



KIẾN TRÚC MÁY TÍNH

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

08/2020

Bài tập/Thực hành 4

CHƯƠNG 2 KIẾN TRÚC TẬP LỆNH MIPS: GỌI HÀM (LẬP TRÌNH CẤU TRÚC), THỜI GIAN THỰC THI

Mục tiêu

- Chuyển từ ngôn ngữ cấp cao (C) sang hợp ngữ MIPS.
- Sử dụng lệnh điều khiển (nhảy, rẽ nhánh) để lập trình cấu trúc.
- Biết nguyên lý gọi hàm. Sử dụng các lệnh gọi hàm jal, jr.
- Tính toán thời gian thực thi của chương trình.

Yêu cầu

- Xem cách dùng các lệnh (set, branch, jump, load, store, **jal**, **jr**) trong slide và trong file tham khảo.
- Nộp các file code hợp ngữ đặt tên theo format [Bai*.asm] chứa trong thư mục Lab4_MSSV

Tập lệnh [tham khảo nhanh]

Cú pháp	Ảnh hưởng	Mô tả
slt Rd, Rs, Rt	$Rd = (Rs < Rt) ? 1 : 0$	[Có dấu] $Rd = 1$ khi $Rs < Rt$, ngược lại $Rd = 0$
sltu Rd, Rs, Rt	$Rd = (Rs < Rt) ? 1 : 0$	[Không dấu] $Rd = 1$ khi $Rs < Rt$, ngược lại $Rd = 0$
Lệnh nhảy, rẽ nhánh		
beq Rs, Rt, label	if ($Rs == Rt$) $PC \leftarrow label$	Rẽ nhánh đến label nếu $Rs == Rt$
bne Rs, Rt, label	if ($Rs != Rt$) $PC \leftarrow label$	Rẽ nhánh đến label nếu $Rs != Rt$
bltz Rs, label	if ($Rs < 0$) $PC \leftarrow label$	Rẽ nhánh đến label nếu $Rs < 0$
blez Rs, label	if ($Rs \leq 0$) $PC \leftarrow label$	Rẽ nhánh đến label nếu $Rs \leq 0$
bgtz Rs, label	if ($Rs > 0$) $PC \leftarrow label$	Rẽ nhánh đến label nếu $Rs > 0$
bgez Rs, label	if ($Rs \geq 0$) $PC \leftarrow label$	Rẽ nhánh đến label nếu $Rs \geq 0$
j label	$PC \leftarrow label$	Nhảy không điều kiện đến label
Gọi hàm		
jr Rs	$PC \leftarrow Rs$	Trở về vị trí thanh ghi Rs trở đến
jal label	$\$ra \leftarrow PC+4, PC \leftarrow label$	Gọi hàm label, khi đó \$ra nắm vị trí lệnh tiếp theo
jalr Rs	$\$ra \leftarrow PC+4, PC \leftarrow Rs$	Gọi hàm Rs đang trở đến, khi đó \$ra nắm vị trí lệnh tiếp theo

Review: MIPS instruction types

R-type

Op_6	Rs_5	Rt_5	Rd_5	$Shamt_5$	$Function_6$
--------	--------	--------	--------	-----------	--------------

Kiểu I-type

Op_6	Rs_5	Rt_5	$Immediate_{16}$
--------	--------	--------	------------------

Kiểu J-type

Op_6	$Immediate_{26}$
--------	------------------

- Op (opcode) Mã lệnh, dùng để xác định lệnh thực thi (trong kiểu R, Op = 0).
- Rs, Rt, Rd (register): Trường xác định thanh ghi (trường thanh ghi 5 bit tương ứng với 32 thanh ghi).
- Shamt (shift amount): Xác định số bits dịch trong các lệnh dịch bit.
- Function: Xác định toán tử(operator hay còn gọi là lệnh) trong kiểu lệnh R.
- Immediate: Đại diện cho con số trực tiếp, địa chỉ, offset.

