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	
0	abs	des, src1	des gets the absolute value of src1.	
	add(u)	des, src1, src2	$des ext{ gets } src1 + src2.$	
	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.	
0	div(u)	des, src1, src2	des gets src1 / src2.	
0	mul	des, src1, src2	des gets $src1 \times src2$.	
0	mulo	des, src1, src2	des gets $src1 \times src2$, 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.	
0	neg(u)	des, src1	des gets the negative of src1.	
	nor	des, src1, src2	des gets the bitwise logical nor of src1 and src2.	
0	not	des, src1	des gets the bitwise logical negation of src1.	
	or	des, src1, src2	des gets the bitwise logical or of $src1$ and $src2$.	
0	rem(u)	des, src1, src2	des gets the remainder of dividing src1 by src2.	
0	rol	des, src1, src2	des gets the result of rotating left the contents of	
			src1 by src2 bits.	
0	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

loading an immediate 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

li pseudo-instruction

- li \$t0, c → ori \$t0, \$zero, c
- Note that $(0 \le c \le 65,535)$

• ori = or immediate

la (load address) pseudo-instruction

- la \$t0, message
- Is similar to 1i pseudo-instruction.

move pseudo-instruction

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



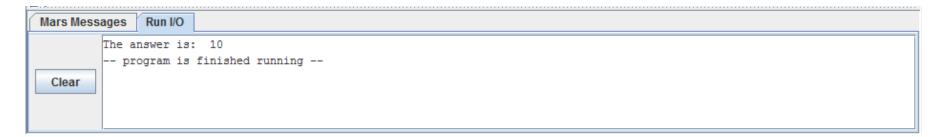
Add .. Subtract

```
What is the function that
 5
            .text
    main:
                                                       is implemented?
                             $t0, 9
            li
            1i
                             $t1, 6
 8
 9
            1i
                             $t2, 7
            sub
                             $t3, $t0, $t1
10
            add
                             $t4, $t3, $t2
11
            la
                             $aO, message
12
                                              # prints message
            1i
                             $v0, 4
13
            syscall
14
                             $a0, $t4
                                               # move to a0 for printout
15
            move
            1i
                             $v0, 1
16
            syscall
17
            li
                             $v0, 10
                                               # exit
18
            syscall
19
20
             .data
21
22
23
    message:
             .asciiz
                             "The answer is: "
24
```

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
 3
 4
 5
            .text
   main:
 6
                            $t0, 9
 7
            li
            li
                            $t1, 6
 8
            li
                            $t2, 7
 9
                            $t3, $t0, $t1
10
            sub
            add
                            $t4, $t3, $t2
11
            la
                            $aO, message
                                            # prints message
12
            1i
                            $v0, 4
13
            syscall
14
                            $a0, $t4
                                             # move to a0 for printout
15
            move
                            $v0, 1
            li
16
            syscal1
17
                                          # exit
            li
                            $v0, 10
18
            syscall
19
20
            .data
21
22
23
    message:
                            "The answer is: "
24
            .asciiz
```

Assemble ... GO



Į	Registers C	oproc 1 Copro	c 0
	Name	Number	Value
	\$zero	0	0
	\$at	1	268500992
	\$v0	2	10
	\$v1	3	0
\rightarrow	\$a0	4	10
	\$a1	5	0
	\$a2	6	0
1	\$a3	7	0
\rightarrow	\$t0	8	9
\rightarrow	\$t1	9	6
\rightarrow	\$t2	10	7
1	\$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

```
$t1, $t2 # Multiply ($t1*$t2) to produce a 64-bit number (Hi,Lo)
Mult
                       # move result to special32-bit register Hi ($t3)
Mfhi
        $t3
                       # move result to special 32-bit register Lo ($t4)
Mflo $t4
         $t1, $t2 # Div ($t1/$t2)
Div
                       # to produce a 32 -bit Lo [$t1 / $t2 integer quotient]
                       # to produce a 32 -bit Hi [$t1 % $t2 remainder]
                        # move result to special32-bit register Hi ($t3)
Mfhi
         $t3
                        # move result to special 32-bit register Lo ($t4)
Mflo
         $t4
```

