CO2008 - KIẾN TRÚC MÁY TÍNH

Khoa Khoa Học và Kỹ Thuật Máy Tính
Đại học Bách Khoa – Tp.HCM

08/2019

Bài tập/Thực hành 2

CHƯƠNG 2 KIẾN TRÚC TẬP LỆNH MIPS: Lệnh đại số, luận lý, truy xuất dữ liệu

Mục tiêu

- Sử dụng thành thạo công cụ mô phỏng MARS. Biết cấu trúc một chương trình hợp ngữ MIPS.
- Sử dụng lệnh **syscall** để xuất/nhập dữ liệu, dùng trong hiển thị, debug
- Nắm được các lệnh luận lý, đại số trong hợp ngữ MIPS.
- Nắm được cách khai báo các kiểu dữ liệu và sử dụng được các lệnh về truy xuất dữ liệu (load/store).

Yêu cầu

- Tìm hiểu công cụ MARS và thực hành trên máy cá nhân.
- Xem các lệnh hợp ngữ trong slide/trong mục references trên bkelearning.
- Tham khảo tập lệnh nhanh cuối tài liệu này [trang 2].
- Nộp các file code hợp ngữ đặt tên theo format [Bai*.asm] chứa trong thư mục Lab2_MSSV

Bài tập và Thực hành

Bài 1: Syscall
Tham khảo manual của lệnh syscall trong phần help của công cụ MARS và hiện thực các yêu cầu dưới đây dùng lệnh syscall.

(a) Viết chương trình **nhập vào 3 số nguyên a, b, c** rồi **xuất ra màn hình giá trị của hàm f(a,b,c) = a - b + c**.

(b) Viết chương trình xuất ra chuỗi **"Kien Truc May Tinh 2019"**. (giống ví dụ HelloWorld!)

(c) Viết chương trình đọc vào một chuỗi 10 ký tự sau đó xuất ra màn hình chuỗi ký tự đó.

Bài 2: Các lệnh số học luận lý.

(a) Viết chương trình dùng các lệnh add, addi, sub, subi, or, ori ... để thực hiện phép tính bên dưới.

```
 100000 # This immediate number is greater than 16-bit
+  1000
-   100
```

Kết quả chứa vào thanh ghi $\$s_0$ và xuất kết quả ra màn hình (console).

Bài 3: Các lệnh về số học, phép nhân.
Viết chương trình tính giá trị biểu thức f(x) bên dưới. Kết quả lưu vào thanh ghi $\$s_0$ và xuất ra màn hình.

```
f = a.x^3 + b.x^2 - c.x - d
```

Dùng syscall để nhập a, b, c, d, x và xuất kết quả ra màn hình.
Gợi ý: (theo phương pháp Horner's Method, sinh viên có thể làm theo cách của riêng mình)

- Nhân a với x rồi lưu kết quả vào thanh ghi tạm. **t = a.x**
- Thực hiện phép số tính giữa thanh ghi tạm với b. **t = t + b** //t = a.x + b
- Nhân thanh ghi tạm với x. **t = t*x** //t = (ax + b)x
- Thực hiện phép số tính giữa thanh ghi tạm với c. **t = t – c** //t = a.x2 + b.x - c
- Nhân thanh ghi tạm với x. **t = t*x** // t = (ax2 + bx – c)x
- Thực hiện phép số tính giữa thanh ghi tạm với d. **t = t - d** // t = a.x3 + b.x2 – c.x - d

Bài 4: Lệnh load/store.

(a) Cho dãy số nguyên 10 phần tử, xuất ra kết quả là HIỆU của phần tử thứ 7 và 3. Mảng bắt đầu từ phần tử thứ 0.

(b) Chuyển đổi vị trí cuối và đầu của chuỗi "MSSV - Ho-Ten". Ví dụ chuỗi "123456 - Nguyen Van A" sẽ chuyển thành "A23456 - Nguyen Van 1". Sinh viên thay tên và mã số sinh viên của mình vào chuỗi trên

Làm thêm

1. Xác định các trường (OP, Rs, Rt, Rd, shamt, function, immediate) của các lệnh sau và chuyển các lệnh đó qua mã máy (dạng hex)

```
1 add  $t0, $s0, $a0          # add register to register
2 addi $v0, $a1, 200          # add register to immediate
3 lw   $t0, 4($a0)            # load word
4 sw   $t0, 4($a0)            # store word
5 lb   $t0, 4($a0)            # load byte
6 sb   $t0, 4($a0)            # store byte
7 sll  $t1, $s0, 5            # shift left logic (5-bit)
```

MIPS32® Instruction Set Quick Reference

R_D	—	Destination register
R_S, R_T	—	Source operand registers
R_A	—	Return address register (R31)
PC	—	Program counter
Acc	—	64-bit accumulator
Lo, Hi	—	Accumulator low ($Acc_{31:0}$) and high ($Acc_{63:32}$) parts
±	—	Signed operand or sign extension
∅	—	Unsigned operand or zero extension
::	—	Concatenation of bit fields
R2	—	MIPS32 Release 2 instruction
DOTTED	—	Assembler pseudo-instruction

Please refer to *"MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set"* for complete instruction set information.

