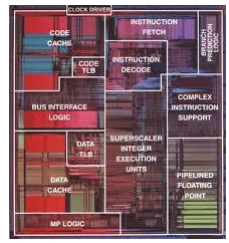
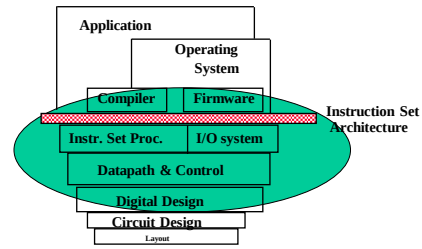
CS/SE 3340
Computer Architecture





Introduction to MIPS Assembly Language

Adapted from "Computer Organization and Design, 4th Ed." by D. Patterson and J. Hennessy

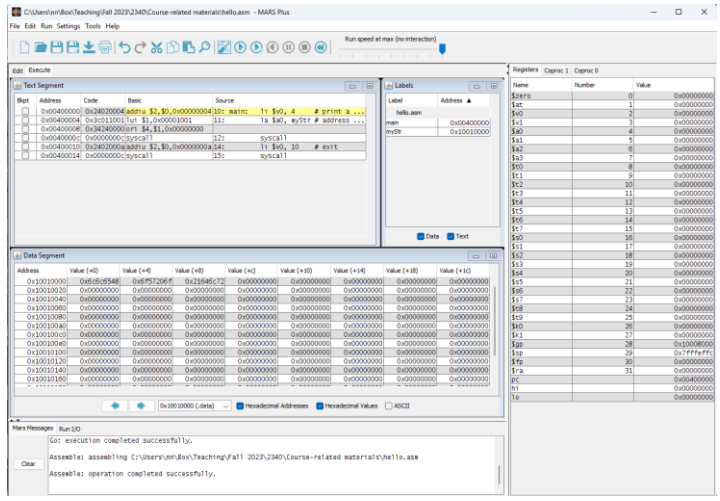
1

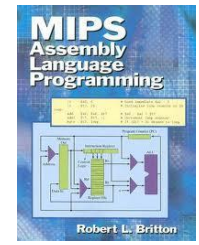
Questions

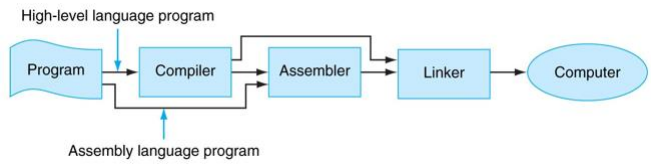
- What are the abstraction levels of a computer?
- How to 'instruct' the processor to process information
- What is the process to translate a HLL program to an executable and run it on a computer?
- How to write a MIPS assembly language statement?
- How to use MARS to develop MIPS assembly language programs?

