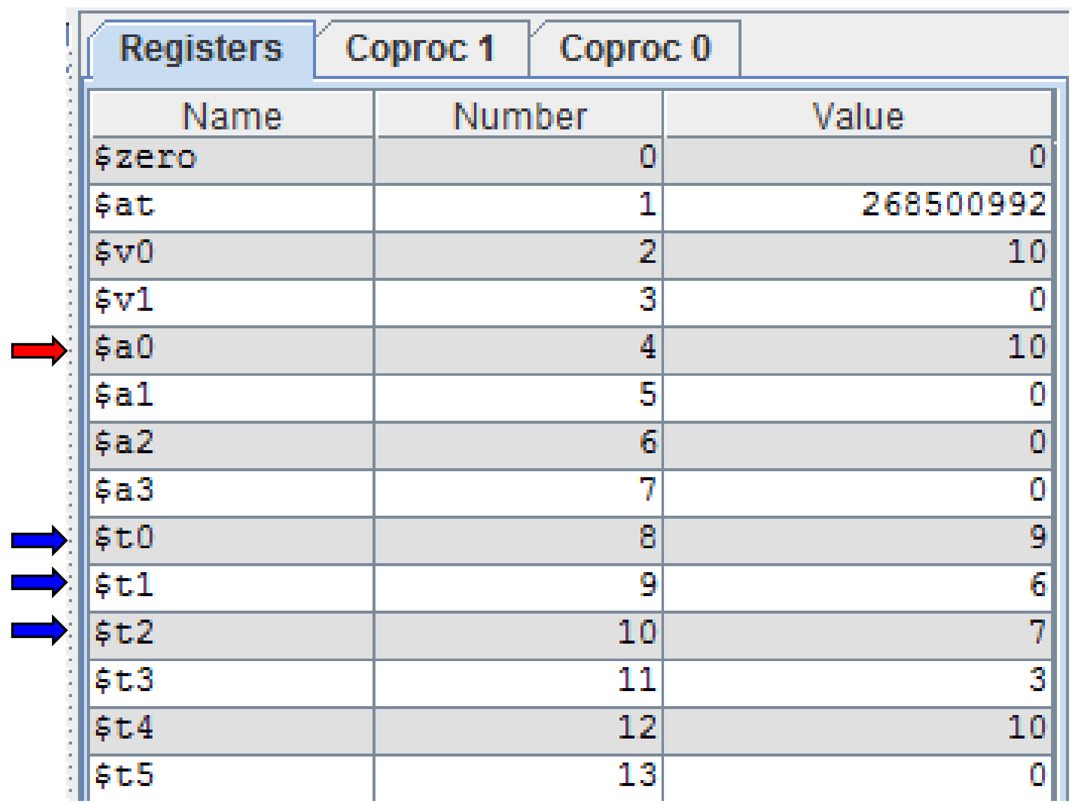
Add .. Subtract

arithmeric-Add-Sub.asm*

```
## Folder: L1/arithmetic-Add-Sub.asm
## prints out the results for  the following equation:
## (t0-t1)+t2  .... (9-6) + 7

        .text
main:
        li              $t0, 9
        li              $t1, 6
        li              $t2, 7
        sub             $t3, $t0, $t1
        add             $t4, $t3, $t2
        la              $a0, message        # prints message
        li              $v0, 4
        syscall
        move            $a0, $t4            # move to a0 for printout
        li              $v0, 1
        syscall
        li              $v0, 10             # exit
        syscall

        .data

message:
        .asciiz         "The answer is:  "
```

Assemble … GO





Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	0
$a0	4	10
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	9
$t1	9	6
$t2	10	7
$t3	11	3
$t4	12	10
$t5	13	0

Overflow/Underflow

- Overflow (Underflow) occurs whenever two positive (negative) numbers are added and the result indicates a negative (positive) number.

Sign extension

- To avoid overflow, convert a 4-bit 2's complement number into an 8-bit number
- If the number is positive number >> add 0's to left
  - 0101
  - 00000101
- If the negative number >> add 1's to left
  - 1010
  - 11111010

Multiplication…Division

MDU

- MIPS has a special Multiplication/Division Unit (MDU)
- The MDU multiplies two 32-bit numbers and stores the result in 64-bits (2 32-bit registers)
- The 2 32-bit registers are named **`mfhi`** and **`mflo`**.