Arithmetic Operations			
ADD	R_D, R_S, R_T	$R_D = R_S + R_T$	(overflow trap)
ADDI	R_D, R_S, const16	$R_D = R_S + const16^{\pm}$	(overflow trap)
ADDIU	R_D, R_S, const16	$R_D = R_S + const16^{\pm}$	
ADDU	R_D, R_S, R_T	$R_D = R_S + R_T$	
CLO	R_D, R_S	R_D = CountLeadingOnes(R_S)	
CLZ	R_D, R_S	R_D = CountLeadingZeros(R_S)	
LA	R_D, label	R_D = Address(label)	
LI	R_D, imm32	R_D = imm32	
LUI	R_D, const16	R_D = const16 << 16	
MOVE	R_D, R_S	$R_D = R_S$	
NEGU	R_D, R_S	$R_D = -R_S$	
SEB[R2]	R_D, R_S	$R_D = R_{S7:0}^{\pm}$	
SEH[R2]	R_D, R_S	$R_D = R_{S15:0}^{\pm}$	
SUB	R_D, R_S, R_T	$R_D = R_S - R_T$	(overflow trap)
SUBU	R_D, R_S, R_T	$R_D = R_S - R_T$	

Shift and Rotate Operations		
ROTR[R2]	R_D, R_S, bits5	$R_D = R_{SBITS5-1:0} :: R_{S31:BITS5}$
ROTRV[R2]	R_D, R_S, R_T	$R_D = R_{SRT4:0-1:0} :: R_{S31:RT4:0}$
SLL	R_D, R_S, shift5	$R_D = R_S$ << shift5
SLLV	R_D, R_S, R_T	$R_D = R_S << R_{T4:0}$
SRA	R_D, R_S, shift5	$R_D = R_S^{\pm} >>$ shift5
SRAV	R_D, R_S, R_T	$R_D = R_S^{\pm} >> R_{T4:0}$
SRL	R_D, R_S, shift5	$R_D = R_S^{\varnothing} >>$ shift5
SRLV	R_D, R_S, R_T	$R_D = R_S^{\varnothing} >> R_{T4:0}$

Logical and Bit-Field Operations		
AND	R_D, R_S, R_T	$R_D = R_S$ & R_T
ANDI	R_D, R_S, const16	$R_D = R_S$ & $const16^{\varnothing}$
EXT[R2]	R_D, R_S, P, S	$R_S = R_{SP+S-1:P}^{\varnothing}$
INS[R2]	R_D, R_S, P, S	$R_{DP+S-1:P} = R_{S_{S-1:0}}$
NOP		No-op
NOR	R_D, R_S, R_T	$R_D = \sim(R_S \mid R_T)$
NOT	R_D, R_S	$R_D = \sim R_S$
OR	R_D, R_S, R_T	$R_D = R_S \mid R_T$
ORI	R_D, R_S, const16	$R_D = R_S \mid const16^{\varnothing}$
WSBH[R2]	R_D, R_S	$R_D = R_{S23:16} :: R_{S31:24} :: R_{S7:0} :: R_{S15:8}$
XOR	R_D, R_S, R_T	$R_D = R_S \oplus R_T$
XORI	R_D, R_S, const16	$R_D = R_S \oplus const16^{\varnothing}$

Condition Testing and Conditional Move Operations		
MOVN	R_D, R_S, R_T	if $R_T \neq 0$, $R_D = R_S$
MOVZ	R_D, R_S, R_T	if $R_T = 0$, $R_D = R_S$
SLT	R_D, R_S, R_T	$R_D = (R_S^{\pm} < R_T^{\pm})$? 1 : 0
SLTI	R_D, R_S, const16	$R_D = (R_S^{\pm} < const16^{\pm})$? 1 : 0
SLTIU	R_D, R_S, const16	$R_D = (R_S^{\varnothing} < const16^{\varnothing})$? 1 : 0
SLTU	R_D, R_S, R_T	$R_D = (R_S^{\varnothing} < R_T^{\varnothing})$? 1 : 0

