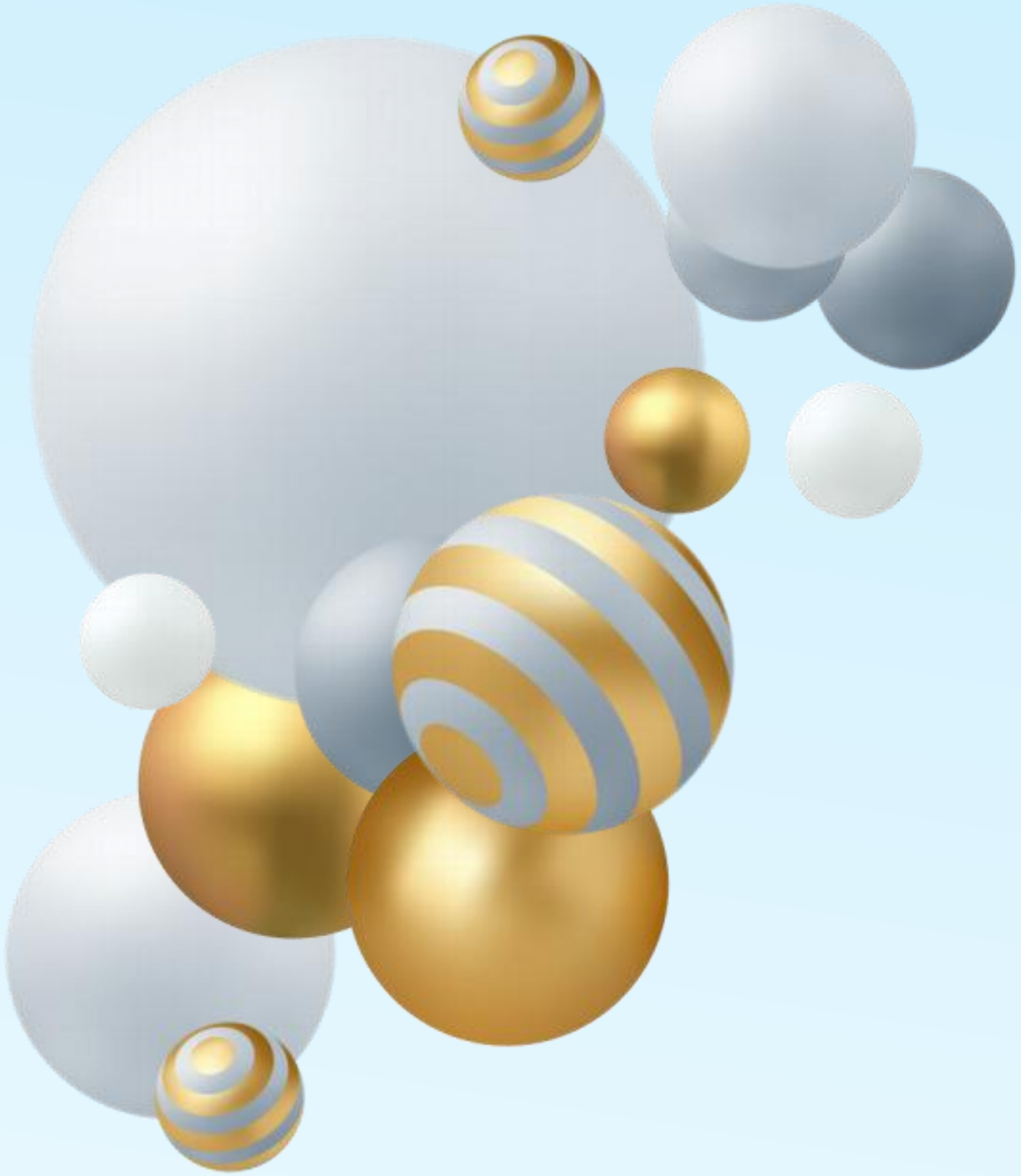
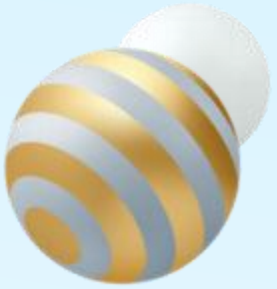