MIPS: Multiply and Divide

```
Mult   $t1, $t2   # Multiply ($t1*$t2) to produce a 64-bit number (Hi,Lo)
Mfhi   $t3        # move result to special32-bit  register  Hi ($t3)
Mflo   $t4        # move result to special 32-bit register  Lo ($t4)
```

```
Div    $t1, $t2   # Div ($t1/$t2)
                  # to produce a 32 -bit  Lo [$t1 / $t2  integer quotient ]
                  # to produce a 32 -bit  Hi [$t1 % $t2  remainder]
Mfhi   $t3        # move result to special32-bit  register  Hi ($t3)
Mflo   $t4        # move result to special 32-bit register  Lo ($t4)
```

mfhi ("**m**ove **f**rom **hi**")

mflo ("**m**ove **f**rom **lo**")

Multiply

```
        .text
main:
        li      $t1, 50
        li      $t2, 2
        mult    $t1, $t2
        mflo    $t3
        la      $a0, message
        li      $v0, 4
        syscall

        move    $a0, $t3
        li      $v0, 1
        syscall

        li      $v0, 10
        syscall

        .data

message:
        .asciiz "The answer is:  "
```

What is the function that is implemented ?

Multiply

Multiply2.asm | Multiply.asm*

```
## Folder: L1\Multiply.asm
## L. Ruela, 2011
## (t1*t2) ...   (2*50)

        .text
main:
        li              $t1, 50
        li              $t2, 2
        mult            $t1, $t2
        mflo            $t3
        la              $a0, message            # prints message
        li              $v0, 4
        syscall

        move            $a0, $t3                # move to a0 for printout.
        li              $v0, 1
        syscall

        li              $v0, 10                 # exit
        syscall

        .data

message:
        .asciiz         "The answer is:  "
```

Assemble … GO

```
The answer is:  100
-- program is finished running --
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	0
$a0	4	100
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	50
$t2	10	2
$t3	11	100

Multiply in MIPS; Hi-Lo registers

- $2_{10} = 0010_2$
- $50_{10} = 110010_2$
- $2*50 = 100_{10} = 1100100_2$

$$\underbrace{00000000000000000000000000000000}_{\text{Hi}} \quad \underbrace{000000000000000000000000\ \mathbf{01100100}}_{\text{Lo}}$$

mfhi ("**m**ove **f**rom **hi**")
mflo ("**m**ove **f**rom **lo**")

Divide

```
        .text
main:
        li      $t0, 24
        li      $t1, 8
        div     $t0, $t1
        mflo    $t3
        mfhi    $t4
        la      $a0, message        # prints message
        li      $v0, 4
        syscall

        move    $a0, $t3            # move to a0 for printout
        li      $v0, 1
        syscall

        li      $v0,10              # exit
        syscall

        .data

message:
        .asciiz         "The answer is:  "

```

What is the function that is implemented ?

Divide

```
## Folder: L1\Divide.asm
## Lurdes Ruela, 2011S
## s0/s1 ... (24/8)

        .text
main:
        li          $t0, 24
        li          $t1, 8
        div         $t0, $t1
        mflo        $t3
        mfhi        $t4
        la          $a0, message     # prints message
        li          $v0, 4
        syscall

        move        $a0, $t3         # move to a0 for printout
        li          $v0, 1
        syscall

        li          $v0,10           # exit
        syscall

        .data

message:
        .asciiz             "The answer is:  "

```

Divide

```
The answer is:  3
-- program is finished running --
```

Registers	Coproc 1	Coproc 0

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	0
$a0	4	3
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	24
$t1	9	8
$t2	10	0
$t3	11	3
$t4	12	0

Divide in MIPS; Hi-Lo registers

- $24_{10} = 11000_2$
- $8_{10} = 1001_2$
- $24/8_{10} = 0_{10} + 3_{10} = 0000\ 0011$

00000000000000000000000000000000 00000000000000000000000000000011

Hi (Remainder) Lo (Quotient)

mfhi ("**m**ove **f**rom **hi**")
mflo ("**m**ove **f**rom **lo**")

Another Divide Example