2

2



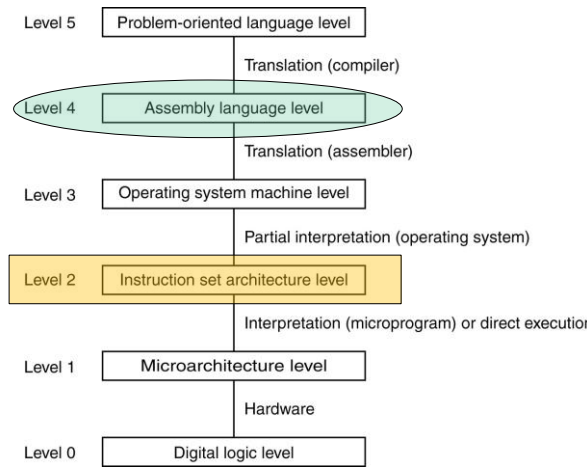




3

3

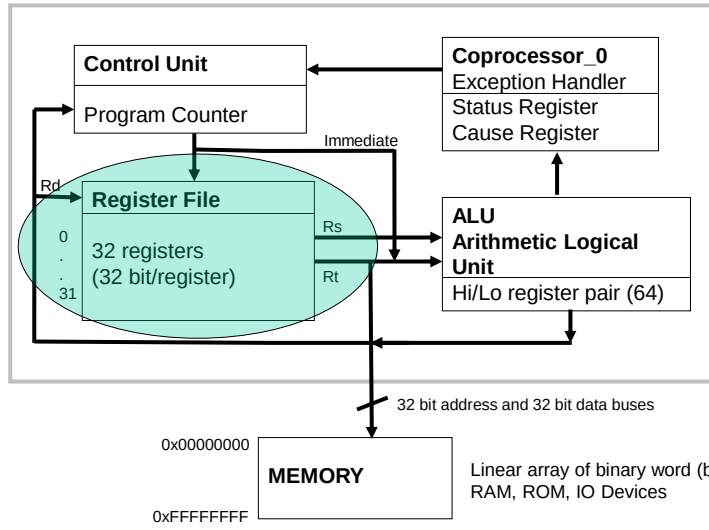
Computer Abstraction Levels



4

4

MIPS CPU Functional Model



5

5

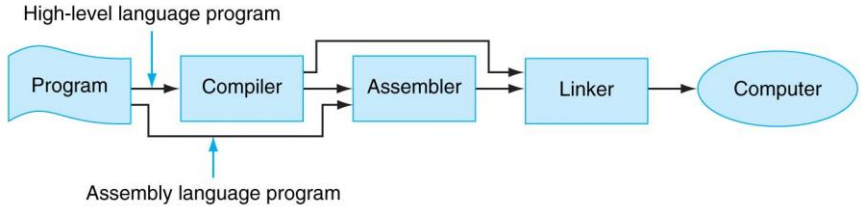
MIPS Registers

Register	Name	Usage
0	$zero	constant 0
1	$at	Reserved for assembler (pseudo-instructions)
2-3	$v0,$v1	Return function values
4-7	$a0-$a3	Function arguments
8-15 and 24-25	$t0-$t7, $t8,$t9	Temporaries (not preserved across call)
16-23	$s0-$s7	Save registers (preserved across call)
26-27	$k0,$k1	Reserved for kernel/OS
28	$gp	Pointer to global data area
29	$sp	Stack pointer. MARS initializes to 0x7FFF FFFC
30	$fp	Frame pointer
31	$ra	Return address, used by “link” instruction (HW)

6

6

From HLL to Executable

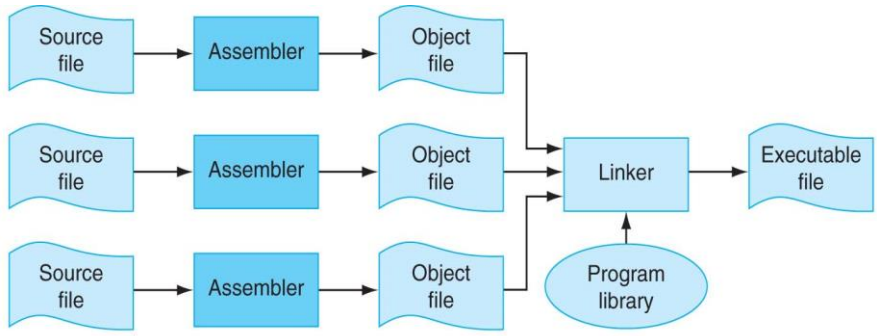



```
#include <stdio.h>

int
main (int argc, char *argv[])
{
    int i;
    int sum = 0;

    for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
    printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

7

7

Assembler



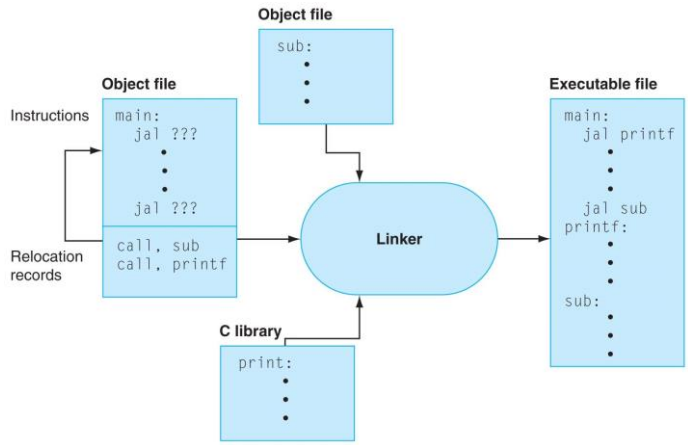
Why multiple object files?

8

8

Object Files and Linker

Object file format on Unix



9

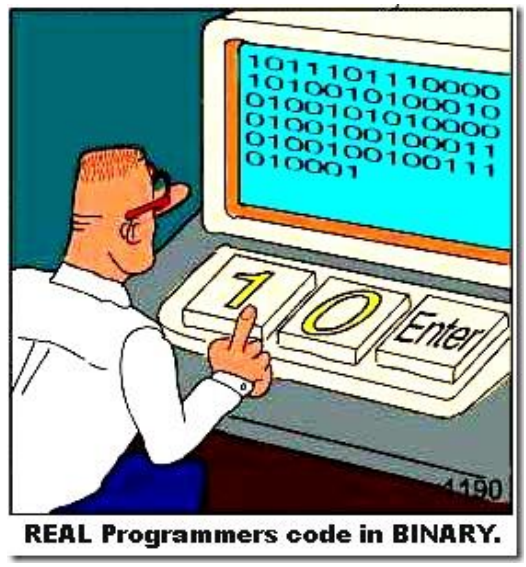
9

Object Code