Bài tập và Thực hành

Sinh viên chuyển chương trình C bên dưới qua hợp ngữ MIPS tương ứng.

1. Leaf function (hàm lá)

Chuyển thủ tục "reverse" (đảo thứ tự chuỗi) từ ngôn ngữ C sang hợp ngữ MIPS. Thủ tục reverse được gọi khi thực thi lệnh `jal reverse` từ vùng `.text`. `cArray`, `cArray_size` được gắn vào các thanh ghi thanh ghi `$a0`, `$a1`. Giá trị trả về (nếu có) chứa vào `$v0`. Xuất chuỗi ra console.

```

1 char[] cArray = "Computer Architecture 2020"
2 int cArray_size = 26;
3 void reverse(char[] cArray, int cArray_size)
4 {
5     int i;
6     char temp;
7     for (i = 0; i < cArray_size/2; i++)
8     {
9         temp = cArray[i];
10        cArray[i] = cArray[cArray_size - 1 - i];
11        cArray[cArray_size - 1 - i] = temp;
12    }
13 }
```

Lưu ý: Dùng `"jal reverse"` để gọi thủ tục "reverse" và dùng `"jr $ra"` trở về vị trí thanh ghi `$ra` đánh dấu.

2. Non-leaf function (là hàm/thủ tục gọi một hàm/thủ tục bên trong).

Chuyển thủ tục range từ C sang hợp ngữ MIPS tương đương.

```

1 int iArray[10];
2 int iArray_size = 10;
3 int range(iArray, iArray_size)
4 {
5     int temp1 = max(iArray, iArray_size);
6     int temp2 = min(iArray, iArray_size);
7     int range = temp1 - temp2;
8     return range;
9 }
```

Chương trình bắt đầu từ vùng `.text`, sau đó nó gọi hàm `range`. Trong hàm `range` lại gọi 2 hàm con là `max` và `min`. Giả sử địa chỉ và kích thước `iarray` được gắn lần lượt vào các thanh ghi `$a0`, `$a1`. Xuất giá trị `range` ra ngoài console.

Lưu ý: Khi gọi các hàm/thủ tục thanh ghi `$ra` sẽ tự đánh dấu lệnh tiếp theo như là vị trí trở về. Do đó trước khi gọi hàm con trong hàm `range` thì sinh viên cần lưu lại giá trị thanh ghi `$ra` trong stack. Sau khi thực thi xong, sinh viên cần phục hồi lại giá trị cho thanh ghi `$ra` từ stack. Dùng `"jal range"`, `"jal max"`, `"jal min"` để gọi thủ tục `range`, `max`, `min`. Dùng `"jr $ra"` để trở về vị trí lệnh mà thanh ghi `$ra` đã đánh dấu.

Để lưu (push) giá trị $\$r_a$ vào stack, ta có thể dùng các lệnh sau:

```

1 addi $sp, $sp, -4    # adjust stack for 1 item
2 sw   $ra, 0($sp)     # save return address
```

Để phục hồi (pop) $\$r_a$ từ stack, ta có thể dùng các lệnh sau:

```
1 lw    $ra, 0($sp)      # restore return address
2 addi $sp, $sp, 4      # pop 1 item from stack
```

3. Cho đoạn code hợp ngữ MIPS bên dưới

```
1      addi $a0, $zero, 100    // upper threshold
2      addi $a1, $zero, 0      // count variable
3      add  $a2, $zero, $zero  // sum initialization
4 loop:
5      beq  $a0, $a1, exit
6      add  $a2, $a2, $a1
7      addi $a1, $a1, 1
8      j    loop
9 exit:
```

- (a) Xác định giá trị của thanh ghi \$a2 sau khi thực thi đoạn code trên.
- (b) Xác định tổng số chu kỳ thực thi khi thực thi đoạn chương trình trên. Giả sử CPI của các lệnh là 1.
- (c) Giả sử vùng .text (text segment - vùng để chứa các lệnh thực thi) bắt đầu từ địa chỉ 0x10080000. Xác định mã máy của lệnh "**j loop**" ở dạng HEX.