```
        .text
main:
        li      $t0, 26
        li      $t1, 8
        div     $t0, $t1
        mflo    $t3
        mfhi    $t4

        li      $v0,10
        syscall
```

Another Divide Example

Divide2.asm

```
## Folder: L1\Divide2.asm
## L. Ruela, 2011S
## t0/t1 ... (26/8)

            .text
main:
            li          $t0, 26
            li          $t1, 8
            div         $t0, $t1
            mflo        $t3
            mfhi        $t4

            li          $v0,10
            syscall
```

mflo = ?

Mfhi = ?

$t3 = ?

$t4 = ?

Assemble GO

$zero	0	0
$at	1	0
$v0	2	10
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	26
$t1	9	8
$t2	10	0
$t3	11	3
$t4	12	2
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194332
hi		2
lo		3

26/8 = 3 and 2/8

Divide in MIPS; Hi-Lo registers

- $26_{10} = 11010_2$
- $8_{10} = 1001_2$
- $26/8_{10} = 2_{10} + 3_{10} = 0010$ 0011

00000000000000000000000000000010 00000000000000000000000000000011

Hi (Remainder) Lo (Quotient)

mfhi ("**m**ove **f**rom **hi**")
mflo ("**m**ove **f**rom **lo**")

Logic operations

Logical

4.4.1 Arithmetic Instructions

	Op	Operands	Description
○	`abs`	*des, src1*	*des* gets the absolute value of *src1*.
	`add(u)`	*des, src1, src2*	*des* gets *src1* + *src2*.
	`and`	*des, src1, src2*	*des* gets the bitwise and of *src1* and *src2*.
	`div(u)`	*src1, reg2*	Divide *src1* by *reg2*, leaving the quotient in register `lo` and the remainder in register `hi`.
○	`div(u)`	*des, src1, src2*	*des* gets *src1* / *src2*.
○	`mul`	*des, src1, src2*	*des* gets *src1* × *src2*.
○	`mulo`	*des, src1, src2*	*des* gets *src1* × *src2*, with overflow.
	`mult(u)`	*src1, reg2*	Multiply *src1* and *reg2*, leaving the low-order word in register `lo` and the high-order word in register `hi`.
○	`neg(u)`	*des, src1*	*des* gets the negative of *src1*.
	`nor`	*des, src1, src2*	*des* gets the bitwise logical **nor** of *src1* and *src2*.
○	`not`	*des, src1*	*des* gets the bitwise logical negation of *src1*.
	`or`	*des, src1, src2*	*des* gets the bitwise logical **or** of *src1* and *src2*.
○	`rem(u)`	*des, src1, src2*	*des* gets the remainder of dividing *src1* by *src2*.
○	`rol`	*des, src1, src2*	*des* gets the result of rotating left the contents of *src1* by *src2* bits.
○	`ror`	*des, src1, src2*	*des* gets the result of rotating right the contents of *src1* by *src2* bits.
	`sll`	*des, src1, src2*	*des* gets *src1* shifted left by *src2* bits.
	`sra`	*des, src1, src2*	Right shift arithmetic.
	`srl`	*des, src1, src2*	Right shift logical.
	`sub(u)`	*des, src1, src2*	*des* gets *src1* - *src2*.
	`xor`	*des, src1, src2*	*des* gets the bitwise exclusive **or** of *src1* and *src2*.

Logic operations …

- AND
- OR
- NOR
- XOR

and

and_2.asm

```
# Folder: L2\and_2.asm
# Ryan Preidel

    li $t4, 1                # Load two value 1 and 1 to be compared
    li $t5, 1
    and $t0, $t5, $t4        # comparing values of 0 and 0
    li $v0, 1
    move $a0, $t0            # moves value to print output
    syscall

    li  $v0, 10              # system call code for exit = 10
    syscall                  # call operating sys o exit
```

and

```
1
-- program is finished running --
```

Registers	Coproc 1	Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	10
$v1	3	0
$a0	4	1
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	1
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	1
$t5	13	1
$t6	14	0

A four bit `and`

