



Bài tập/Thực hành 3

CHƯƠNG 2 KIẾN TRÚC TẬP LỆNH MIPS: CÁC LỆNH ĐIỀU KIỆN

Mục tiêu

- Chuyển từ ngôn ngữ cấp cao (C) sang hợp ngữ MIPS.
- Sử dụng lệnh điều khiển (nhảy, rẽ nhánh) để điều khiển luồng chương trình.

Yêu cầu

- Xem cách dùng các lệnh (set, branch, jump, load, store) trong slide và trong file tham khảo [trang 4].
- Nộp các file code hợp ngữ đặt tên theo format [Bai*.asm] chứa trong thư mục Lab3_MSSV

Kiểu lệnh

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.

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

Bài tập và Thực hành

Lập trình có cấu trúc.

Sinh viên chuyển các cấu trúc sau của ngôn ngữ C qua ngôn ngữ assembly. Tham khảo hình ảnh về các cấu trúc ở cuối bài thực hành.

Bài 1: Phát biểu IF-ELSE (1)

```
1  if( a % 2 == 0) { Print string: "Computer Science and Engineering, HCMUT"}
2  else           { Print string: "Computer Architecture 2019"}
```

Bài 2: Phát biểu IF-ELSE (2)

```
1  if( a >= -3 && a <= 4) { a = b - c; }
2  else                  { a = b + c; }
```

Bài 3: Phát biểu SWITCH-CASE

Hiện thực phát biểu switch-case bên dưới bằng hợp ngữ. Cho biết $b = 100$, $c = 2$. Giá trị input nhập từ người dùng. **Xuất ra giá trị của a.**

```
1  switch (input)
2  {
3      case 1: a = b + c; break;
4      case 2: a = b - c; break;
5      case 3: a = b x c; break;
6      case 4: a = b / c; break;
7      default: NOP; // No-Operation; a = 0
8              break;
9  }
```

Bài 4: Vòng lặp FOR - xác định chuỗi Fibonacci bằng vòng lặp. Nhập vào n (nguyên dương), xuất ra số Fibonacci Fn.

```
1  if      (n == 0) {return 0;}
2  else if (n == 1) {return 1;}
3  else{
4      f0= 0; f1 = 1;
5      for ( i = 2; i <= n; i++) {
6          fn = fn-1 + fn-2;
7      }
8  }
9  return fn;
```

Note: sinh viên có thể là theo cách riêng để tìm ra số Fibonacci Fn.

Dãy số Fibonacci http://en.wikipedia.org/wiki/Fibonacci_number

F ₀	F ₁	F ₂	F ₃	F ₄	F ₅	F ₆	F ₇	F ₈	F ₉
0	1	1	2	3	5	8	13	21	34
F ₁₀	F ₁₁	F ₁₂	F ₁₃	F ₁₄	F ₁₅	F ₁₆	F ₁₇	F ₁₈	F ₁₉
55	89	144	233	377	610	987	1597	2584	4181

Bài 5: Vòng lặp WHILE

Xác định vị trí chữ 'u' đầu tiên trong chuỗi "Computer Architecture CSE-HCMUT".

```
1 i = 0;
2 while( charArray[i] != 'u' && charArray[i] != '\0' ) {
3     i++;
4 }
```

Xuất ra giá trị index của ký tự 'u'. Nếu không tìm thấy thì xuất ra -1.

Làm thêm

1. ENDIANESS.

Cho mảng số nguyên bên dưới.

```
1 .data
2 intArray: .word 0xCA002019, 0xC0002009
3 .text
4     la $a0, intArray
5     lb $t0, 0($a0)
6     lb $t1, 1($a0)
7     lb $t2, 2($a0)
8     lb $t3, 3($a0)
9     lbu $t4, 0($a0)
10    lbu $t5, 1($a0)
11    lbu $t6, 2($a0)
12    lbu $t7, 3($a0)
```

- (a) Giả sử MIPS được thiết kế theo kiểu BIG ENDIAN, xác định giá trị các ô nhớ (theo byte) của mảng trên.
- (b) Giả sử MIPS được thiết kế theo kiểu LITTLE ENDIAN, xác định giá trị các ô nhớ (theo byte) của mảng trên.
- (c) Xác định giá trị các thanh ghi \$t của đoạn code bên dưới, giả sử MIPS được thiết kế theo kiểu BIG ENDIAN.
- (d) Xác định giá trị các thanh ghi \$t của đoạn code bên dưới, giả sử MIPS được thiết kế theo kiểu LITTLE ENDIAN.