MIPS32® Instruction Set
Quick Reference

- Rd
- Rs, Rt
- Ra
- PC
- Acc
- Lo, Hi
- ±
- ∅
- ::
- R2
- DOTTED
- DESTINATION REGISTER
- SOURCE OPERAND REGISTERS
- RETURN ADDRESS REGISTER (R31)
- PROGRAM COUNTER
- 64-BIT ACCUMULATOR
- 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
- MIPS32 RELEASE 2 INSTRUCTION
- ASSEMBLER PSEUDO-INSTRUCTION

PLEASE REFER TO “MIPS32 ARCHITECTURE FOR PROGRAMMERS VOLUME II:
The MIPS32 INSTRUCTION SET” FOR COMPLETE INSTRUCTION SET INFORMATION.

ARITHMETIC OPERATIONS		
ADD	Rd, Rs, Rt	Rd = Rs + Rt (OVERFLOW TRAP)
ADDI	Rd, Rs, CONST16	Rd = Rs + CONST16 [±] (OVERFLOW TRAP)
ADDIU	Rd, Rs, CONST16	Rd = Rs + CONST16 [±]
ADDU	Rd, Rs, Rt	Rd = Rs + Rt
CLO	Rd, Rs	Rd = COUNTLEADINGONES(Rs)
CLZ	Rd, Rs	Rd = COUNTLEADINGZEROS(Rs)
<u>LA</u>	Rd, LABEL	Rd = ADDRESS(LABEL)
<u>LI</u>	Rd, IMM32	Rd = IMM32
LUI	Rd, CONST16	Rd = CONST16 << 16
<u>MOVE</u>	Rd, Rs	Rd = Rs
<u>NEGU</u>	Rd, Rs	Rd = −Rs
SEB ^{R2}	Rd, Rs	Rd = Rs _{7:0} [±]
SEH ^{R2}	Rd, Rs	Rd = Rs _{15:0} [±]
SUB	Rd, Rs, Rt	Rd = Rs − Rt (OVERFLOW TRAP)
SUBU	Rd, Rs, Rt	Rd = Rs − Rt

SHIFT AND ROTATE OPERATIONS		
ROTR ^{R2}	Rd, Rs, BITS5	Rd = Rs _{BITS5−1:0} :: Rs _{31:BITS5}
ROTRV ^{R2}	Rd, Rs, Rt	Rd = Rs _{RT40−1:0} :: Rs _{31:RT40}
SLL	Rd, Rs, SHIFT5	Rd = Rs << SHIFT5
SLLV	Rd, Rs, Rt	Rd = Rs << RT _{4:0}
SRA	Rd, Rs, SHIFT5	Rd = Rs [±] >> SHIFT5
SRAV	Rd, Rs, Rt	Rd = Rs [±] >> RT _{4:0}
SRL	Rd, Rs, SHIFT5	Rd = Rs [∅] >> SHIFT5
SRLV	Rd, Rs, Rt	Rd = Rs [∅] >> RT _{4:0}

Copyright © 2008 MIPS Technologies, Inc. All rights reserved.

LOGICAL AND BIT-FIELD OPERATIONS		
AND	Rd, Rs, Rt	Rd = Rs & Rt
ANDI	Rd, Rs, CONST16	Rd = Rs & CONST16 [∅]
EXT ^{R2}	Rd, Rs, P, S	Rs = Rs _{P+S−1:P} [∅]
INS ^{R2}	Rd, Rs, P, S	Rd _{P+S−1:P} = Rs _{S−1:0}
NOP		No-OP
NOR	Rd, Rs, Rt	Rd = ~(Rs Rt)
NOT	Rd, Rs	Rd = ~Rs
OR	Rd, Rs, Rt	Rd = Rs Rt
ORI	Rd, Rs, CONST16	Rd = Rs CONST16 [∅]
WSBH ^{R2}	Rd, Rs	Rd = Rs _{23:16} :: Rs _{31:24} :: Rs _{7:0} :: Rs _{15:8}
XOR	Rd, Rs, Rt	Rd = Rs ⊕ Rt
XORI	Rd, Rs, CONST16	Rd = Rs ⊕ CONST16 [∅]

