汇编语言程序设计

第13讲 MIPS32指令集与编程

目录

MIPS32指令集

- 以经典的嵌入式处理器MIPS 4kc系列为参照

编程实例

- 使用“SPIM” MIPS系统模拟器
- http://spimsimulator.sourceforge.net/

指令分类（主要部分，不包括浮点）

算术（Arithmetic）指令（部分）

ADD	Add Word
ADDI	Add Immediate Word
ADDIU	Add Immediate Unsigned Word
ADDU	Add Unsigned Word
CLO	Count Leading Ones in Word
CLZ	Count Leading Zeros in Word
DIV	Divide Word
DIVU	Divide Unsigned Word
MADD	Multiply and Add Word to Hi, Lo
MADDU	Multiply and Add Unsigned Word to Hi, Lo
MSUB	Multiply and Subtract Word to Hi, Lo
MSUBU	Multiply and Subtract Unsigned Word to Hi, Lo
MUL	Multiply Word to GPR
MULT	Multiply Word
MULTU	Multiply Unsigned Word
SLT	Set on Less Than
SLTI	Set on Less Than Immediate
SLTIU	Set on Less Than Immediate Unsigned
SLTU	Set on Less Than Unsigned
SUB	Subtract Word
SUBU	Subtract Unsigned Word

举例

指令	Format	指令功能	其它
ADD	ADD rd, rs, rt	rd ← rs + rt	执行32位整数加法；如果补码运算溢出则产生异常
ADDI	ADDI rt, rs, immediate	rt ← rs + immediate	16位带符号立即数符号扩展后执行加法；如果补码运算溢出则产生异常
ADDU	ADDU rd, rs, rt	rd ← rs + rt	不产生异常

指令	Format	指令功能	其它
CLO	CLO rd, rs	rd←rs前导1的个数	X86指令集中有类似的BSF（Bit Scan Forward）、BSR指令
CLZ	CLZ rd, rs	rd←rs前导0的个数	

补充*

lib_c库中有相应的函数

- ffs, ffsl, ffsll - find first bit set in a word

```
#include <strings.h>
int ffs(int i);

 #include <string.h>
int ffsl(long int i);
int ffsll(long long int i);
```

指令	Format	指令功能	其它
MUL	MUL rd, rs, rt	rd ← rs × rt	32位整数相乘，结果只保留低32位；Hi/Lo寄存器无定义
MULT	MULT rs, rt	(HI, LO) ← rs× rt	32位带符号整数相乘，结果存于Hi/Lo寄存器
MULTU	MULTU rs, rt	(HI, LO) ← rs × rt	32位无符号整数相乘，结果存于Hi/Lo寄存器
DIV	DIV rs, rt	(HI, LO) ← rs / rt	32位带符号数… 不会产生算术异常（即便除以0）
DIVU	DIVU rs, rt	(HI, LO) ← rs / rt	32位无符号数… 不会产生算术异常（即便除以0）

指令	Format	指令功能	其它
MADD	MADD rs, rt	(HI,LO) ← (HI,LO) + (rs × rt)	32位带符号整数乘加
MADDU	？？		
MSUB	？？		
MSUBU	？？		
SLT	SLT rd, rs, rt	rd ← (rs < rt)	比较两个带符号32位整数，比较结果（1或者0）存入rd寄存器
SLTI	？？		
SLTIU	？？		
SLTU	？？		

分支（Branch）和跳转（Jump）指令（部分）

BEQ	Branch on Equal
BGEZ	Branch on Greater Than or Equal to Zero
BGEZAL	Branch on Greater Than or Equal to Zero and Link
BGTZ	Branch on Greater Than Zero
BLEZ	Branch on Less Than or Equal to Zero
BLTZ	Branch on Less Than Zero
BLTZAL	Branch on Less Than Zero and Link
BNE	Branch on Not Equal
J	Jump
JAL	Jump and Link
JALR	Jump and Link Register
JR	Jump Register

指令控制（Instruction Control）指令

NOP (伪指令)	No Operation（SLL, r0, r0, 0) 伪指令

PC指的是下一条指令地址(delay slot)

指令	Format	指令功能	其它
BEQ	BEQ rs, rt, offset	if rs = rt then branch	target_offset ← sign_extend(offset || 00); if (rs = rt) then PC ← PC + target_offset offset的宽度为16位
BGEZ	BGEZ rs, offset	if rs ≥ 0 then branch	…
BGEZAL	BGEZAL rs, offset	if rs ≥ 0 then procedure_call	… GPR[31] ← PC + 4

PC指的是下一条指令地址(delay slot)

