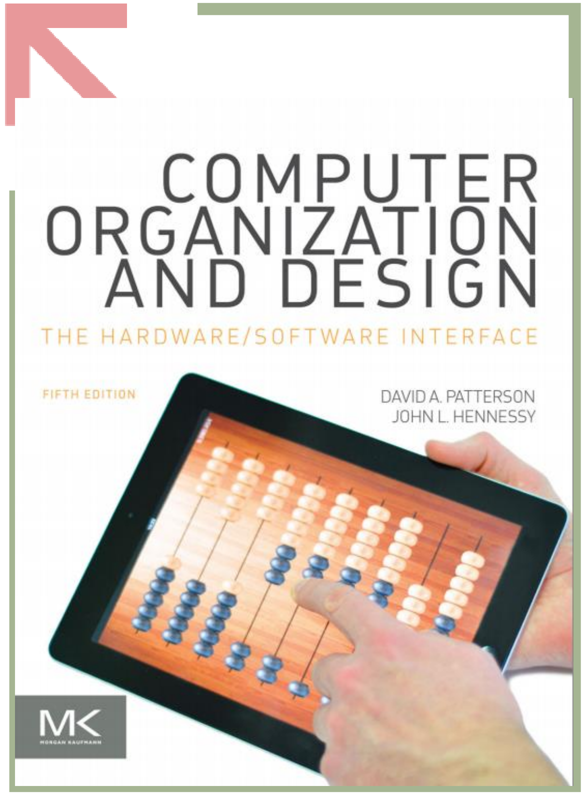
Assembly programming

data details

wangw6@sustc.edu.cn


COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
FIFTH EDITION
DAVID A. PATTERSON
JOHN L. HENNESSY
MK

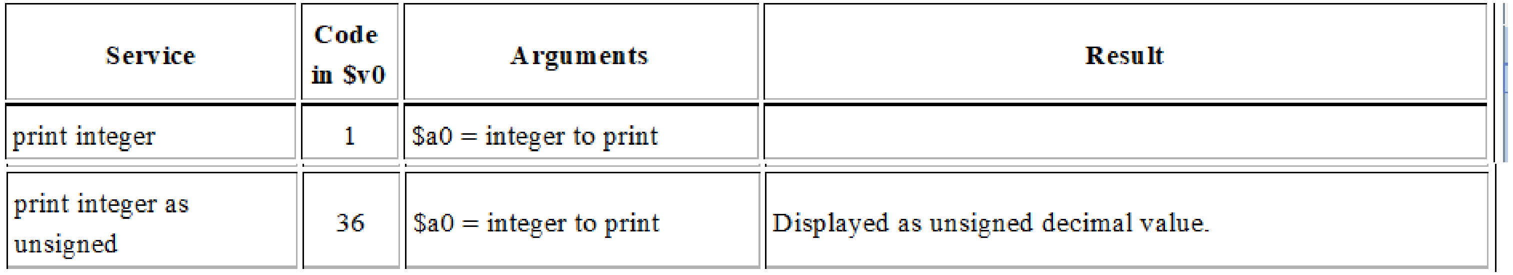
Overview

- 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

Print integer as signed or unsigned decimal value

Run the demo to find the difference between two ‘syscall’ in the 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.

Signed-extended vs unsigned-extended

Run the two demos, check the content of $a0 after the operator ‘lb’ and ‘lbu’

```
.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
```

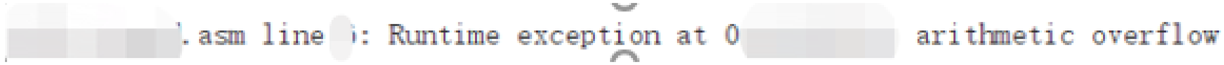
```
.include "macro_print_str.asm"
.data
        tdata: .byte 0x80
.text
main:
        lbu $a0,tdata
        li $v0,1
        syscall

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

        end
```

Calculation with signed and unsigned value (1)

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

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

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

        end
```

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

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

        end
```



```
[redacted].asm line [redacted]: Runtime exception at 0[redacted] arithmetic overflow
```

Calculation with signed and unsigned value (2)