Computer Organization

Lab3 MIPS(2)

Details of Data



TOPIC

- 1. Data Processing Details
  - **Signed** vs **Unsigned**
    - Identification （syscall,Instruction）
    - Signed-extended vs Zero-extended
    - Caculation
  - **Big-endian** vs **Little-endian**
- 2. Logic operation, Shift operation

Identification Numbers

Run the demo to find the difference between two 'syscall' in the following demo :

```
.include "macro_print_str.asm"
.data
        tdata: .byte 0x0F00F0FF
.text
main:
        lb $a0,tdata
        li $v0,1
        syscall

        print_string("\n")
        lb $a0,tdata
        li $v0,36
        syscall

        end
```

Service	Code in $v0	Arguments	Result
print integer	1	$a0 = integer to print	
print integer as unsigned	36	$a0 = integer to print	Displayed as unsigned decimal value.

Both "print_string" and "end" are macros which had been defined in "macro_print_str.asm" file.

Signed vs Unsigned (extension)

```
.include "macro_print_str.asm"
.data
        tdata: .byte 0x80
.text
main:
        lb $a0,tdata
        li $v0,1
        syscall

        print_string("\n")
        lb $a0,tdata
        li $v0,36
        syscall

        end
```

```
.include "macro_print_str.asm"
.data
        tdata: .byte 0x80
.text
main:
        lbu $a0,tdata
        li $v0,1
        syscall

        print_string("\n")
        lbu $a0,tdata
        li $v0,36
        syscall

        end
```

Q1: Run the two demos, what's the value stored in the register $a0 after the operation of 'lb' and 'lbu'
Q2: using "-1" as initial value of tdata instead of "0x80", answer Q1 again.

Signed vs Unsigned (compare)

Run the demo to observe the comparison of signed and unsigned numbers

```
.include "macro_print_str.asm"
.data
.text
main:
        print_string("\n -1 less than 1 using slt:")
        li $t0,-1
        li $t1,1
        slt $a0,$t0,$t1
        li $v0,1
        syscall

        print_string("\n -1 less than 1 using sltu:")
        sltu $a0,$t0,$t1
        li $v0,1
        syscall
        end
```

TIPS:

1) slt $t1,$t2,$t3

set less than: if $t2 is less than $t3, then set $t1 to 1 else set $t1 to 0

2) sltu $t1,$t2,$t3

set less than unsigned: if $t2 is less than $t3 using unsigned comparision, then set $t1 to 1 else set $t1 to 0

Signed vs Unsigned (caculation)

Run the two demos, which one will invoke the exception (arithmetic overflow), why?

```
.include "macro_print_str.asm"
.data
        tdata: .word 0x11111111
.text
main:
        lw $t0,tdata
        addu $a0,$t0,$t0
        li $v0,1
        syscall

        print_string("\n")
        add $a0,$t0,$t0
        li $v0,1
        syscall

        end   #A
```

```
.include "macro_print_str.asm"
.data
        tdata: .word 0x71111111
.text
main:
        lw $t0,tdata
        addu $a0,$t0,$t0
        li $v0,1
        syscall

        print_string("\n")
        add $a0,$t0,$t0
        li $v0,1
        syscall

        end  #B
```

Big-endian vs Little-endian(1)

The CPU's **byte ordering scheme** (or **endian issues**) affects memory organization and defines the relationship between address and byte position of data in memory.

- a **Big-endian** system means byte 0 is always the most-significant (leftmost) byte.
- a L**ittle-endian** system means byte 0 is always the least-significant (rightmost) byte.