指令	Format	指令功能	其它
J	J target	PC ← PC（高四位）|| (target || 00) (target 26位)	在当前的256MB对齐的空间内跳转
JAL	JAL target	GPR[31] ← PC + 4; PC ← PC（高四位）|| (target || 00)	在当前的256MB对齐的空间内执行过程调用
JALR	JALR rs (rd = 31 implied) JALR rd, rs	rd ← PC + 4 PC← rs	执行过程调用，过程入口地址位于rs内
JR	JR rs	PC ← rs	跳转至rs内存储的地址

装载（Load）、存储（Store）指令（部分）

LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned
LL	Load Linked Word
LW	Load Word
LWL	Load Word Left
LWR	Load Word Right
SB	Store Byte
SC	Store Conditional Word
SH	Store Halfword
SW	Store Word
SWL	Store Word Left
SWR	Store Word Right

这一对指令可以完成一个不对齐load操作

这一对指令可以完成一个不对齐store操作

指令	Format	指令功能	其它
LW	LW rt, offset(base)	从内存中读取一个字存入目的寄存器	rt ← memory[base+offset]（offset是16位带符号整数）地址必须4字节对齐，否则产生异常
LB	LB rt, offset(base)	从内存中读取一个字节，符号扩展后存入目的寄存器	rt ← sign_extend (memory[base+offsset])（offset是16位带符号整数）
LBU	LBU rt, offset(base)	无符号扩展，其它同上	…
SW	SW rt, offset(base)	从源寄存器读取字存入内存	…
SB	SB rt, offset(base)	从源寄存器读取低8位存入内存	…

单选题 0.5分

设置

MIPS32中函数调用指令（设该指令地址为N）的相应返回地址为(单位：字节）：

A N

B N+4

C N+8

提交

LL / SC指令

在多线程程序中，为了实现对共享变量的互斥访问，一般需要一个TestAndSet的原子操作

- 这种原子操作通常是需要专门的硬件支持才能完成

在MIPS中，是通过特殊的Load/Store指令：LL（Load Linked，链接加载）以及SC（Store Conditional，条件存储）这一‘指令对’完成的

- **当使用 LL 指令从内存中读取一个字之后，处理器会“记住” LL 指令的这次操作，同时 LL 指令读取的地址也会保存在处理器中**
- **接下来的 SC 指令，会检查上次 LL 指令执行后的操作是否是原子操作（即不存在其它对这个地址的操作)**
  - 如果是原子操作，则 V0（见如下示例）的值将会被更新至内存中，同时 V0 的值也会变为 1，表示操作成功
  - 反之，如果不是原子操作（即存在其它对这个地址的访问冲突），则V0 的值不会被更新至内存中，且 V0的值也会变为0，表示操作失败；如果成功，V0值设为1

```
atomic_inc:
    ll      v0, 0(a0)               # a0 has pointer to 'mycount'
    addu    v0, 1
    sc      v0, 0(a0)
    beq     v0, zero, atomic_inc    # retry if sc fails
    nop
    jr      ra
    nop
```

逻辑（Logical）指令 8条

AND	And
ANDI	And Immediate
LUI	Load Upper Immediate
NOR	Not Or
OR	Or
ORI	Or Immediate
XOR	Exclusive Or
XORI	Exclusive Or Immediate

转移（Move）指令 6条

MFHI	Move From HI Register
MFLO	Move From LO Register
MOVN	Move Conditional on Not Zero
MOVZ	Move Conditional on Zero
MTHI	Move To HI Register
MTLO	Move To LO Register

移位（Shift）指令 6条

SLL	Shift Word Left Logical
SLLV	Shift Word Left Logical Variable
SRA	Shift Word Right Arithmetic
SRAV	Shift Word Right Arithmetic Variable
SRL	Shift Word Right Logical
SRLV	Shift Word Right Logical Variable

指令	Format	指令功能	其它
AND	AND rd, rs, rt	针对32位寄存器执行逻辑与操作	rd ← rs AND rt
ANDI	ANDI rt, rs, immediate	针对32位寄存器与立即数（0扩展后）执行逻辑与操作	rt ← rs AND zero_extend(immediate)
LUI	LUI rt, immediate	将16位立即数装入目的寄存器的高16位（低16位清0）	rt ← immediate || 00…0(16)
MOVZ	MOVZ rd, rs, rt	条件移动	if rt = 0 then rd ← rs
SLL	SLL rd, rt, sa	左移操作	rd ← rt << sa sa是一个5位立即数（无符号）
SLLV	SLLV rd, rt, rs	左移操作	寄存器rs的低5位表示左移的位数