CONDITION TESTING AND CONDITIONAL MOVE OPERATIONS		
MOVN	Rd, Rs, Rt	IF Rt ≠ 0, Rd = Rs
MOVZ	Rd, Rs, Rt	IF Rt = 0, Rd = Rs
SLT	Rd, Rs, Rt	Rd = (Rs [±] < Rt [±]) ? 1 : 0
SLTI	Rd, Rs, CONST16	Rd = (Rs [±] < CONST16 [±]) ? 1 : 0
SLTIU	Rd, Rs, CONST16	Rd = (Rs [∅] < CONST16 [∅]) ? 1 : 0
SLTU	Rd, Rs, Rt	Rd = (Rs [∅] < Rt [∅]) ? 1 : 0

MULTIPLY AND DIVIDE OPERATIONS		
DIV	Rs, Rt	Lo = Rs [±] / Rt [±] ; Hi = Rs [±] MOD Rt [±]
DIVU	Rs, Rt	Lo = Rs [∅] / Rt [∅] ; Hi = Rs [∅] MOD Rt [∅]
MADD	Rs, Rt	Acc += Rs [±] × Rt [±]
MADDU	Rs, Rt	Acc += Rs [∅] × Rt [∅]
MSUB	Rs, Rt	Acc -= Rs [±] × Rt [±]
MSUBU	Rs, Rt	Acc -= Rs [∅] × Rt [∅]
MUL	Rd, Rs, Rt	Rd = Rs [±] × Rt [±]
MULT	Rs, Rt	Acc = Rs [±] × Rt [±]
MULTU	Rs, Rt	Acc = Rs [∅] × Rt [∅]

ACCUMULATOR ACCESS OPERATIONS		
MFHI	Rd	Rd = Hi
MFLO	Rd	Rd = Lo
MTHI	Rs	Hi = Rs
MTLO	Rs	Lo = Rs

JUMPS AND BRANCHES (NOTE: ONE DELAY SLOT)		
B	OFF18	PC += OFF18 [±]
<u>BAL</u>	OFF18	RA = PC + 8, PC += OFF18 [±]
BEQ	Rs, Rt, OFF18	IF Rs = Rt, PC += OFF18 [±]
BEQZ	Rs, OFF18	IF Rs = 0, PC += OFF18 [±]
BGEZ	Rs, OFF18	IF Rs ≥ 0, PC += OFF18 [±]
BGEZAL	Rs, OFF18	RA = PC + 8; IF Rs ≥ 0, PC += OFF18 [±]
BGTZ	Rs, OFF18	IF Rs > 0, PC += OFF18 [±]
BLEZ	Rs, OFF18	IF Rs ≤ 0, PC += OFF18 [±]
BLTZ	Rs, OFF18	IF Rs < 0, PC += OFF18 [±]
BLTZAL	Rs, OFF18	RA = PC + 8; IF Rs < 0, PC += OFF18 [±]
BNE	Rs, Rt, OFF18	IF Rs ≠ Rt, PC += OFF18 [±]
<u>BNEZ</u>	Rs, OFF18	IF Rs ≠ 0, PC += OFF18 [±]
J	ADDR28	PC = PC _{31:28} :: ADDR28 [∅]
JAL	ADDR28	RA = PC + 8; PC = PC _{31:28} :: ADDR28 [∅]
JALR	Rd, Rs	Rd = PC + 8; PC = Rs
JR	Rs	PC = Rs