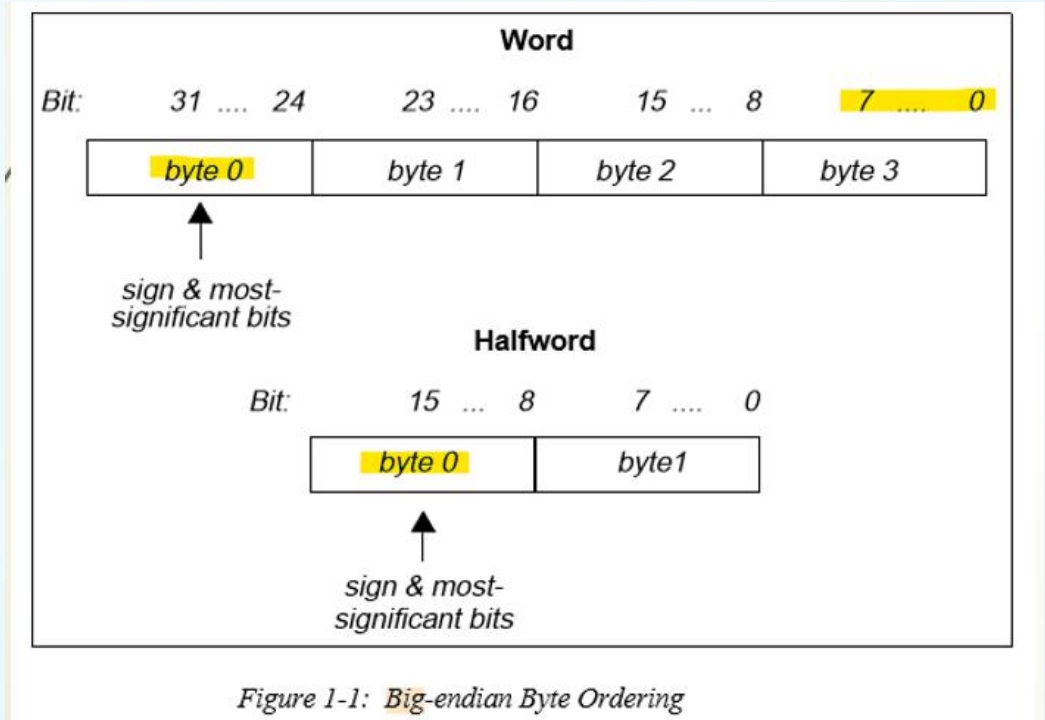


Figure 1-1: Big-endian Byte Ordering

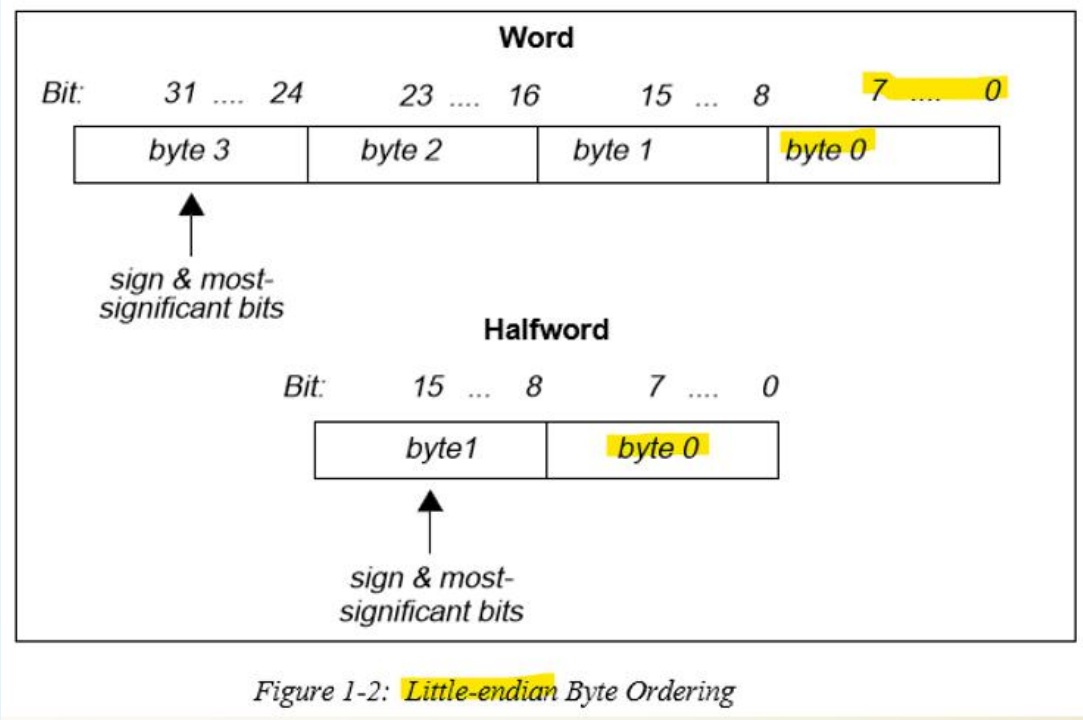


Figure 1-2: Little-endian Byte Ordering

Big-endian vs Little-endian(2)

Run the demo to anwer the question :
Does your simulator work on big-endian or little-endian, prove your conclusion.

```
.include "macro_print_str.asm"
.data
        tdata0: .byte   0x11,0x22,0x33,0x44
        tdata:   .word  0x44332211
.text
main:
        lb $a0,tdata
        li $v0,34
        syscall

        end
```

```
.include "macro_print_str.asm"
.data
        tdata0: .byte   0x11,0x22,0x33,0x44
        tdata:   .word  0x44332211
.text
main:
        lh $a0,tdata
        li $v0,34
        syscall

        end
```

print integer in hexadecimal	34	$a0 = integer to print	Displayed value is 8 hexadecimal digits, left-padding with zeroes if necessary.

Big-endian or Little-endian?

```
.include "macro_print_str.asm"
.data
        tdata0: .word   0x00112233, 0x44556677
.text
main:
        la $t0,tdata0
        lb $a0,($t0)
        li $v0,34
        syscall

        lb $a0,1($t0)
        syscall

        lb $a0,2($t0)
        syscall

        lb $a0,3($t0)
        syscall

        lw $a0,4($t0)
        syscall

        end
```

Run the demo to anwer the question :

Q1. What's the output of this demo?

*A. **0x**00000033**0x**00000022**0x**00000011**0x**00000000**0x**44556677*