```
mfhi ("move from hi")
mflo ("move from lo")
```

Multiply

```
4
             .text
 5
    main:
             li
                              $t1, 50
             li
                              $t2, 2
 8
             mult
                              $t1, $t2
 9
             mflo
                              $t3
10
             la
                              $aO, message
11
12
             li
                              $v0, 4
             syscall
13
14
                              $a0, $t3
15
             move
                              $v0, 1
16
             li
             syscall
17
18
             li
19
                              $v0, 10
             syscall
20
21
22
             .data
23
24
    message:
25
             asciiz
                              "The answer is:
```

What is the function that is implemented?

Multiply

```
Multiply.asm*
 Multiply2.asm
   ## Folder: L1\Multiply.asm
    ## L. Ruela, 2011
    ## (t1*t2) ... (2*50)
 3
 4
 5
            .text
 6
    main:
                            $t1, 50
            li
 7
                            $t2, 2
            li
 8
            mult
                            $t1, $t2
 9
                            $t3
            mflo
10
                            $aO, message
                                                # prints message
            la
11
            li
                            $v0, 4
12
            syscall
13
14
                            $a0, $t3
                                                  # move to a0 for printout.
15
            move
16
            li
                             $v0, 1
            syscall
17
18
                            $v0, 10
                                                  # exit
19
            li
            syscall
20
21
22
            .data
23
24
    message:
25
                             "The answer is: "
            .asciiz
```

Assemble ... GO

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

	Registers C	oproc 1 Copro	c 0
	Name	Number	Value
	\$zero	0	0
	\$at	1	268500992
	\$v0	2	10
	\$v1	3	0
	\$a0	4	100
	\$a1	5	0
000000000000000000000000000000000000000	\$a2	6	0
	\$a3	7	0
	\$t0	8	0
\rightarrow	\$t1	9	50
\rightarrow	\$t2	10	2
	\$t3	11	100

Multiply in MIPS; Hi-Lo registers

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

```
mfhi ("move from hi")
mflo ("move from lo")
```

Divide

```
What is the function that
            .text
  main:
                                                      is implemented?
            li 
                           $t0, 24
            li
                           $t1, 8
                           $t0, $t1
            div
            mflo.
                           $ t.3
1.0
            mfhi
                           $t4
11
                           $a0, message # prints message
12
            1a
            1i
                           $v0, 4
13
14
            syscall
15
16
                           $a0, $t3
                                         # move to a0 for printout
            move
            li.
                           $v0, 1
17
            syscall
18
19
            1i
                           $v0,10
                                            # exit
20
21
            syscall
22
            .data
23
24
25 message:
            .asciiz
                                    "The answer is:
26
27
```

Divide

```
1 ## Folder: L1\Divide.asm
 2 ## Lurdes Ruela, 2011S
 3 ## s0/s1 ... (24/8)
            .text
 6 main:
            li.
                          $t0, 24
            li.
                          $t1, 8
            div
                          $t0, $t1
           mflo
                          $t3
10
           mfhi
                          $t4
11
                          $a0, message # prints message
12
           1a
13
            li
                          $v0, 4
14
            syscall
15
16
                          $a0, $t3 # move to a0 for printout
            move
17
           li
                          $v0, 1
            syscall
18
19
                                       # exit
            li
                          $v0,10
2.0
            syscall
21
22
23
            .data
24
25 message:
                                  "The answer is: "
26
            .asciiz
27
```

Divide

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

Registers C	oproc 1 Copro	c 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

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

Hi (Remainder)

Lo (Quotient)