```
00100111101111011111111111100000    addiu   $29, $29, -32
10101111101111110000000000010100    sw      $31, 20($29)
10101111101001000000000000100000    sw      $4,  32($29)
10101111101001010000000000100100    sw      $5,  36($29)
10101111101000000000000000011000    sw      $0,  24($29)
10101111101000000000000000011100    sw      $0,  28($29)
10001111101011100000000000011100    lw      $14, 28($29)
10001111101111000000000000011000    lw      $24, 24($29)
00000001110011100000000000011001    multu   $14, $14
00100101110010000000000000000001    addiu   $8,  $14, 1
00101001000000010000000001100101    slti    $1,  $8, 101
10101111101010000000000000011100    sw      $8,  28($29)
00000000000000000111100000010010    mflo    $15
00000011000011111100100000100001    addu    $25, $24, $15
00010100001000001111111111110111    bne     $1,  $0, -9
10101111101110010000000000011000    sw      $25, 24($29)
00111100000001000001000000000000    lui     $4,  4096
10001111101001010000000000011000    lw      $5,  24($29)
00001100000100000000000011101100    jal     1048812
00100100100001000000010000110000    addiu   $4,  $4, 1072
10001111101111110000000000010100    lw      $31, 20($29)
00100111101111010000000000100000    addiu   $29, $29, 32
00000011111000000000000000001000    jr      $31
00000000000000000001000000100001    move    $2,  $0
```

10

10



11

11

Assembly Language

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra

        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

12

12

Assembly Language Instructions

- Two types: *native* and *pseudo*
  - Does not make any differences from programming point of view
- Native instructions
  - Directly understood by machine, i.e. one-to-one encoding to machine code
  - Example: `add Rd, Rs, Rt`
- Pseudo instructions
  - Sugar-coated for programmers
  - May be consisted of one or more native instructions
  - Example: `move Rd, Rs`

13

13

Assembly Instruction Format

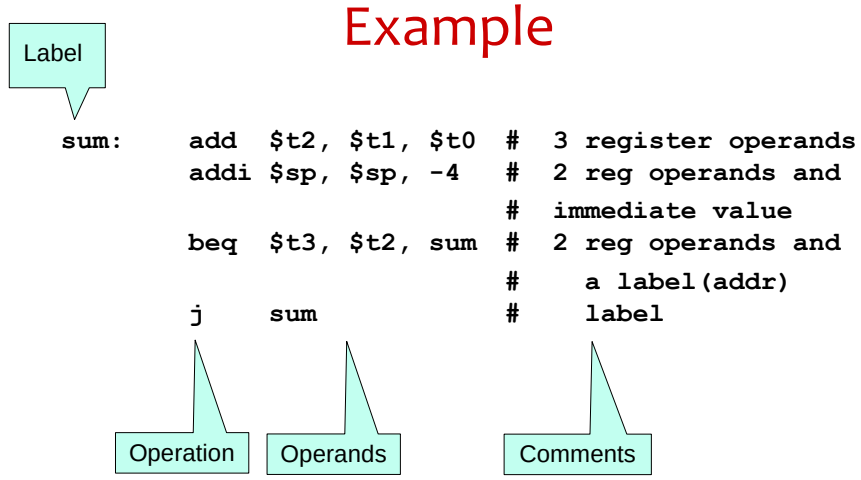
[label:] operation [operand1 [, operand2 [, operand3]]] [# [comment]]

1. Labels: A symbol string associated with a specific memory address
2. Operations:
   a) Assembler directive
   b) Machine instruction
3. Operands:
   a. Register names (i.e. $0, $29, named: $a0, 0($t0)),
   b. Immediate value Numeric expression
   c. Address label (instruction or data, i.e. Loop2:, myVal:)
4. Comments: Text string from # symbol to end of line. Ignored by assembler.

14

14

Assembly Instruction Format – Example



15

15

Assembler Directives

- Instructions understood by the assembler, not by the CPU
  - Start with a '.'
  - Executed by assembler at assembly time, not at run-time
- Directives for allocating data items
  - **e.g. .word,.half,.byte,.asciiz,** …
- Directives for segments information
  - **e.g. .data,.text**
- Symbol related directives
  - **e.g. .globl**

16

16

Useful Assembler Directives

`.data` Subsequent data items stored in user(kernel) data segment(.kdata)

`.text` Subsequent items are stored in user(kernel) text segment (.ktext)

`.asciiz str` Store ascii string in memory and '\0' terminate

`.word w1,…` Store 32 bit words in memory

`.half h1,…` Store 16 bit half-words in memory

`.byte b1,…` Store 8 bit bytes in memory

`.double d1,…` Store 64 bit words in memory

`.space nbytes` Allocate nbytes of space in current segment

`.globl sym` Declare sym global. Can be referenced from other object files

`.align n` Align next datum on a 2^n boundary

17

17

MIPS Assembly Language Syntax

- Numbers are base **10**
- Hex numbers are preceded **"0x"**
- Special string characters:
  a) newline \n
  b) tab \t
  c) quote \"
- Labels are followed by ":"
- Identifiers begin with letter and may contain alphanumeric, underscore, and dots

  Note: keywords and instruction opcodes can not be used as identifiers
- Comments begin with a "#" symbols and run to end-of-line
- Assembly language statements cannot be split across multiple lines

18

18

MIPS Assembly Program Example