*B.**0x**00000000**0x**00000011**0x**00000022**0x**00000033**0x**44556677*

*C.**0x**00000033**0x**00000022**0x**00000011**0x**00000000**0x**77665544*

*D.**0x**00000000**0x**00000011**0x**00000022**0x**00000033**0x**77665544*

Q2. Does your simulator work on big-endian or little-endian, explain the reasons.

print integer in hexadecimal	34	$a0 = integer to print	Displayed value is 8 hexadecimal digits, left-padding with zeroes if necessary.

Common Operations

Description	Op-code	Operand
Add with Overflow	add	destination, src1, src2
Add without Overflow	addu	destination, src1, src2
AND	and	destination, src1, immediate
Divide Signed	div	destination/src1, immediate
Divide Unsigned	divu	
Exclusive-OR	xor	
Multiply	mul	
Multiply with Overflow	mulo	
Multiply with Overflow Unsigned	mulou	
NOT OR	nor	
OR	or	
Set Equal	seq	
Set Greater	sgt	
Set Greater/Equal	sge	
Set Greater/Equal Unsigned	sgeu	
Set Greater Unsigned	sgtu	
Set Less	slt	
Set Less/Equal	sle	
Set Less/Equal Unsigned	sleu	
Set Less Unsigned	sltu	
Set Not Equal	sne	
Subtract with Overflow	sub	
Subtract without Overflow	subu	

Description	Op-code	Operand
Rotate Left	rol	
Rotate Right	ror	
Shift Right Arithmetic	sra	
Shift Left Logical	sll	
Shift Right Logical	srl	
Absolute Value	abs	destination,src1
Negate with Overflow	neg	destination/src1
Negate without Overflow	negu	
NOT	not	
Move	move	destination,src1
Multiply	mult	src1,src2
Multiply Unsigned	multu	

Logic Operation(1)