```
mfhi ("move from hi")
mflo ("move from lo")
```

Another Divide Example

```
.text
 6
    main:
              li
                                $t0, 26
 8
                                $t1, 8
               li
              div
                                $t0, $t1
 9
              mflo
                                $t3
10
11
              mfhi
                                $t4
12
              li
                                $v0,10
13
              syscall
14
```

Another Divide Example

```
Divide2.asm
    ## Folder: L1\Divide2.asm
    ## L. Ruela, 2011S
3
    ## t0/t1 ... (26/8)
 4
 5
             .text
    main:
                            $t0, 26
             li
                                                  mflo = ?
             li
                            $t1, 8
             div
                                                  Mfhi
                            $t0, $t1
             mflo
                            $t3
10
                                                   $t3
11
             mfhi
                            $t4
12
                                                   $t4
             li
                            $v0,10
13
             syscall
14
```

Assemble GO

zero	0	0
at	1	0
7V0	2	10
vl	3	-0
a0	4	0
al	5	-0
a2	6	0
a3	7	0
t0	8	26
tl	9	- 8-
t2	10	0
t3	11	113
t4	12	2
t5	13	- 0
t6	14	0
t7	15	-0
s0	16	0
sl	17	.0
s2	18	0
s 3	19	: 0
34	20	0
s5	21	-0
36	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
С		4194332
i		2
0		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

Hi (Remainder)

Lo (Quotient)

```
mfhi ("move from hi")
mflo ("move from lo")
```

Logic operations

Logical

4.4.1 Arithmetic Instructions

		Op	Operands	Description
	0	abs	des, src1	des gets the absolute value of src1.
		add(u)	des, src1, src2	$des ext{ gets } src1 + src2.$
\Rightarrow		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.
	0	div(u)	des, src1, src2	des gets src1 / src2.
	0	mul	des, src1, src2	des gets $src1 \times src2$.
	0	mulo	des, src1, src2	des gets $src1 \times src2$, 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.
	0	neg(u)	des, src1	des gets the negative of src1.
•		nor	des, src1, src2	des gets the bitwise logical nor of src1 and src2.
	0	not	des, src1	des gets the bitwise logical negation of src1.
		or	des, src1, src2	des gets the bitwise logical or of $src1$ and $src2$.
	0	rem(u)	des, src1, src2	des gets the remainder of dividing src1 by src2.
	0	rol	des, src1, src2	des gets the result of rotating left the contents of
				src1 by src2 bits.
	0	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
 3
       li $t4, 1
                            # Load two value 1 and 1 to be compared
 4
       li $t5, 1
 5
                            # comparing values of 0 and 0
 6
        and $t0, $t5, $t4
 7
        li $v0, 1
                             # moves value to print output
 8 9
        move $a0, $t0
        syscall
10
11
                             # system call code for exit = 10
        li $v0, 10
12
                            # call operating sys o exit
        syscall
13
```

and

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

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

A four bit and

```
1 # Folder: L2\and.asm
 2 # Ryan Preidel, 2011S
 4
       li $t4, 0
       li $t5, 1
 5
       and $t0, $t4, $t4
 6
 7
       li $v0, 1
       move $a0, $t0
 8
 9
       syscall
10
11
       and $t1, $t4, $t5
12
       li $v0, 1
13
       move $a0, $t1
14
       syscall
15
       and $t2, $t5, $t4
16
17
       li $v0, 1
18
       move $a0, $t2
19
       syscall
20
       and $t3, $t5, $t5
21
22
       li $v0, 1
23
       move $a0, $t3
24
       syscall
25
       li $v0, 10
26
27
       syscall
```