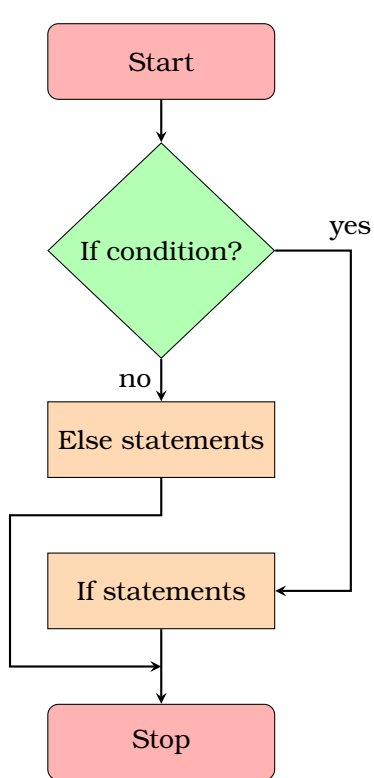
2. Memory alignment.

Cho đoạn code mips bên dưới

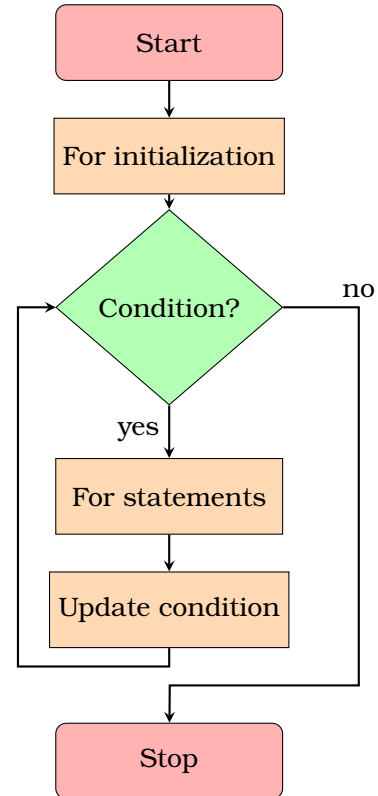
```
1 .data
2 int_1: .word 0xCA002018
3 char_1: .byte 0xFF
4 int_2: .word 2018
5 char_2: .byte 0xCA 0xFE 0xED
6 .text
7     la $a0, int_1
8     lw $t0, 0($a0)
9     lw $t1, 1($a0)
10    lh $t2, 2($a0)
11    lh $t3, 3($a0)
12    lb $t4, 0($a0)
13    lb $t5, 1($a0)
```

- (a) Xác định nội dung của vùng nhớ dữ liệu và xác định các lệnh sẽ gây ra lỗi khi thực thi, giải thích. Biết MIPS chuẩn được thiết kế theo kiểu BIG ENDIAN.
- (b) Xếp lại dữ liệu sao cho bộ nhớ tối ưu hơn (trong kiến trúc 32 bit).

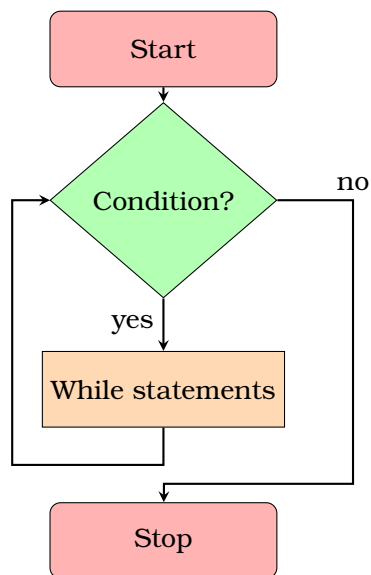
Sơ đồ cấu trúc của phát biểu (if-else, for, while, do-while)



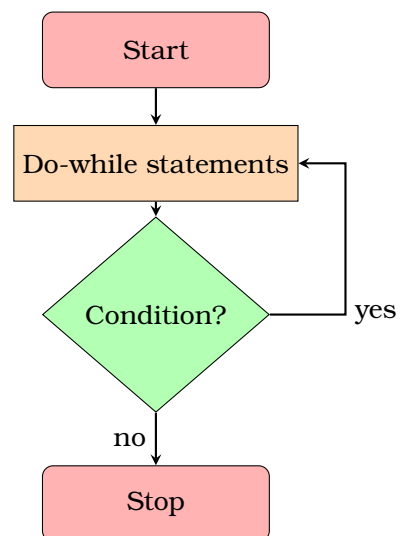
Hình. 1: If-else statement



Hình. 2: For statements



Hình. 3: While statement



Hình. 4: Do-while statement

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 (ACC31:0) AND HIGH (ACC63:32) 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)	
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
<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
- 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.