```
# Folder: L2\and.asm
# Ryan Preidel, 2011S

    li $t4, 0
    li $t5, 1
    and $t0, $t4, $t4
    li $v0, 1
    move $a0, $t0
    syscall

    and $t1, $t4, $t5
    li $v0, 1
    move $a0, $t1
    syscall

    and $t2, $t5, $t4
    li $v0, 1
    move $a0, $t2
    syscall

    and $t3, $t5, $t5
    li $v0, 1
    move $a0, $t3
    syscall

    li  $v0, 10
    syscall
```

A four bit **and**

```
# Folder: L2\and.asm
# Ryan Preidel, 2011S

    li $t4, 0                # Load two value 0 and 1 to be compared
    li $t5, 1
    and $t0, $t4, $t4        # comparing values of 0 and 0
    li $v0, 1
    move $a0, $t0            # moves value to print output
    syscall

    and $t1, $t4, $t5        # comparing values of 0 and 1
    li $v0, 1
    move $a0, $t1            # moves value to print output
    syscall

    and $t2, $t5, $t4        # comparing values of 1 and 0
    li $v0, 1
    move $a0, $t2            # moves value to print output
    syscall

    and $t3, $t5, $t5        # comparing values of 1 and 1
    li $v0, 1
    move $a0, $t3            # moves value to print output
    syscall

    li  $v0, 10              # system call code for exit = 10
    syscall                  # call operating sys o exit
```

Assemble …GO

```
0001
-- program is finished running --
```

Registers	Coproc 1	Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	10
$v1	3	0
$a0	4	1
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	1
$t4	12	0
$t5	13	1
$t6	14	0

or

```
# Folder: L2\or.asm
# Ryan Preidel, 2011S


    li $t4, 0
    li $t5, 1
    or $t0, $t4, $t4
    li $v0, 1
    move $a0, $t0
    syscall

    or $t1, $t4, $t5
    li $v0, 1
    move $a0, $t1
    syscall

    or $t2, $t5, $t4
    li $v0, 1
    move $a0, $t2
    syscall

    or $t3, $t5, $t5
    li $v0, 1
    move $a0, $t3
    syscall

    li  $v0, 10
    syscall
```

or

```
# Folder: L2\or.asm
# Ryan Preidel, 2011S


    li $t4, 0           # Load two value 0 and 1 to be compared
    li $t5, 1
    or $t0, $t4, $t4    # comparing values of 0 and 0
    li $v0, 1
    move $a0, $t0       # moves value to print output
    syscall

    or $t1, $t4, $t5    # comparing values of 0 and 1
    li $v0, 1
    move $a0, $t1       # moves value to print output
    syscall

    or $t2, $t5, $t4    # comparing values of 1 and 0
    li $v0, 1
    move $a0, $t2       # moves value to print output
    syscall

    or $t3, $t5, $t5    # comparing values of 1 and 1
    li $v0, 1
    move $a0, $t3       # moves value to print output
    syscall

    li  $v0, 10         # system call code for exit = 10
    syscall             # call operating sys
```

Assemble … GO

```
0111
-- program is finished running --
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	10
$v1	3	0
$a0	4	1
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	1
$t2	10	1
$t3	11	1
$t4	12	0
$t5	13	1
$t6	14	0

Negation (not)

- MIPS does not support bitwise negation (this can be achieved with the following two instructions):

  - **`xor`**
  - **`xori`**

5 Minutes …

xor

```
# Folder: L2\XOR.asm
# Ryan Preidel, 2011S

    li $t4, 0
    li $t5, 1
    xor $t0, $t4, $t4
    li $v0, 1
    move $a0, $t0
    syscall
    xor $t1, $t4, $t5
    li $v0, 1
    move $a0, $t1
    syscall
    xor $t2, $t5, $t4
    li $v0, 1
    move $a0, $t2
    syscall
    xor $t3, $t5, $t5
    li $v0, 1
    move $a0, $t3
    syscall
    li  $v0, 10
    syscall
```

xor

```
# Folder: L2\XOR.asm
# Ryan Preidel, 2011S

    li $t4, 0               # Load two value 0 and 1 to be compared
    li $t5, 1
    xor $t0, $t4, $t4       # comparing values of 0 and 0
    li $v0, 1
    move $a0, $t0           # moves value to print output
    syscall
    xor $t1, $t4, $t5       # comparing values of 0 and 1
    li $v0, 1
    move $a0, $t1           # moves value to print output
    syscall
    xor $t2, $t5, $t4       # comparing values of 1 and 0
    li $v0, 1
    move $a0, $t2           # moves value to print output
    syscall
    xor $t3, $t5, $t5       # comparing values of 1 and 1
    li $v0, 1
    move $a0, $t3           # moves value to print output
    syscall
    li  $v0, 10             # system call code for exit = 10
    syscall                 # call operating sys
```

