MIPS Programming

Rathindra Nath Dutta & Subhojit Sarkar

Senior Research Fellow
Advanced Computing & Microelectronics Unit
Indian Statistical Institute, Kolkata

STATISTICAL INDIAN INSTITUTE
UNITY IN DIVERSITY

September 1, 2022

Writing Your First MIPS Code

```
// C code
a = 10
b = 20
c = a + b
```

⇒

```
# using instructions
li a, 10
li b, 20
add c, a, b
```

⇒

```
# final MIPS Code
li $t0, 10
li $t1, 20
add $t3, $t0, $t1
```

A MIPS Code Template

```
# Declare main as a global function
.globl main

# All program code is placed after the
# .text assembler directive
.text

# The label 'main' represents the starting point
main:
    # YOUR CODE GOES HERE

    # Exit the program by means of a syscall.
    # by placing its code in $v0. The code for exit is "10"
    li $v0, 10 # exit syscall
    syscall

# All memory structures are placed after the
# .data assembler directive
.data

# The .word assembler directive reserves space
# in memory for one or more 4-byte words
list:  .word 1, 4, 8
```

Running a MIPS Code

- Ideally one should execute on a MIPS hardware
- We will be using a free simulator tool: SPIM[1]

[1] more specifically QtSPIM: `http://spimsimulator.sourceforge.net/`

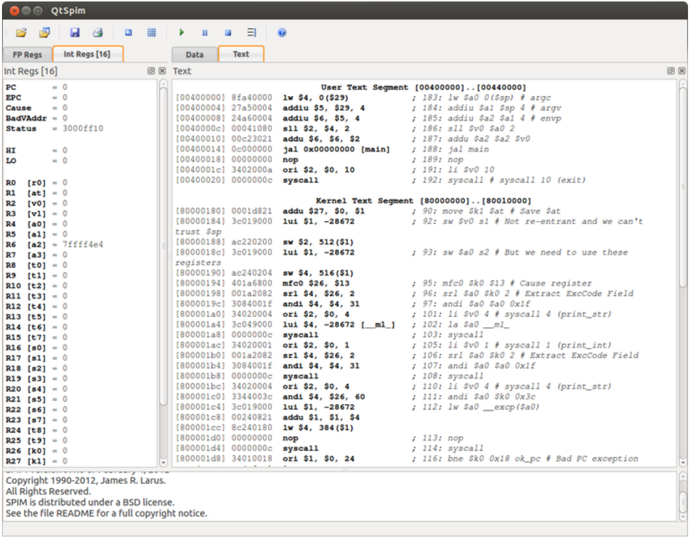
Running a MIPS Code

- Ideally one should execute on a MIPS hardware
- We will be using a free simulator tool: SPIM[1]
- Name of the simulator is a reversal of the letters 'MIPS'

[1] more specifically QtSPIM: `http://spimsimulator.sourceforge.net/`

Getting started with QtSPIM

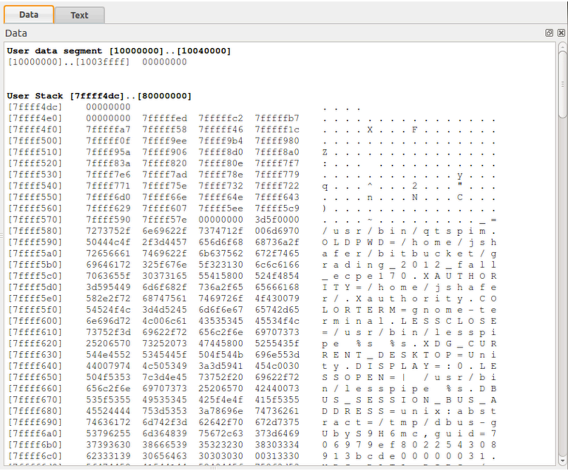
3 primary sections: Register panel, Memory panel, & Messages panel.



Getting started with QtSPIM contd.

Text tab in Memory panel shows the Program memory contents

Data tab shows the contents of the Data memory space



QtSPIM Demo

- Save your MIPS code with `.s` or `.asm` extension
- Load your code in QtSPIM via ‘Reinitialize and Load File’ option
- Click on the play button to run your code

Using QtSPIM Console

Consider the following C code fragment

```c
int x = 10;
int y = 20;
printf("%d", x + y); // prints a integer
```

Using QtSPIM Console contd.

```
.globl main
.text

main:
    lw $t0, x
    lw $t1, y
    add $t3, $t0, $t1

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

    li $v0, 10 # exit syscall
    syscall

.data
x:  .word   10
y:  .word   20
```

[1] SPIM syscalls: https://www.doc.ic.ac.uk/lab/secondyear/spim/node8.html

Array and Loops

Consider the following C code fragment

```c
int arr[] = {1, 5, 8, 10, 3};
int n = 5; // lenght of arr
int sum = 0;
int i = 0;
while (i != n) {
    sum = sum + arr[i];
    i = i + 1;
}
printf("%d", sum);
```

Array and Loops contd.