A four bit and

```
1 # Folder: L2\and.asm
2 # Ryan Preidel, 2011S
4
      li $t4, 0
                         # Load two value 0 and 1 to be compared
      li $t5, 1
5
    and $t0, $t4, $t4 # comparing values of 0 and 0
6
7
      li $v0, 1
8
      move $a0, $t0 # moves value to print output
9
      syscall
10
11 and $t1, $t4, $t5 # comparing values of 0 and 1
12
      li $v0, 1
13
      move $a0, $tl
                     # moves value to print output
      syscall
14
15
16 and $t2, $t5, $t4 # comparing values of 1 and 0
      li $v0, 1
17
                     # moves value to print output
18
      move $a0, $t2
19
      syscall
20
   ⇒ and $t3, $t5, $t5 # comparing values of 1 and 1
21
22
      li $v0, 1
                     # moves value to print output
23
      move $a0, $t3
24
      syscall
25
                       # system call code for exit = 10
      li $v0, 10
26
                           # call operating sys o exit
27
      syscall
```

Assemble ...GO

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

	Registers	C	oproc 1	Copro	c 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	
\Rightarrow	\$t0			8	0	
\Rightarrow	\$t1			9	0	
\Rightarrow	\$t2			10	0	
	\$t3			11	1	
,	\$t4			12	0	
	\$t5			13	1	
	\$t6			14	0	

or

```
1 # Folder: L2\or.asm
 2 # Ryan Preidel, 2011S
 3
4
5
       li $t4, 0
 6
       li $t5, 1
 7
       or $t0, $t4, $t4
 8
       li $v0, 1
 9
       move $a0, $t0
10
       syscall
11
       or $t1, $t4, $t5
12
       li $v0, 1
13
       move $a0, $t1
14
15
       syscall
16
17
       or $t2, $t5, $t4
       li $v0, 1
18
       move $a0, $t2
19
       syscall
20
21
22
       or $t3, $t5, $t5
       li $v0, 1
23
24
       move $a0, $t3
       syscall
25
26
       li $v0, 10
27
       syscall
28
```

or

```
1 # Folder: L2\or.asm
2 # Ryan Preidel, 2011S
3
 4
5
      li $t4, 0
                         # Load two value 0 and 1 to be compared
      li $t5, 1
6
      or $t0, $t4, $t4
                         # comparing values of 0 and 0
8
      li $v0, 1
                         # moves value to print output
      move $a0, $t0
9
      syscall
10
11
                         # comparing values of 0 and 1
12 or $t1, $t4, $t5
      li $v0, 1
13
      move $a0, $tl
                         # moves value to print output
14
      syscall
15
16
17
      or $t2, $t5, $t4
                         # comparing values of 1 and 0
18
      li $v0, 1
      move $a0, $t2
                         # moves value to print output
19
      syscall
20
21
      or $t3, $t5, $t5
                         # comparing values of 1 and 1
22
23
      li $v0, 1
      move $aO, $t3
                         # moves value to print output
24
25
      syscall
26
                  # system call code for exit = 10
      li $v0, 10
27
      syscall
                         # call operating sys
28
```

Assemble ... GO

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

Registers C	oproc 1 Copro	c 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
	Name \$zero \$at \$v0 \$v1 \$a0 \$a1 \$a2 \$a3 \$t0 \$t1 \$t2 \$t3 \$t4 \$t5	Name Number \$zero 0 \$at 1 \$v0 2 \$v1 3 \$a0 4 \$a1 5 \$a2 6 \$a3 7 \$t0 8 \$t1 9 \$t2 10 \$t3 11 \$t4 12 \$t5 13

Negation (not)

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

- xor
- xori

5 Minutes ...

xor

```
1 # Folder: L2\XOR.asm
   2 # Ryan Preidel, 2011S
   3
    4
          li $t4, 0
    5
          li $t5, 1
          xor $t0, $t4, $t4
          li $v0, 1
    8
          move $a0, $t0
    9
          syscall
\Rightarrow10
          xor $t1, $t4, $t5
          li $v0, 1
  11
          move $a0, $t1
  12
  13
          syscall
          xor $t2, $t5, $t4
\Rightarrow14
          li $v0, 1
  15
          move $a0, $t2
  16
          syscall
  17
⇒18
          xor $t3, $t5, $t5
          li $v0, 1
  19
  20
          move $a0, $t3
  21
          syscall
  22
          li $v0, 10
  23
          syscall
```