Instruction name	description
and (AND) *and dst,sr1,sr2(im)*	Computes the **Logical AND** of two values. This instruction ANDs (bit-wise) the contents of src1 with the contents of src2, or it can AND the contents of src1 with the immediate value. The immediate value is **NOT** sign extended. AND puts the result in the destination register.
or (OR) *or dst,sr1,sr2(im)*	Computes the **Logical OR** of two values. This instruction ORs (bit-wise) the contents of src1 with the contents of src2, or it can OR the contents of src1 with the immediate value. The immediate value is **NOT** sign extended. OR puts the result in the destination register
xor (Exclusive-OR) *xor dst,sr1,sr2(im)*	Computes the **XOR** of two values. This instruction XORs (bit-wise) the contents of src1 with the contents of src2, or it can XOR the contents of src1 with the immediate value. The immediate value is **NOT** sign extended. Exclusive-OR puts the result in the destination register
not (NOT) *not dst,src1*	Computes the **Logical NOT** of a value. This instruction complements (bit-wise) the contents of src1 and puts the result in the destination register.
nor (NOT OR) *nor dst,sr1,sr2*	Computes the **NOT OR** of two values. This instruction combines the contents of src1 with the contents of src2 (or the immediate value). NOT OR complements the result and puts it in the destination register.

Logic Operation(2)

Run the demo and answer the question :

```
.data
        dvalue1: .byte 27
        dvalue2: .byte 4
.text
        lb $t0,dvalue1
        lb $t1,dvalue2

        div $t0,$t1
        mfhi $a0

        li $v0,1
        syscall

        li $v0,10
        syscall
```

```
.data
        dvalue1: .byte 27
        dvalue2: .byte 4
.text
        lb $t0,dvalue1
        lb $t1,dvalue2

        sub $t1,$t1,1
        and $a0,$t0,$t1

        li $v0,1
        syscall

        li $v0,10
        syscall
```

Q1: Is the output of two demos the same?

Q2: If use 5 instead of 4 as the initial value on dvalue2, is the output of two demos the same?

Q3: On which situation could use 'and' operation to get the remainder instead of division?

Q4: Do the logic operations work quicker than arithmetic operations?

Shift Operation

Type	Instruction name	description	
shift	**sll** (Shift Left Logical)	**Shifts the contents of a register left** (toward the sign bit) and inserts zeros at the least-significant bit.	The contents of **src1 specify the value to shift**, and the contents of **src2 or the immediate value specify the amount to shift**. If src2 (or the immediate value) is greater than 31 or less than 0, src1 shifts by the result of src2 MOD 32.
	sra (Shift Right Arithmetic)	**Shifts the contents of a register right** (toward the least-significant bit) and inserts the sign bit at the most-significant bit.	
	srl (Shift Right Logical)	**Shifts the contents of a register right** (toward the least-significant bit) and inserts zeros at the most-significant bit.	
rotate	**rol** (Rotate Left)	**Rotates the contents of a register left** (toward the sign bit). This instruction inserts in the least-significant bit any bits that were shifted out of the sign bit.	The contents of **src1 specify the value to shift**, and the contents of **src2 (or the immediate value) specify the amount to shift**. Rotate Left/right puts the result in the destination register. If src2 (or the immediate value) is greater than 31, src1 shifts by the result of src2 MOD 32.
	ror (Rotate Right)	**Rotates the contents of a register right** (toward the least-significant bit). This instruction inserts in the sign bit any bits that were shifted out of the least-significant bit.	

Run the demo to see if the output is same with the sample picture below ?
if not please find the reason and modify the code.

```
.include "macro_print_str.asm"
.data
.text
main:
        print_string("please input an integer : ")
        li $v0,5
        syscall

        add  $t0,  $v0,     $zero
        nor  $t1,  $zero,   $zero
        sra  $t2,  $t1,     31
        and  $a0, $t2,      $t0

        print_string("it is an odd number (0: false,1:true) : ")
        li $v0,1
        syscall

        end
```

```
please input an integer : 3
it is an odd number (0: false,1:true) : 1
-- program is finished running --
```

Practice

1. Read an integer and store it in a word, print it in hexadecimal mode. Exchange byte0 and btye1 of the integer, and print out the exchanged data in hexadecimal mode. Exchange byte2 and btye3 of the integer, and print out the exchanged data in hexadecimal mode.

