[LOGIC, SHIFTING]

MIPS Assembly Programming

Logic and Shifting Operations

MIPS Instruction Set

- Arithmetic
  - Algebraic (`add`, `sub`, `mult`, `div`) instructions
  - Logic (`and`, `nor`, `or`, `xor`) instructions
  - Shifting (`sll`, `sra`, `srl`) instructions
- Branch Instructions
- Function Call Instructions
- Load and Store Instructions.

Logic 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*.

and
nor
or
xor

Digital computers work using binary, with data represented as 1’s and 0’s …therefore…

`and` operation/instruction

Example-1

and

```
        .text
        .globl main
main:
        li      $t4, 1
        li      $t5, 1
        and     $t0, $t5, $t4

        move    $a0, $t0
        li      $v0, 1
        syscall

        li      $v0, 10
        syscall
```

$t0 = ?

Result

```
        .text
        .globl main
main:
        li      $t4, 1
        li      $t5, 1
        and     $t0, $t5, $t4

        move    $a0, $t0
        li      $v0, 1
        syscall

        li      $v0, 10
        syscall
```

$t0 = 1

Assemble … GO

```
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

and

```
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

move    $a0, $t0        # moves value to print output

li      $v0, 1
syscall

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

A four bit `and`

Example-2

A four bit **and**

```
        .text
        .globl main
main:

        li      $t4, 0
        li      $t5, 1
        and     $t0, $t4, $t4
        move    $a0, $t0
        li      $v0, 1
        syscall
        and     $t1, $t4, $t5
        move    $a0, $t1
        li      $v0, 1
        syscall
        and     $t2, $t5, $t4
        move    $a0, $t2
        li      $v0, 1
        syscall
        and     $t3, $t5, $t5
        move    $a0, $t3
        li      $v0, 1
        syscall

       li      $v0, 10
       syscall
```

Trace the code

$t0 = ?

$t1 = ?

$t2 = ?

$t3 = ?

Assemble … GO

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

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

$t0 = 0
$t1 = 0
$t2 = 0
$t3 = 1

A four bit **and**

```
        .text
        .globl main
main:

        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
        move    $a0, $t0            # moves value to print output
        li      $v0, 1
        syscall
        and     $t1, $t4, $t5       # comparing values of 0 and 1
        move    $a0, $t1            # moves value to print output
        li      $v0, 1
        syscall
        and     $t2, $t5, $t4       # comparing values of 1 and 0
        move    $a0, $t2            # moves value to print output
        li      $v0, 1
        syscall
        and     $t3, $t5, $t5       # comparing values of 1 and 1
        move    $a0, $t3            # moves value to print output
        li      $v0, 1
        syscall

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

`or` instruction, 4-bits

Example-3

or

```
        .text
        .globl main
main:

        li $t4, 0
        li $t5, 1
        or $t0, $t4, $t4
        or $t1, $t4, $t5
        or $t2, $t5, $t4
        or $t3, $t5, $t5

        li $v0, 10
        syscall
```

Trace the code

$t0 = ?

$t1 = ?

$t2 = ?

$t3 = ?

or

```
        .text
        .globl main
main:

        li $t4, 0
        li $t5, 1
        or $t0, $t4, $t4
        or $t1, $t4, $t5
        or $t2, $t5, $t4
        or $t3, $t5, $t5

        li $v0, 10
        syscall
```

Trace the code

$t0 = 0
$t1 = 1
$t2 = 1
$t3 = 1

Assemble … GO

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

$t0 = 0

$t1 = 1

$t2 = 1

$t3 = 1

or

```
        .text
        .globl main
main:

        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
        or $t1, $t4, $t5      # comparing values of 0 and 1
        or $t2, $t5, $t4      # comparing values of 1 and 0
        or $t3, $t5, $t5      # comparing values of 1 and 1

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

`ori` instruction

(`ori` = `or-i`mmediate)

Example-4a

`ori` example

```
ori $t0, $0, 0x12
```

`0x12` << Hexadecimal `12`

`$t0 = ?`

2 Minutes …

`ori` example

`ori $t0, $0, 0x12`

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	18
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$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		4194308
hi		0
lo		0

(Registers | Coproc 1 | Coproc 0)

$t0 = 18 (decimal)

$t0 = 0x12 (Hex)

`ori … ori`

Example-4b

ori … ori

```
        .text
        .globl main
main:
        ori $t0,$0,0xA
        ori $t1,$0,0xB
        add $t2,$t1,$t0

        li $v0, 10
        syscall
```

Trace the code

$t0 = ?
$t1 = ?
$t2 = ?

ori … ori

```
        .text
        .globl main
main:
        ori $t0,$0,0xA
        ori $t1,$0,0xB
        add $t2,$t1,$t0

        li $v0, 10
        syscall
```

$t0 = 10
$t1 = 11
$t2 = 21

`not` operation

`xor` operation/instruction

Negation (binary NOT operation)

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

  - `xor des, src1, src2`

scr1	src2	des
0	0	0
0	1	1
1	0	1
1	1	0

What should be modified in the instruction in order to behave as NOT operation?