Assemble GO

```
0110
-- program is finished running --
```

Registers	Coproc 1	Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	10
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	1
$t2	10	1
$t3	11	0
$t4	12	0
$t5	13	1
$t6	14	0

Shifting

Shifting

4.4.1 Arithmetic Instructions

	Op	Operands	Description
◦	`abs`	*des*, *src1*	*des* gets the absolute value of *src1*.
	`add(u)`	*des*, *src1*, *src2*	*des* gets *src1* + *src2*.
	`and`	*des*, *src1*, *src2*	*des* gets the bitwise and of *src1* and *src2*.
	`div(u)`	*src1*, *reg2*	Divide *src1* by *reg2*, leaving the quotient in register `lo` and the remainder in register `hi`.
◦	`div(u)`	*des*, *src1*, *src2*	*des* gets *src1* / *src2*.
◦	`mul`	*des*, *src1*, *src2*	*des* gets *src1* × *src2*.
◦	`mulo`	*des*, *src1*, *src2*	*des* gets *src1* × *src2*, with overflow.
	`mult(u)`	*src1*, *reg2*	Multiply *src1* and *reg2*, leaving the low-order word in register `lo` and the high-order word in register `hi`.
◦	`neg(u)`	*des*, *src1*	*des* gets the negative of *src1*.
	`nor`	*des*, *src1*, *src2*	*des* gets the bitwise logical **nor** of *src1* and *src2*.
◦	`not`	*des*, *src1*	*des* gets the bitwise logical negation of *src1*.
	`or`	*des*, *src1*, *src2*	*des* gets the bitwise logical **or** of *src1* and *src2*.
◦	`rem(u)`	*des*, *src1*, *src2*	*des* gets the remainder of dividing *src1* by *src2*.
◦	`rol`	*des*, *src1*, *src2*	*des* gets the result of rotating left the contents of *src1* by *src2* bits.
◦	`ror`	*des*, *src1*, *src2*	*des* gets the result of rotating right the contents of *src1* by *src2* bits.
	`sll`	*des*, *src1*, *src2*	*des* gets *src1* shifted left by *src2* bits.
	`sra`	*des*, *src1*, *src2*	Right shift arithmetic.
	`srl`	*des*, *src1*, *src2*	Right shift logical.
	`sub(u)`	*des*, *src1*, *src2*	*des* gets *src1* - *src2*.
	`xor`	*des*, *src1*, *src2*	*des* gets the bitwise exclusive **or** of *src1* and *src2*.

Shift left-logical (`sll`)

```
# Folder: L2\ShiftLeft.asm
# Rayan Preidel, 2011S

    li $t1, 100
    sll $t0, $t1, 1
    li $v0, 1
    move $a0, $t0          # moves value to print output
    syscall

    li  $v0, 10            # system call code for exit = 10
    syscall                # call operating sys to exit

```

$t0 = $a0 = ?

sll

```
200
-- program is finished running --
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	10
$v1	3	0
$a0	4	200
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	200
$t1	9	100
$t2	10	0

sll

- Shifts value left for said amount and adds zero as shifted
- Starting value is: 100_{10} or 01100100_2
- End result is: 200_{10} or 11001000_2
- 01100100_2
- 11001000_2

Multiply by 2

Shift right-logical (**`srl`**)

```
# Folder: L2\ShiftRight.asm
#  Rayan Preidel, 2011S

    li $t1, 100
    srl $t0, $t1, 1
    li $v0, 1
    move $a0, $t0       # moves value to print output
    syscall

    li  $v0, 10         # system call code for exit = 10
    syscall             # call operating sys

```

`$t0 = $a0 = ?`

Shift right-logical

```
50
-- program is finished running --
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	10
$v1	3	0
$a0	4	50
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	50
$t1	9	100
$t2	10	0

Shift right-logical

- Shifts value Right for said amount and adds zero as shifted
- Starting value is: 100_{10} or 01100100_2
- End result is: 50_{10} or 00110010_2
- 01100100_2
- 00110010_2

Divide by 2