2. Write 3 demos which trigger exception by using subtraction,multiplication and division separately, tell the difference between these three exceptions.

3. Read a binary number ‘x’ and a bit width ‘y’ to generate a palindrome binary number. This binary palindrome number consists of ‘x’ with a bit width of ‘y’. For example:

   ‘x’ is $(101)_2$, ’y’ is 6, then the generated binary palindrome number is $(101101)_2$;

   ‘x’ is $(1)_2$, ’y’ is 3, then the generated binary palindrome number is $(111)_2$;

   ‘x’ is $(10)_2$, ’y’ is 3, the generation of binary palindrome number failed.

   ‘x’ is $(10)_2$, ’y’ is 4, the generation of binary palindrome number failed. $(1010)_2$ is not a binary palindrome number

4. “ rol” is a extened(pseudo) instuction. Find the basic instruction set of “ rol” and prove that they have the same function.

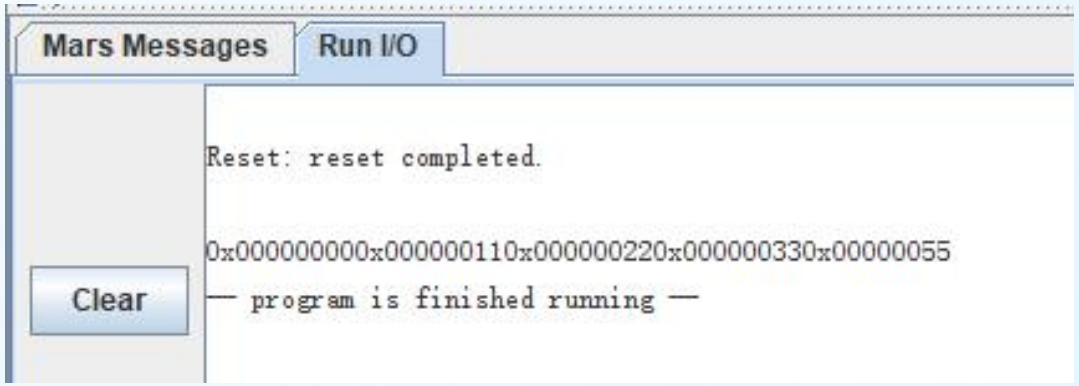
Tips： macro_print_str.asm

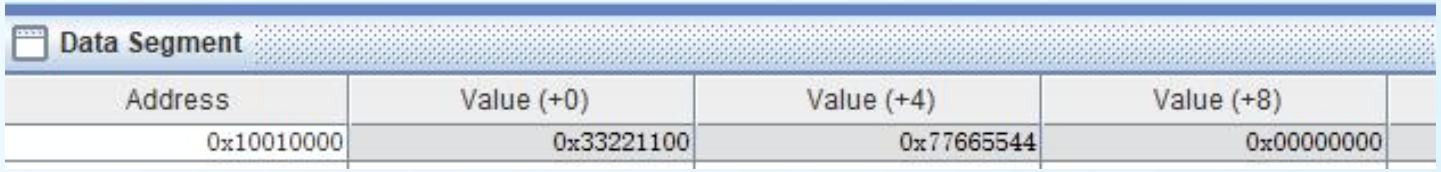
```
.macro print_string(%str)
        .data
                pstr:   .asciiz   %str
        .text
                la $a0,pstr
                li $v0,4
                syscall
.end_macro

.macro end
        li $v0,10
        syscall
.end_macro
```

Get help of defination and usage about macro from Mars‘ help page

Tips: the data address in Mars






```
.include "macro_print_str.asm"
.data
        tdata0: .byte
0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77
.text
main:
        la $t0,tdata0
        lb $a0, ($t0)
        li $v0,34
        syscall

        la $t0,tdata0
        lb $a0, 1($t0)
        syscall

        lb $a0, 2($t0)
        syscall

        lb $a0, 3($t0)
        syscall

        lb $a0, 5($t0)
        syscall

        end
```