Multiply and Divide Operations		
DIV	R_S, R_T	$Lo = R_S^{\pm} / R_T^{\pm}$; $Hi = R_S^{\pm}$ mod $R_T^{\pm}$
DIVU	R_S, R_T	$Lo = R_S^{\varnothing} / R_T^{\varnothing}$; $Hi = R_S^{\varnothing}$ mod $R_T^{\varnothing}$
MADD	R_S, R_T	$Acc \mathrel{+}= R_S^{\pm} \times R_T^{\pm}$
MADDU	R_S, R_T	$Acc \mathrel{+}= R_S^{\varnothing} \times R_T^{\varnothing}$
MSUB	R_S, R_T	$Acc \mathrel{-}= R_S^{\pm} \times R_T^{\pm}$
MSUBU	R_S, R_T	$Acc \mathrel{-}= R_S^{\varnothing} \times R_T^{\varnothing}$
MUL	R_D, R_S, R_T	$R_D = R_S^{\pm} \times R_T^{\pm}$
MULT	R_S, R_T	$Acc = R_S^{\pm} \times R_T^{\pm}$
MULTU	R_S, R_T	$Acc = R_S^{\varnothing} \times R_T^{\varnothing}$

Accumulator Access Operations		
MFHI	R_D	$R_D = Hi$
MFLO	R_D	$R_D = Lo$
MTHI	R_S	$Hi = R_S$
MTLO	R_S	$Lo = R_S$

Jumps And Branches (Note: One Delay Slot)		
B	off18	PC += $off18^{\pm}$
BAL	off18	R_A = PC + 8, PC += $off18^{\pm}$
BEQ	R_S, R_T, off18	if $R_S = R_T$, PC += $off18^{\pm}$
BEQZ	R_S, off18	if $R_S = 0$, PC += $off18^{\pm}$
BGEZ	R_S, off18	if $R_S \geq 0$, PC += $off18^{\pm}$
BGEZAL	R_S, off18	R_A = PC + 8; if $R_S \geq 0$, PC += $off18^{\pm}$
BGTZ	R_S, off18	if $R_S > 0$, PC += $off18^{\pm}$
BLEZ	R_S, off18	if $R_S \leq 0$, PC += $off18^{\pm}$
BLTZ	R_S, off18	if $R_S < 0$, PC += $off18^{\pm}$
BLTZAL	R_S, off18	R_A = PC + 8; if $R_S < 0$, PC += $off18^{\pm}$
BNE	R_S, R_T, off18	if $R_S \neq R_T$, PC += $off18^{\pm}$
BNEZ	R_S, off18	if $R_S \neq 0$, PC += $off18^{\pm}$
J	addr28	PC = $PC_{31:28}$ :: $addr28^{\varnothing}$
JAL	addr28	R_A = PC + 8; PC = $PC_{31:28}$ :: $addr28^{\varnothing}$
JALR	R_D, R_S	R_D = PC + 8; PC = R_S
JR	R_S	PC = R_S

Load and Store Operations		
LB	R_D, off16(R_S)	R_D = $mem8(R_S + off16^{\pm})^{\pm}$
LBU	R_D, off16(R_S)	R_D = $mem8(R_S + off16^{\pm})^{\varnothing}$
LH	R_D, off16(R_S)	R_D = $mem16(R_S + off16^{\pm})^{\pm}$
LHU	R_D, off16(R_S)	R_D = $mem16(R_S + off16^{\pm})^{\varnothing}$
LW	R_D, off16(R_S)	R_D = $mem32(R_S + off16^{\pm})$
LWL	R_D, off16(R_S)	R_D = LoadWordLeft(R_S + $off16^{\pm}$)
LWR	R_D, off16(R_S)	R_D = LoadWordRight(R_S + $off16^{\pm}$)
SB	R_S, off16(R_T)	$mem8(R_T + off16^{\pm}) = R_{S7:0}$
SH	R_S, off16(R_T)	$mem16(R_T + off16^{\pm}) = R_{S15:0}$
SW	R_S, off16(R_T)	$mem32(R_T + off16^{\pm}) = R_S$
SWL	R_S, off16(R_T)	StoreWordLeft(R_T + $off16^{\pm}$, R_S)
SWR	R_S, off16(R_T)	StoreWordRight(R_T + $off16^{\pm}$, R_S)
ULW	R_D, off16(R_S)	R_D = unaligned_$mem32(R_S + off16^{\pm})$
USW	R_S, off16(R_T)	unaligned_$mem32(R_T + off16^{\pm}) = R_S$

Atomic Read-Modify-Write Operations		
LL	R_D, off16(R_S)	R_D = $mem32(R_S + off16^{\pm})$; link
SC	R_D, off16(R_S)	if Atomic, $mem32(R_S + off16^{\pm}) = R_D$; R_D = Atomic ? 1 : 0

Registers

0	zero	Always equal to zero
1	at	Assembler temporary; used by the assembler
2-3	v0-v1	Return value from a function call
4-7	a0-a3	First four parameters for a function call
8-15	t0-t7	Temporary variables; need not be preserved
16-23	s0-s7	Function variables; must be preserved
24-25	t8-t9	Two more temporary variables
26-27	k0-k1	Kernel use registers; may change unexpectedly
28	gp	Global pointer
29	sp	Stack pointer
30	fp/s8	Stack frame pointer or subroutine variable
31	ra	Return address of the last subroutine call

Default C Calling Convention (O32)

Stack Management

- The stack grows down.
  - Subtract from $sp to allocate local storage space.
  - Restore $sp by adding the same amount at function exit.
- The stack must be 8-byte aligned.
  - Modify $sp only in multiples of eight.