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
```

Big-endian vs little-endian

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. While a little-endian system means byte 0 is always the least significant (rightmost byte).

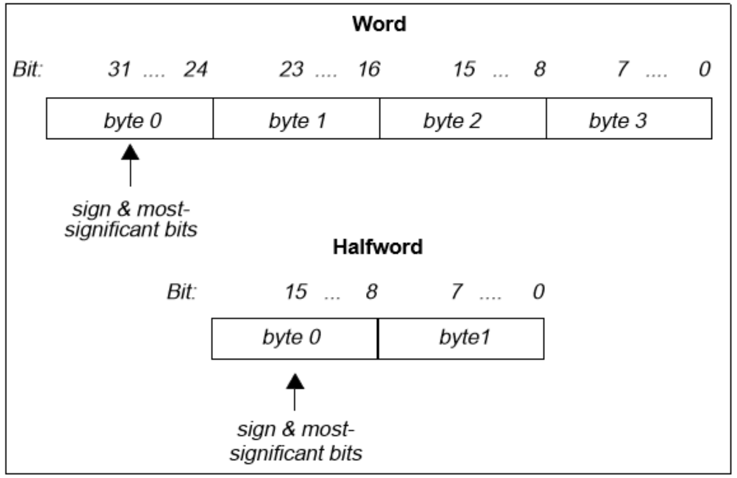

Word
Bit: 31 .... 24 23 .... 16 15 ... 8 7 0
byte 0
byte 1
byte 2
byte 3
sign & most-significant bits
Halfword
Bit: 15 ... 8 7 0
byte 0
byte1
sign & most-significant bits

Figure 1-1: Big-endian Byte Ordering

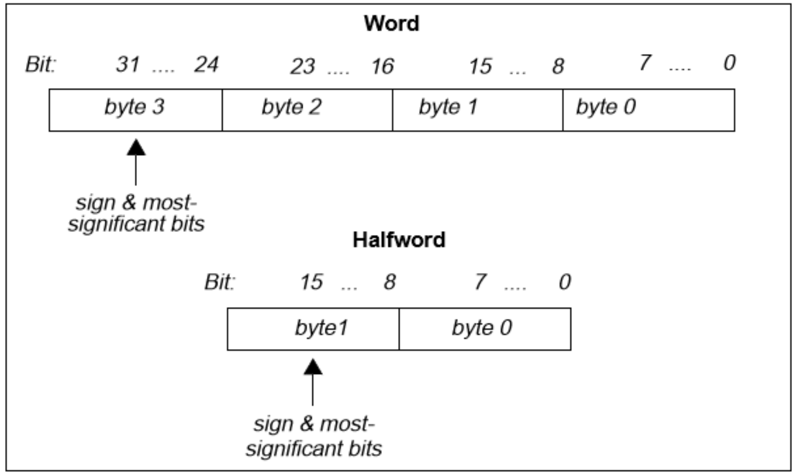

Word
Bit: 31 .... 24 23 .... 16 15 ... 8 7 0
byte 3
byte 2
byte 1
byte 0
sign & most-significant bits
Halfword
Bit: 15 ... 8 7 0
byte1
byte 0
sign & most-significant bits

Figure 1-2: Little-endian Byte Ordering

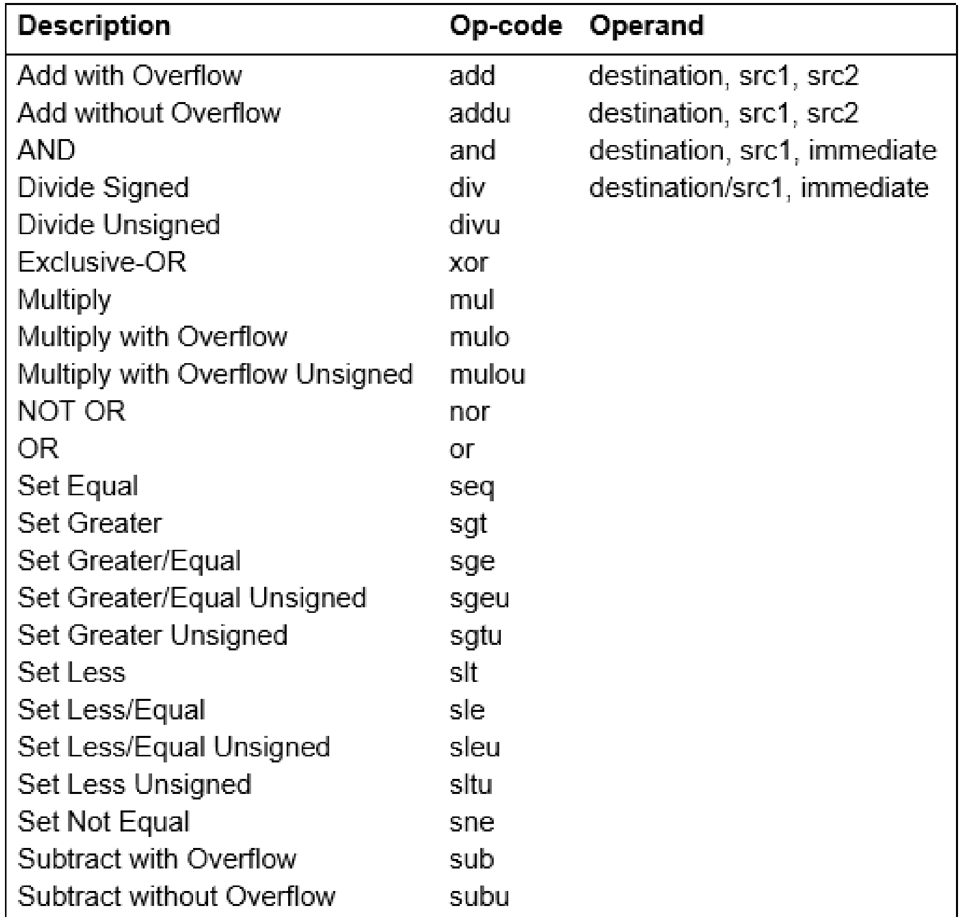
Is your computer big-endian or small-endian?

Run the demo to find the result

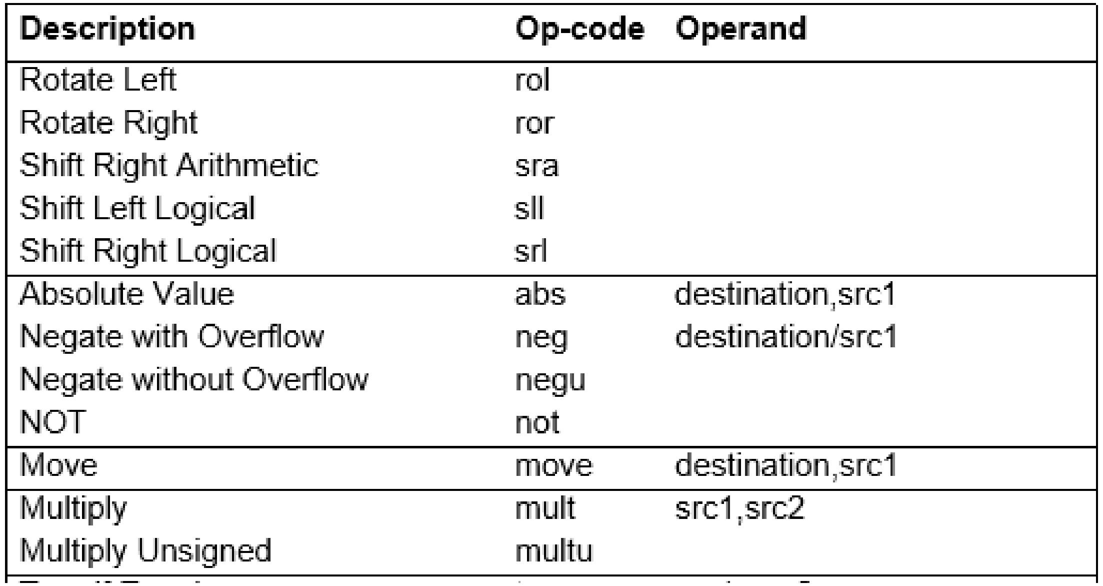
```
.include "macro_print_str.asm"
.data
        tdata: .byte 0x87654321
.text
main:
        lb $a0,tdata
        li $v0,34
        syscall