陷阱（Trap）指令

BREAK	Breakpoint
SYSCALL	**System Call**
TEQ	Trap if Equal
TEQI	Trap if Equal Immediate
TGE	Trap if Greater or Equal
TGEI	Trap if Greater of Equal Immediate
TGEIU	Trap if Greater or Equal Immediate Unsigned
TGEU	Trap if Greater or Equal Unsigned
TLT	Trap if Less Than
TLTI	Trap if Less Than Immediate
TLTIU	Trap if Less Than Immediate Unsigned
TLTU	Trap if Less Than Unsigned
TNE	Trap if Not Equal
TNEI	Trap if Not Equal Immediate

分支（Branch Likely）指令 （不再建议使用）

BEQL	Branch on Equal Likely
BGEZALL	Branch on Greater Than or Equal to Zero and Link Likely
BGEZL	Branch on Greater Than or Equal to Zero Likely
BGTZL	Branch on Greater Than Zero Likely
BLEZL	Branch on Less Than or Equal to Zero Likely
BLTZALL	Branch on Less Than Zero and Link Likely
BLTZL	Branch on Less Than Zero Likely
BNEL	Branch on Not Equal Likely

EJTAG指令（调试用）

DERET	Debug Exception Return
SDBBP	Software Debug Breakpoint

特权（Privileged）指令

CACHE	Perform Cache Operation
ERET	Exception Return
MFC0	Move from Coprocessor 0
MTC0	Move to Coprocessor 0
TLBP	Probe TLB for Matching Entry
TLBR	Read Indexed TLB Entry
TLBWI	Write Indexed TLB Entry
TLBWR	Write Random TLB Entry
WAIT	Enter Standby Mode

某些操作数的限制（如立即数宽度）增加了汇编编程难度；为降低难度，MIPS汇编器会做一些预处理

```
addu    $2, $4, 64     ⇒    addiu   $2, $4, 64
```

#立即数加法指令，不产生溢出异常

```
addu    $4, 0x12345    ⇒    li      at, 0x12345
                            addu    $4, $4, at
```

#装载立即数（因为立即数超过了16位二进制所能表示的范围）；at则是保留给汇编器使用的寄存器；**注意li也是一条伪指令**

```
li      $3, -5         ⇒    addiu   $3, $0, -5
li      $4, 0x8000     ⇒    ori     $4, $0, 0x8000
li      $5, 0x120000   ⇒    lui     $5, 0x12
li      $6, 0x12345    ⇒    lui     $6, 0x1
                            ori     $6, $6, 0x2345
```

请解释下这些指令（左侧）转换?

```
lw      $2, ($3)          ⇒   lw     $2, 0($3)
lw      $2, 8+4($3)       ⇒   lw     $2, 12($3)
lw      $2, addr          ⇒   lui    at, %hi(addr)
                              lw     $2, %lo(addr)(at)
sw      $2, addr($3)      ⇒   lui    at, %hi(addr)
                              addu   at, at, $3
                              sw     $2, %lo(addr)(at)
```

目录

MIPS32指令集

编程实例

- 使用“SPIM” MIPS系统模拟器
- http://spimsimulator.sourceforge.net/

MIPS汇编指示（Directives）

段说明　.text　.rdata　.data　.bss

```
.rdata
      msg:
         .asciiz "Hello world!\n"
.data
      table:
         .word 1
         .word 2
         .word 3
.text
      func:
         sub sp, 64
      ...
 .bss
        .comm  dbgflag, 4 # global common variable, 4 bytes
       .lcomm array, 300  # local common variable, 300 bytes
```

数据类型定义

.byte .half .word .dword .float .double .ascii .asciiz

```
.byte 3                      # 1 byte: 3
.half 1, 2, 3                # 3 half-words: 1 2 3
.word 5 : 3, 6, 7            # 5 words: 5 5 5 6 7

.float 1.4142175             # 1 single-precision value
.double 1e+10, 3.1415        # 2 double-precision values

.ascii "Hello\0"
.asciiz "Hello"
```

```
.align 4                     # 4-byte boundary对齐
var:
        .word   0
```

```
struc:
        .word   3
        .space  120                  # 120-byte的空间
        .word   -1
```

各类标识的属性

.globl .extern

```
.data
.globl status              # global variable
      status: .word 0
.text
.globl set_status          # global function
 set_status:
      subu sp, 24
....

.extern index, 4
.extern array, 100
lw $3, index($28)          # load a 4-byte(1-word) external
lw $2, array($28)          # load part of a 100-byte external
```

A global symbol. This directive enables the assembler to store the datum in a portion of the data segment that is efficiently accessed via register $gp (28#): Extern variables are stored in a 64KB memory area; $GP points to the middle of this area, and all subsequent accesses are based on $GP.