2 Minutes …

`des` (destination), `src1` (`source1`), `src2` (`source2`)

Negation (Solution)

- Just set: **`src1`** or **`src2`** to 1:
  - **`xor des, 1, src2`**

scr1	src2	des
0	0	0
0	1	1
1	0	1
1	1	0

  - **`xor des, src1, 1`**

`xor` instruction, 4-bits

Example-5

xor

```
        .text
        .globl main
main:
        li   $t4, 0
        li   $t5, 1
        xor $t0, $t4, $t4
        xor $t1, $t4, $t5
        xor $t2, $t5, $t4
        xor $t3, $t5, $t5

        li   $v0, 10
        syscall
```

$t0 = ?

$t1 = ?

$t2 = ?

$t3 = ?

Result

```
        .text
        .globl main
main:
        li   $t4, 0
        li   $t5, 1
        xor  $t0, $t4, $t4
        xor  $t1, $t4, $t5
        xor  $t2, $t5, $t4
        xor  $t3, $t5, $t5

        li   $v0, 10
        syscall
```

$t0 = 0
$t1 = 1
$t2 = 1
$t3 = 0

Assemble … GO

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

Registers | Coproc 1 | Coproc 0

$t0 = 0

$t1 = 1

$t2 = 1

$t3 = 0

xor

```
        .text
        .globl main
main:
        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
        xor $t1, $t4, $t5       # comparing values of 0 and 1
        xor $t2, $t5, $t4       # comparing values of 1 and 0
        xor $t3, $t5, $t5       # comparing values of 1 and 1

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

Shifting Operations

MIPS Instruction Set

- Arithmetic
  - Algebraic (**`add`**, **`sub`**, **`mult`**, **`div`**) instructions
  - Logic (**`and`**, **`nor`**, **`or`**, **`xor`**) instructions
  - Shifting (**`sll`**, **`sra`**, **`srl`**) instructions
- Branch Instructions
- Function Call Instructions
- Load and Store Instructions.

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		`sll`	*des, src1, src2*	*des* gets *src1* shifted left by *src2* bits.
		`sra`	*des, src1, src2*	Right shift arithmetic.
srl		`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 **l**eft-**l**ogical (`sll`)

Example-6

Shift left-logical (sll)

```
        .text
        .globl main
main:
        li      $t1, 100
        sll     $t0, $t1, 1

        move    $a0, $t0
        li      $v0, 1
        syscall

        li       $v0, 10
        syscall
```

Trace the code

100 = Decimal number

sll by 1 bit

$t0 = ?

Assembly … GO

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

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

$t0 = 200

Digital computers work using binary, with data represented as 1’s and 0’s …therefore…

Our example with Shift left-logical (`sll`)

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

Shift Left by 1

Multiply by 2

shift **r**ight-**l**ogical (**`srl`**)

Example-7

Shift right-logical (srl)

```
        .text
        .globl main
main:
        li      $t1, 100
        srl     $t0, $t1, 1

        move    $a0, $t0
        li      $v0, 1
        syscall

        li      $v0, 10
        syscall
```

Trace the code

100 = Decimal number

srl by 1 bit

$t0 = ?

Assemble … GO

```
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

$t0 = 50

Shift right-logical

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

Shift Right by 1

Divide by 2

With `srl`...

Example-8

Two … srl

Trace the code

```
        .text
        .globl main
main:
        li      $t1, 100
        srl     $t0, $t1, 1
        srl     $t2, $t0, 1

        li      $v0, 10
        syscall
```

100 = decimal
srl by 1 bit
srl by 1 bit

$t2 = ?

Assemble … GO

```
        .text
        .globl main
main:
        li      $t1, 100
        srl     $t0, $t1, 1
        srl     $t2, $t0, 1

        li      $v0, 10
        syscall
```

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	50
$t1	9	100
$t2	10	25
$t3	11	0
$t4	12	0
$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		4194324
hi		0
lo		0

$t2 = 25

Therefore …

Binary Multiplication/Division

Multiplying by 2^n is the same as shifting left by **n** bits

Dividing by 2^n is the same as shifting right by **n** bits