Computer Organization

Lab3 MIPS(2) - data details

2021 Spring term

wangw6@sustech.edu.cn

TOPIC

- 1. Data Processing Details
  - **Signed** vs **Unsigned**
  - **Signed-extended** vs **Zero-extended**
  - **Exception** while processing signed data
  - **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 0xffffffff
.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-Extended vs Zero-Extended

```
.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 the initial value of tdata instead of "0x80", answer Q1 again.

Signed vs Unsigned (1)

Run the demo to find the difference between 'slt' and 'sltu'

```
.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
```

slt $t1,$t2,$t3

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

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 (2)

Run the two demos, which one will invoke the exception ,why?

```
.asm line : Runtime exception at 0          arithmetic overflow
```

```
.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
```

```
.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
```

Big-endian vs Little-endian(1)

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

- A **big-endian** system means byte 0 is always the most-significant (leftmost) byte.
- A **little-endian** system means byte 0 is always the least-significant (rightmost) byte.

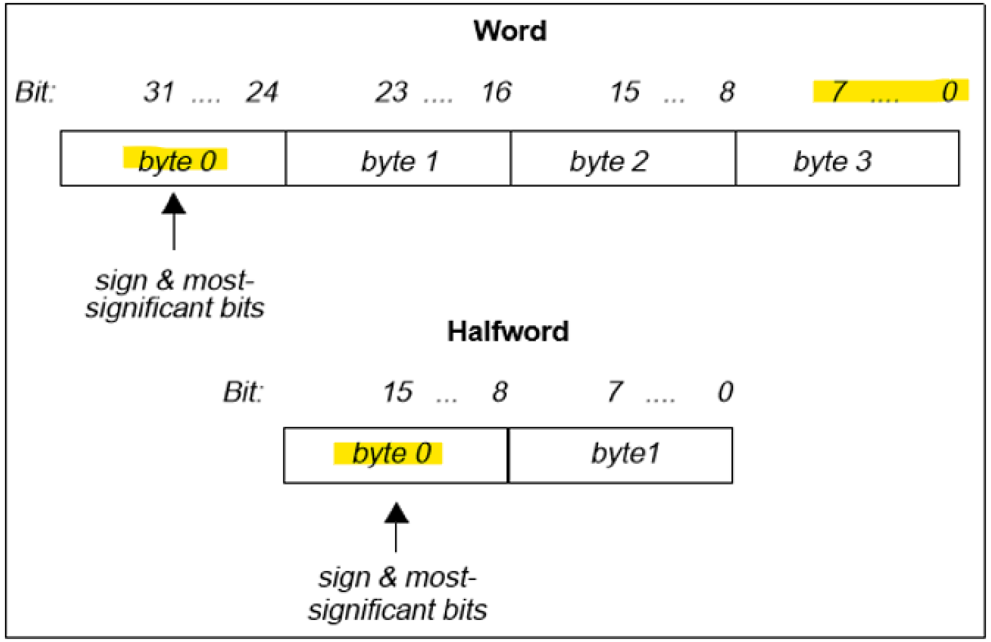

Figure 1-1: Big-endian Byte Ordering

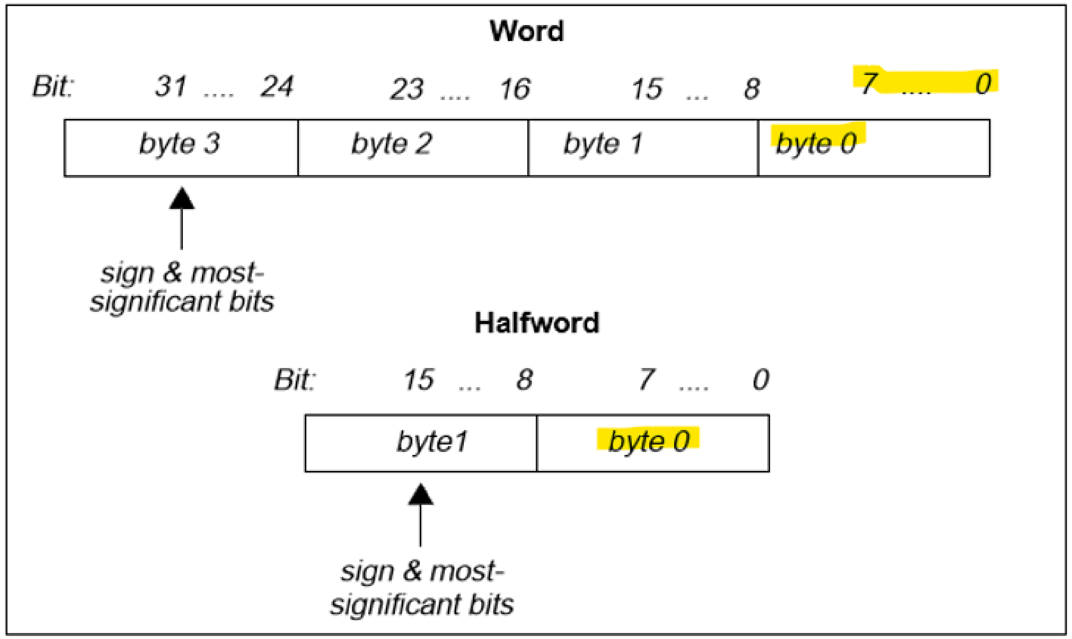

Figure 1-2: Little-endian Byte Ordering

Big-endian vs Little-endian(2)

Run the demo to anwer the question :
Which scheme does your simulator work, big-endian or little-endian?
Explain the reason.