过程指示(不是必需的)

.ent .end

```
.text
.ent localfunc
 localfunc:
        addu $v0, $a1, $a2    # return (argl + arg2)
        j $ra
.end localfunc
```

示例一：分别计算整数数组中正数、负数的和

```
.data
array:
            .word -4, 5, 8, -1
msg1:
            .asciiz "\n The sum of the positive values = "
msg2:
            .asciiz "\n The sum of the negative values = "

.globl main
.text
main:
      li   $v0, 4              # system call code for print_str
      la   $a0, msg1           # load address of msg1. into $a0; pseudo-instruction
      syscall                  # print the string
      la $a0, array            # Initialize address Parameter
      li   $a1, 4              # Initialize length Parameter
      jal sum                  # Call sum
      nop                      # delay slot
      move $a0, $v0            # move value to be printed to $a0; pseudo-instruction
      li   $v0, 1              # system call code for print_int
      syscall                  # print sum of Pos:
```

```
li   $v0, 4               # system call code for print_str
la   $a0, msg2            # load address of msg2. into $a0
syscall                   # print the string
li   $v0, 1               # system call code for print_int
move $a0, $v1             # move value to be printed to $a0
syscall                   # print sum of neg

li   $v0, 10              # terminate program run and
syscall                   # return control to system
```

```
sum:
    li   $v0, 0
    li   $v1, 0                  # Initialize v0 and v1 to zero
loop:
    blez $a1, retzz              # If (a1 <= 0) Branch to Return
    nop
    addi $a1, $a1, -1            # Decrement loop count
    lw   $t0, 0($a0)             # Get a value from the array
    addi $a0, $a0, 4             # Increment array pointer to next
    bltz $t0, negg               # If value is negative Branch to negg
    nop
    add  $v0, $v0, $t0           # Add to the positive sum
    b    loop                    # Branch around the next two
                                 # instructions
    nop
negg:
    add  $v1, $v1, $t0           # Add to the negative sum
    b    loop                    # Branch to loop
    nop
retzz:
    jr   $ra                     # Return
    nop
```

SPIM模拟的系统调用

$v0

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

Regs | Int Regs [16] | Data | Text

Int Regs [16]

```
PC       = 40006c
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = a
R3  [v1] = fffffffb
R4  [a0] = fffffffb
R5  [a1] = 0
R6  [a2] = 7ffff3ac
R7  [a3] = 0
R8  [t0] = ffffffff
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

Text