Function Parameters

- Every parameter smaller than 32 bits is promoted to 32 bits.
- First four parameters are passed in registers $a0–$a3.
  - 64-bit parameters are passed in register pairs:
    - Little-endian mode: $a1:$a0 or $a3:$a2.
    - Big-endian mode: $a0:$a1 or $a2:$a3.
- Every subsequent parameter is passed through the stack.
  - First 16 bytes on the stack are not used.
  - Assuming $sp was not modified at function entry:
    - The 1st stack parameter is located at 16($sp).
    - The 2nd stack parameter is located at 20($sp), etc.
  - 64-bit parameters are 8-byte aligned.

Return Values

- 32-bit and smaller values are returned in register $v0.
- 64-bit values are returned in registers $v0 and $v1:
  - Little-endian mode: $v1:$v0.
  - Big-endian mode: $v0:$v1.

MIPS32 Virtual Address Space

kseg3	0xE000.0000	0xFFFF.FFFF	Mapped	Cached
ksseg	0xC000.0000	0xDFFF.FFFF	Mapped	Cached
kseg1	0xA000.0000	0xBFFF.FFFF	Unmapped	Uncached
kseg0	0x8000.0000	0x9FFF.FFFF	Unmapped	Cached
useg	0x0000.0000	0x7FFF.FFFF	Mapped	Cached

Reading The Cycle Count Register From C

```
unsigned mips_cycle_counter_read()
{
    unsigned cc;
    asm volatile("mfc0 %0, $9" : "=r" (cc));
    return (cc << 1);
}
```

Assembly-Language Function Example

```
# int asm_max(int a, int b)
# {
#   int r = (a < b) ? b : a;
#   return r;
# }

    .text
    .set     nomacro
    .set     noreorder

    .global  asm_max
    .ent     asm_max

asm_max:
    move     $v0, $a0         # r = a
    slt      $t0, $a0, $a1    # a < b ?
    jr       $ra              # return
    movn     $v0, $a1, $t0    # if yes, r = b

    .end     asm_max
```

C / Assembly-Language Function Interface

```
#include <stdio.h>

int asm_max(int a, int b);

int main()
{
    int x = asm_max(10, 100);
    int y = asm_max(200, 20);
    printf("%d %d\n", x, y);
}
```

Invoking MULT And MADD Instructions From C

```
int dp(int a[], int b[], int n)
{
    int i;
    long long acc = (long long) a[0] * b[0];
    for (i = 1; i < n; i++)
        acc += (long long) a[i] * b[i];
    return (acc >> 31);
}
```

Atomic Read-Modify-Write Example

```
atomic_inc:
    ll       $t0, 0($a0)      # load linked
    addiu    $t1, $t0, 1      # increment
    sc       $t1, 0($a0)      # store cond'l
    beqz     $t1, atomic_inc  # loop if failed
    nop
```

Accessing Unaligned Data

note: ULW and USW automatically generate appropriate code

Little-Endian Mode		Big-Endian Mode	
LWR	RD, OFF16(RS)	LWL	RD, OFF16(RS)
LWL	RD, OFF16+3(RS)	LWR	RD, OFF16+3(RS)
SWR	RD, OFF16(RS)	SWL	RD, OFF16(RS)
SWL	RD, OFF16+3(RS)	SWR	RD, OFF16+3(RS)

Accessing Unaligned Data From C

```
typedef struct
{
    int u;
} __attribute__((packed)) unaligned;


int unaligned_load(void *ptr)
{
    unaligned *uptr = (unaligned *)ptr;
    return uptr->u;
}
```

MIPS SDE-GCC Compiler Defines

__mips	MIPS ISA (= 32 for MIPS32)
__mips_isa_rev	MIPS ISA Revision (= 2 for MIPS32 R2)
__mips_dsp	DSP ASE extensions enabled
_MIPSEB	Big-endian target CPU
_MIPSEL	Little-endian target CPU
_MIPS_ARCH_*CPU*	Target CPU specified by -march=*CPU*
_MIPS_TUNE_*CPU*	Pipeline tuning selected by -mtune=*CPU*

Notes

- Many assembler pseudo-instructions and some rarely used machine instructions are omitted.
- The C calling convention is simplified. Additional rules apply when passing complex data structures as function parameters.
- The examples illustrate syntax used by GCC compilers.
- Most MIPS processors increment the cycle counter every other cycle. Please check your processor documentation.