LOAD AND STORE OPERATIONS		
LB	Rd, OFF16(Rs)	Rd = MEM8(Rs + OFF16 [±]) [±]
LBU	Rd, OFF16(Rs)	Rd = MEM8(Rs + OFF16 [±]) [∅]
LH	Rd, OFF16(Rs)	Rd = MEM16(Rs + OFF16 [±]) [±]
LHU	Rd, OFF16(Rs)	Rd = MEM16(Rs + OFF16 [±]) [∅]
LW	Rd, OFF16(Rs)	Rd = MEM32(Rs + OFF16 [±])
LWL	Rd, OFF16(Rs)	Rd = LOADWORDLEFT(Rs + OFF16 [±])
LWR	Rd, OFF16(Rs)	Rd = LOADWORDRIGHT(Rs + OFF16 [±])
SB	Rs, OFF16(Rt)	MEM8(Rt + OFF16 [±]) = Rs _{7:0}
SH	Rs, OFF16(Rt)	MEM16(Rt + OFF16 [±]) = Rs _{15:0}
SW	Rs, OFF16(Rt)	MEM32(Rt + OFF16 [±]) = Rs
SWL	Rs, OFF16(Rt)	STOREWORDLEFT(Rt + OFF16 [±] , Rs)
SWR	Rs, OFF16(Rt)	STOREWORDRIGHT(Rt + OFF16 [±] , Rs)
<u>ULW</u>	Rd, OFF16(Rs)	Rd = UNALIGNED_MEM32(Rs + OFF16 [±])
<u>USW</u>	Rs, OFF16(Rt)	UNALIGNED_MEM32(Rt + OFF16 [±]) = Rs

ATOMIC READ-MODIFY-WRITE OPERATIONS		
LL	Rd, OFF16(Rs)	Rd = MEM32(Rs + OFF16 [±]); LINK
SC	Rd, OFF16(Rs)	IF ATOMIC, MEM32(Rs + OFF16 [±]) = Rd; Rd = ATOMIC ? 1 : 0

MD00565 Revision 01.01

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)	
<p>Stack Management</p> <ul style="list-style-type: none"> 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. 	
<p>Function Parameters</p> <ul style="list-style-type: none"> Every parameter smaller than 32 bits is promoted to 32 bits. First four parameters are passed in registers \$a0–\$a3. <ul style="list-style-type: none"> 64-bit parameters are passed in register pairs: <ul style="list-style-type: none"> Little-endian mode: \$a1:\$a0 or \$a3:\$a2. Big-endian mode: \$a0:\$a1 or \$a2:\$a3. Every subsequent parameter is passed through the stack. <ul style="list-style-type: none"> First 16 bytes on the stack are not used. Assuming \$sp was not modified at function entry: <ul style="list-style-type: none"> 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. 	
<p>Return Values</p> <ul style="list-style-type: none"> 32-bit and smaller values are returned in register \$v0. 64-bit values are returned in registers \$v0 and \$v1: <ul style="list-style-type: none"> 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
<pre> unsigned mips_cycle_counter_read() { unsigned cc; asm volatile("mfc0 %0, \$9" : "=r" (cc)); return (cc << 1); } </pre>

ASSEMBLY-LANGUAGE FUNCTION EXAMPLE
<pre> # 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 </pre>

C / ASSEMBLY-LANGUAGE FUNCTION INTERFACE
<pre> #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); } </pre>

INVOKING MULT AND MADD INSTRUCTIONS FROM C
<pre> 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); } </pre>

ATOMIC READ-MODIFY-WRITE EXAMPLE
<pre> atomic_inc: ll \$t0, 0(\$a0) # load linked addiu \$t1, \$t0, 1 # increment sc \$t1, 0(\$a0) # store cond'1 beqz \$t1, atomic_inc # loop if failed nop </pre>

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
<pre> typedef struct { int u; } __attribute__((packed)) unaligned; int unaligned_load(void *ptr) { unaligned *uptr = (unaligned *)ptr; return uptr->u; } </pre>

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
<ul style="list-style-type: none"> 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.