```
[00400024] 34020004  ori $2, $0, 4            ; 8: li $v0, 4 # system call code for print_str
[00400028] 3c011001  lui $1, 4097 [msg1]      ; 9: la $a0, msg1 # load address of msg1. into $a0
[0040002c] 34240010  ori $4, $1, 16 [msg1]
[00400030] 0000000c  syscall                  ; 10: syscall # print the string
[00400034] 3c041001  lui $4, 4097 [array]     ; 11: la $a0, array # Initialize address Parameter
[00400038] 34050004  ori $5, $0, 4            ; 12: li $a1, 4 # Initialize length Parameter
[0040003c] 0c10001c  jal 0x00400070 [sum]     ; 13: jal sum # Call sum
[00400040] 00022021  addu $4, $0, $2          ; 14: move $a0, $v0 # move value to be printed to $a0
[00400044] 34020001  ori $2, $0, 1            ; 15: li $v0, 1 # system call code for print_int
[00400048] 0000000c  syscall                  ; 16: syscall # print sum of Pos:
[0040004c] 34020004  ori $2, $0, 4            ; 17: li $v0, 4 # system call code for print_str
[00400050] 3c011001  lui $1, 4097 [msg2]      ; 18: la $a0, msg2 # load address of msg2. into $a0
[00400054] 34240034  ori $4, $1, 52 [msg2]
[00400058] 0000000c  syscall                  ; 19: syscall # print the string
[0040005c] 34020001  ori $2, $0, 1            ; 20: li $v0, 1 # system call code for print_int
[00400060] 00032021  addu $4, $0, $3          ; 21: move $a0, $v1 # move value to be printed to $a0
[00400064] 0000000c  syscall                  ; 22: syscall # print sum of neg
```

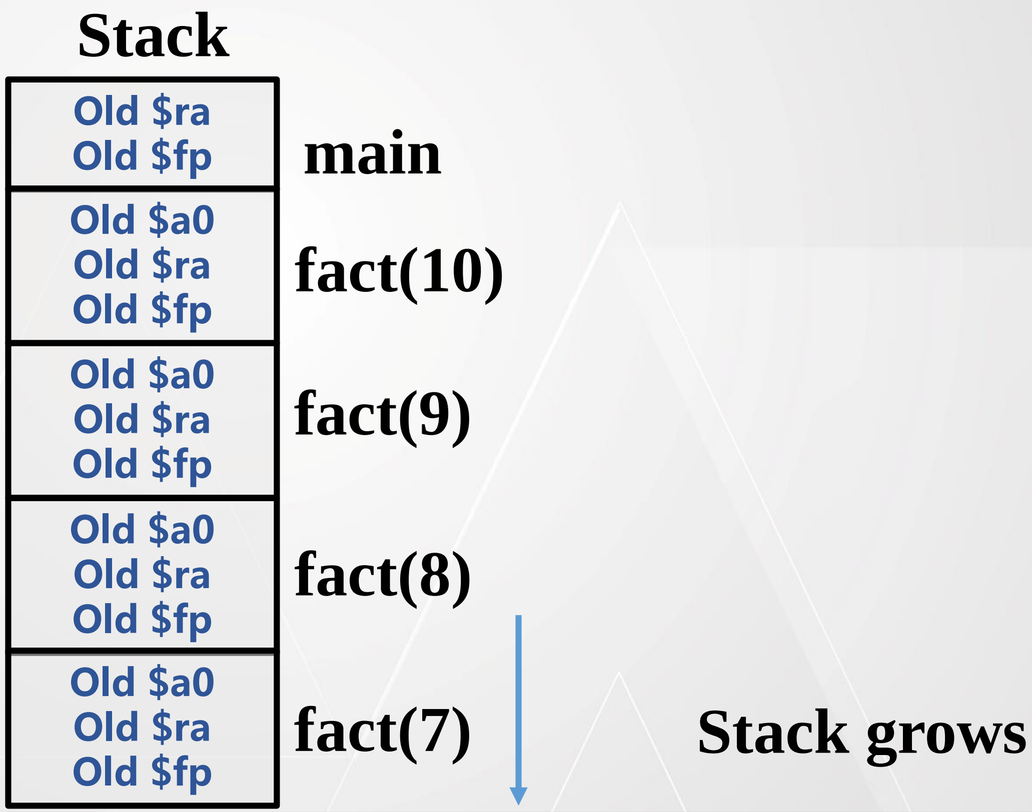
```
Memory and registers cleared
Loaded: C:/Users/Zhang/AppData/Local/Temp/qt_temp.gq5924
SPIM Version 9.1.7 of February 12, 2012
Copyright 1990-2012, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
```

17:17
2013-08-24

示例二：阶乘（delay slot关闭）

```
.text
.globl main
main:
    subu      $sp,$sp,32        # Stack frame is 32 bytes long
    sw        $ra,20($sp)       # Save return address
    sw        $fp,16($sp)       # Save old frame pointer
    addiu     $fp,$sp,28        # Set up frame pointer

    li        $a0,10            # Put argument (10) in $a0
    jal   fact                  # Call factorial function

    move      $a0,$v0           #
    li        $v0,1             # Print the result
    Syscall

    lw        $ra,20($sp)       # Restore return address
    lw        $fp,16($sp)       # Restore frame pointer
    addiu     $sp,$sp,32        # Pop stack frame

    li        $v0, 10           # terminate program run and
    syscall                     # return control to system
```

```
.text
fact:
    subu    $sp,$sp,32      # Stack frame is 32 bytes long
    sw      $ra,20($sp)     # Save return address
    sw      $fp,16($sp)     # Save frame pointer
    addiu   $fp,$sp,28      # Set up frame pointer
    sw      $a0,0($fp)      # Save argument (n)

    lw      $v0,0($fp)      # Load n
    bgtz    $v0,$L2         # Branch if n > 0
    li      $v0,1           # Return 1
    jr      $L1             # Jump to code to return
$L2:
    lw      $v1,0($fp)      # Load n
    subu    $v0,$v1,1       # Compute n - 1
    move    $a0,$v0         # Move value to $a0
```




```
    jal  fact                          # Call factorial function
    lw          $v1,0($fp)             # Load n
    mul         $v0,$v0,$v1            # Compute fact(n-1) * n

$L1:                                   # Result is in $v0
    lw          $ra, 20($sp)           # Restore $ra
    lw          $fp, 16($sp)           # Restore $fp
    addiu       $sp, $sp, 32           # Pop stack
    jr          $ra                    # Return to caller
```