Terminating condition rewritten

Array indexing replaced by pointer operation

```
int arr[] = {1, 5, 8, 10, 3};
int n = 5; // lenght of arr
int sum = 0;
int i = 0;
while (n != 0) {
    sum = sum + *(arr + i); // pointer arithmetic
    i = i + 1;
    n = n - 1;
}
printf("%d", sum);
```

Array and Loops contd.

The `while` loop is converted to `do...while` assuming $n > 0$

```
int arr[] = {1, 5, 8, 10, 3};
int n = 5; // lenght of arr
int sum = 0;
int i = 0;
do {
    sum = sum + *(arr + i); // pointer arithmetic
    i = i + 1;
    n = n - 1;
} while (n != 0); // assume n > 0
printf("%d", sum);
```

Array and Loops contd.

Utility of the index variable `i` is substituted with pointer shifting

```
int arr[] = {1, 5, 8, 10, 3};
int n = 5; // lenght of arr
int sum = 0;
int *p = arr; // base address
do {
    sum = sum + *p;
    p = p + 1; // pointer arithmetic
    n = n - 1;
} while (n != 0); // assume n > 0
printf("%d", sum);
```

Array and Loops contd.

```
.globl main
.data
arr:  .word   1, 5, 8, 10, 3
n:    .word   5
.text
main:
    la $t0, arr # p
    lw $t1, n
    li $t2, 0 # sum
loop:
    lw $t4, 0($t0) # *p
    add $t2, $t2, $t4 # sum = sum + *p
    addi $t0, $t0, 4 # incrementing p, integers are 4 byte long
    addi $t1, $t1, -1 # n = n - 1
    bne $t1, $0, loop

    li $v0, 1 # print_int syscall
    move $a0, $t2 # copy sum
    syscall
```

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code
x = 1 << 20
```

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code
x = 1 << 20
```

```
# using MIPS
li $t0, 1 # load 1
sll $t0, $t0, 20
# shift left by 20 places
```

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code
x = 1 << 20
```

```
# using MIPS
li $t0, 1 # load 1
sll $t0, $t0, 20
# shift left by 20 places
```

- computing $n \times 2^{10}$

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code
x = 1 << 20
```

```
# using MIPS
li $t0, 1 # load 1
sll $t0, $t0, 20
# shift left by 20 places
```

- computing $n \times 2^{10}$

```
// C code
x = n << 10
```

```
# using MIPS
# assume $t0 contains n
sll $t1, $t0, 10
```

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code
x = 1 << 20
```

```
# using MIPS
li $t0, 1 # load 1
sll $t0, $t0, 20
# shift left by 20 places
```

- computing $n \times 2^{10}$

```
// C code
x = n << 10
```

```
# using MIPS
# assume $t0 contains n
sll $t1, $t0, 10
```

- computing $\lfloor n/2^4 \rfloor$

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code
x = 1 << 20
```

```
# using MIPS
li $t0, 1 # load 1
sll $t0, $t0, 20
# shift left by 20 places
```

- computing $n \times 2^{10}$

```
// C code
x = n << 10
```

```
# using MIPS
# assume $t0 contains n
sll $t1, $t0, 10
```

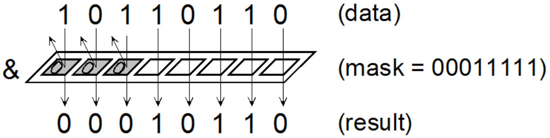
- computing $\lfloor n/2^4 \rfloor$

```
// C code
x = n >> 4
```

```
# using MIPS
# assume $t0 contains n
srl $t1, $t0, 4
```

Bitwise Operation and Masking

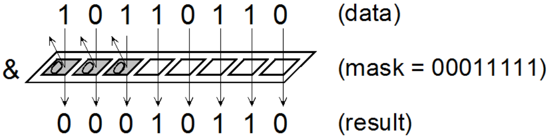
- Bitwise `and` operation



[1] image src: `https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html`

Bitwise Operation and Masking

- Bitwise `and` operation

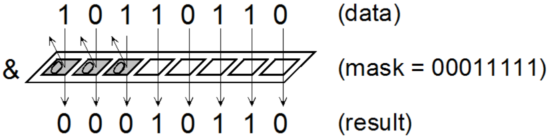


Getting `i`-th bit:

[1] image src: `https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html`

Bitwise Operation and Masking

- Bitwise `and` operation

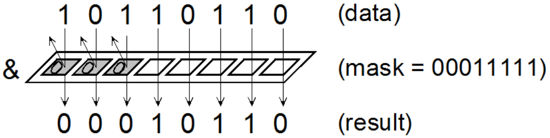


Getting `i`-th bit: `x & (1 << i)`

[1] image src: `https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html`

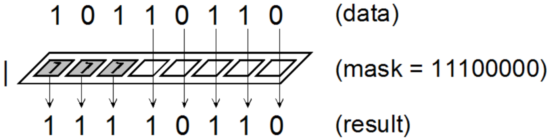
Bitwise Operation and Masking

- Bitwise `and` operation



Getting `i`-th bit:`x & (1 << i)`

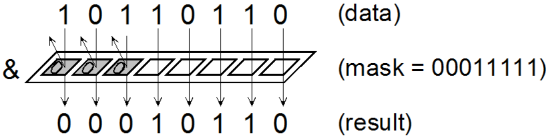
- Bitwise `or` operation



[1] image src: `https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html`

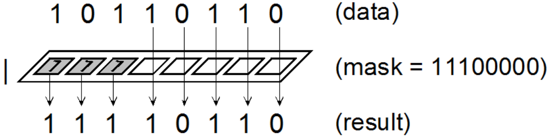
Bitwise Operation and Masking

- Bitwise `and` operation



Getting `i`-th bit: `x & (1 << i)`

- Bitwise `or` operation

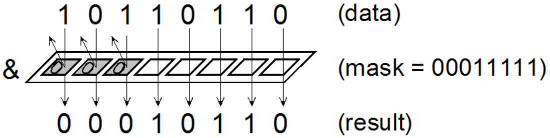


Setting `i`-th bit:

[1] image src: `https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html`

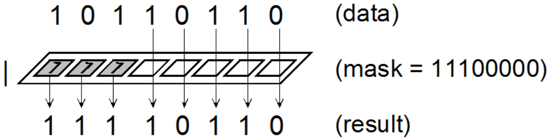
Bitwise Operation and Masking

- Bitwise `and` operation



Getting `i`-th bit:`x & (1 << i)`

- Bitwise `or` operation



Setting `i`-th bit:`x | (1 << i)`

- Ex. What happens with `n & (n - 1)`?

[1] image src: `https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html`

Bitwise Operation and Masking contd.

- The `xor` operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise Operation and Masking contd.

- The `xor` operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

Bitwise Operation and Masking contd.

- The `xor` operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise `xor` operation
  Flipping `i`-th bit:

Bitwise Operation and Masking contd.

- The `xor` operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise `xor` operation
  Flipping `i`-th bit:`x ^ (1 << i)`

Bitwise Operation and Masking contd.

- The `xor` operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise `xor` operation
  Flipping `i`-th bit:`x ^ (1 << i)`
- Ex. What is output of: `n ^ 0xAAAAAAAA`?

Bitwise Operation and Masking contd.

- The `xor` operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise `xor` operation
  Flipping `i`-th bit:`x ^ (1 << i)`
- Ex. What is output of: `n ^ 0xAAAAAAAA`?
- Ex. What is output of: `n ^ 0x55555555`?

Bitwise Operation and Masking contd.

- The `xor` operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise `xor` operation
  Flipping `i`-th bit:`x ^ (1 << i)`
- Ex. What is output of: `n ^ 0xAAAAAAAA`?
- Ex. What is output of: `n ^ 0x55555555`?
- Ex. What is output of: `n ^ 0xFFFFFFFF`?

Bitwise Operation and Masking contd.

- Getting NOTHING out of anything

[1] image src: `https://en.wikipedia.org/wiki/XOR_swap_algorithm`

Bitwise Operation and Masking contd.

- Getting NOTHING out of anything: $X \oplus X = 0$

[1] image src: `https://en.wikipedia.org/wiki/XOR_swap_algorithm`

Bitwise Operation and Masking contd.

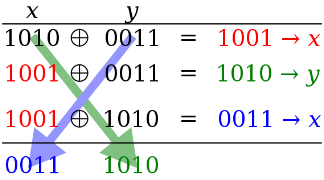
- Getting NOTHING out of anything: $X \oplus X = 0$
- Swapping values of two variables

[1] image src: `https://en.wikipedia.org/wiki/XOR_swap_algorithm`

Bitwise Operation and Masking contd.

- Getting NOTHING out of anything: $X \oplus X = 0$
- Swapping values of two variables

Operation	*Meaning*
$a = a \oplus b$	$a = A \oplus B$
$b = b \oplus a$	$b = B \oplus (A \oplus B) = A$
$a = a \oplus b$	$a = (A \oplus B) \oplus A = B$



[1] image src: `https://en.wikipedia.org/wiki/XOR_swap_algorithm`

Assignment 3 (informally)

Question 2

Devise an efficient way to obtain 1's complement of an integer. You are restricted from specifying any constant explicitly (cannot do $X \oplus -1$).

Assignment 3 (informally)

Question 2

Devise an efficient way to obtain 1's complement of an integer. You are restricted from specifying any constant explicitly (cannot do $X \oplus -1$).

Question 3

Load a constant value without specifying any constant explicitly.

Assignment 3 (informally)

Question 2

Devise an efficient way to obtain 1's complement of an integer. You are restricted from specifying any constant explicitly (cannot do $X \oplus -1$).

Question 3

Load a constant value without specifying any constant explicitly.

Question 4

Count the number of 1s in an integer.

Assignment 3 (informally)

Question 2

Devise an efficient way to obtain 1's complement of an integer. You are restricted from specifying any constant explicitly (cannot do $X \oplus -1$).

Question 3

Load a constant value without specifying any constant explicitly.

Question 4

Count the number of 1s in an integer.

Question 5

Suppose there are n distinct integers all in the closed interval of $[0, n]$, that is only one number is absent, and all others occur exactly once. Your task is to find the missing number efficiently.