xor

```
1 # Folder: L2\XOR.asm
   2 # Ryan Preidel, 2011S
   3
   4
         li $t4, 0
                              # Load two value 0 and 1 to be compared
   5
         li $t5, 1
         xor $t0, $t4, $t4 # comparing values of 0 and 0
         li $v0, 1
         move $a0, $t0
                             # moves value to print output
   9
         syscall
         xor $t1, $t4, $t5 # comparing values of 0 and 1
\Rightarrow10
         li $v0, 1
  11
                            # moves value to print output
         move $a0, $tl
  12
  13
         syscall
         xor $t2, $t5, $t4 # comparing values of 1 and 0
\Longrightarrow14
         li $v0, 1
  15
  16
         move $a0, $t2
                              # moves value to print output
         syscall
  17
         xor $t3, $t5, $t5 # comparing values of 1 and 1
\Rightarrow18
         li $v0, 1
  19
         move $a0, $t3
                             # moves value to print output
  20
         syscall
  21
         li $v0, 10
                           # system call code for exit = 10
  22
         syscall
                              # call operating sys
  23
```

Assemble GO

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

Registers	Copro	c 1	Coproc	0
Name		Num	ber	Value
\$zero			0	(
\$at			1	(
\$v0			2	10
\$v1			3	(
\$a0			4	(
\$a1			5	(
\$a2			6	(
\$a3			7	(
\$t0			8	(
\$t1			9	:
\$t2			10	
\$t3			11	(
\$t4			12	(
\$t5			13	-
\$t6			14	(

Shifting

Shifting

4.4.1 Arithmetic Instructions

Γ		Op	Operands	Description
	0	abs	des, src1	des gets the absolute value of src1.
Г		add(u)	des, src1, src2	$des ext{ gets } src1 + src2.$
Γ		and	des, src1, src2	des gets the bitwise and of src1 and src2.
Γ		div(u)	src1, reg2	Divide src1 by reg2, leaving the quotient in register
ı				lo and the remainder in register hi.
	0	div(u)	des, src1, src2	des gets $src1 / src2$.
Г	0	mul	des, src1, src2	des gets $src1 \times src2$.
	0	mulo	des, src1, src2	des gets $src1 \times src2$, with overflow.
ı		mult(u)	src1, reg2	Multiply src1 and reg2, leaving the low-order word
ı				in register lo and the high-order word in register
				hi.
	0	neg(u)	des, src1	des gets the negative of src1.
Γ		nor	des, src1, src2	des gets the bitwise logical nor of src1 and src2.
	0	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.
	0	rem(u)	des, src1, src2	des gets the remainder of dividing src1 by src2.
Г	0	rol	des, src1, src2	des gets the result of rotating left the contents of
ı				src1 by src2 bits.
	0	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.
Ł		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 (s11)

```
1 # Folder: L2\ShiftLeft.asm
2 # Rayan Preidel, 2011S
 3
      li $t1, 100
       sl1 $t0, $t1, 1
 5
      li $v0, 1
 6
      move $a0, $t0 # moves value to print output
 8
       syscall
 9
                      # system call code for exit = 10
      li $v0, 10
10
                          # call operating sys to exit
11
       syscall
12
```

$$$t0 = $a0 = ?$$

sll

200
-- program is finished running --

Registers C	оргос 1 Сорг	oc 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	1	0 0

sll

- Shifts value left for said amount and adds zero as shifted
- Starting value is: 100₁₀ or 01100100₂
- End result is: 200₁₀ or 11001000₂
- 011001002
- 110010002



Multiply by 2

Shift right-logical (srl)

$$$t0 = $a0 = ?$$

Shift right-logical

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

	Registers C	oproc 1 Copro	oc 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: 100₁₀ or 01100100₂
- End result is: 50₁₀ or 00110010₂
- 011001002
- 001100102

Divide by 2