```
.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
```

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)	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)	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
not (NOT)	Computes the **Logical NOT** of a value. This instruction complements (bit-wise) the contents of src1 and puts the result in the destination register.
xor (Exclusive-OR)	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
nor (NOT OR)	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 :
Q1: Are the outputs of following two demos the same?

```
.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
```

Q2: If use 5 instead of 4 as the initial value of dvalue2, are the outputs of following two demos the same?

Q3: When could the 'and' operation be used to get remainder after division?

Q4: Do logic operations work quicker than arithmetic operations?

Shift Operation

Instruction name	description
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 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. 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.
srl (Shift Right Logical)	**Shifts the contents of a register right** (toward the least-significant bit) and inserts zeros at the most-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 sr2 MOD 32.
rol **(** Rotate Left **)**	**Rotates the contents of a register left** (toward the sign bit). This instruction inserts the bits that are shifted out of the sign bit 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. Rotate Left puts the result in the destination register. If src2 (or the immediate value) is greater than 31, src1 shifts by (src2 MOD 32).
ror **(** Rotate Right **)**	**Rotates the contents of a register right** (toward the least-significant bit). This instruction inserts the bits that were shifted out of the least significant bit at 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 Right puts the result in the destination register. If src2 (or the immediate value) is greater than 32, src1 shifts by src2 MOD 32

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

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

	move	$t0,  $v0
	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

Choose 2 of following tasks to finish:

1. A word's value is 0x11223344, exchange the bytes of this word to get the new value 0x44332211.

2. Answer the questions from page 11 or page 13.

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

4. Calculate the absolute value of a word by basic operations other than abs.

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 Define and use macro form Mars' help page

Tips: data declearation

- **Directive** in assemble language:
  - **.byte** Store the listed value(s) as 8 bit bytes
  - **.half** Store the listed value(s) as 16 bit halfwords on halfword boundary
  - **.word** Store the listed value(s) as 32 bit words on word boundary
  - **.float** Store the listed value(s) as single precision floating point
  - **.double** Store the listed value(s) as double precision floating point
  - **.ascii** Store the string in the Data segment but do not add null terminator
  - **.asciiz** Store the string in the Data segment and add null terminator
  - **.space** Reserve the next specified number of bytes in Data segment
- Immediate count (16bit width)

Tips: Memory & Register