        end
```

```
.include "macro_print_str.asm"
.data
        tdata: .byte 0x87654321
.text
main:
        lh $a0,tdata
        li $v0,34
        syscall

        end
```

Logic operation and shift operation



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

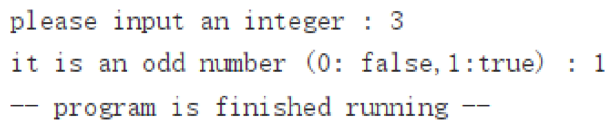
Instruction name	description
AND (**and**)	Computes the Logical AND of two values. This instruction ANDs (bit-wise) the contents of src1 with the contents of src2, or it can AND the contents of src1 with the immediate value. The immediate value is not sign extended. AND puts the result in the destination 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.
Exclusive-OR (**xor**)	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 OR(**nor**)	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.

Shift operation

Instruction name	description
Shift Left Logical (**sll**)	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.
Shift right Arithmetic (**sra**)	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.
Shift Right Logical (**srl**)	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.
Rotate Left (**rol**)	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 puts the result in the destination register. If src2 (or the immediate value) is greater than 31, src1 shifts by (src2 MOD 32).
Rotate Right (**ror**)	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. 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
        srl $t2,$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 --
```

Assignment（23:55 March 12, Tuesday）

1. Exchange the highest 8 bits with the lowest 8 bits in a word.

2. Calculate the bit inversion (0->1,1->0) of the odd digits in a word.

3. For an integer x, calculate the result of 10x. DO NOT use mult/mul/multu in your code.

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
```

Define and use .macro

Get help from the help page of Mars