```
 # Program name, description  and comments
       .data              # data segment
item:  .word    10        # define/name a variable
                          # and initial value
       .text              #  code segment
       .globl  main       # symbol main is global;
main:                     # your code goes here
       lw       $t0, item
       …
       …
       li       $v0, 10   # exit to kernel
       syscall            #  system call (OS)
       .end
```

19

19

Operating System Services – MARS

- To request OS services, e.g., input and output, the **`syscall`** pseudo-instruction can be used
- Usage convention
  - Put requested service (encoded as a number) to register $v0
  - Put input value in register $a0 (or $f12 for floating point numbers)
  - Get output result from register $v0 (or $f0)
- Example:

```
li     $v0, 4       # load request to print
la     $a0, hello   # load address of string
syscall
```

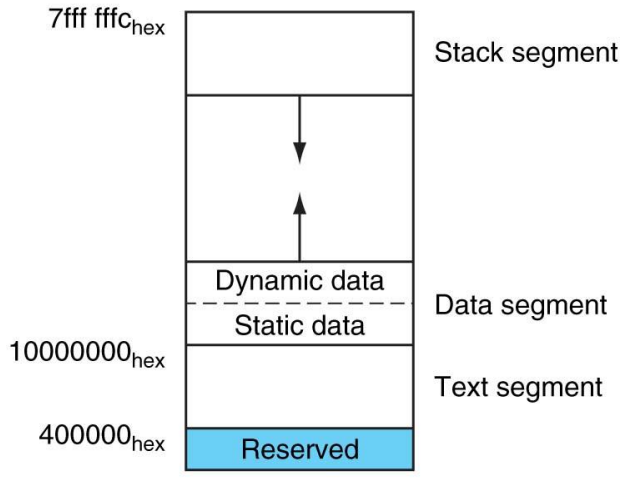
20

20

Syscall Services

Service	Code	Arguments	Result
print integer	**1**	***$a0 = value***	***(none)***
print float	2	$f12 = float value	(none)
print double	3	$f12 = double value	(none)
print str	**4**	***$a0 = address of str***	***(none)***
read integer	**5**	***(none)***	***$v0 = value read***
read float	6	(none)	$f0 = value read
read double	7	(none)	$f0 = value read
read str	**8**	***$a0 = address of str***	***(none)***
		$a1 = number of chars	
memory allocation	9	$a0 = bytes of storage desired	
		$v0 = address of block	
exit (end of program)	10	(none)	(none)

MIPS Main Memory Usage



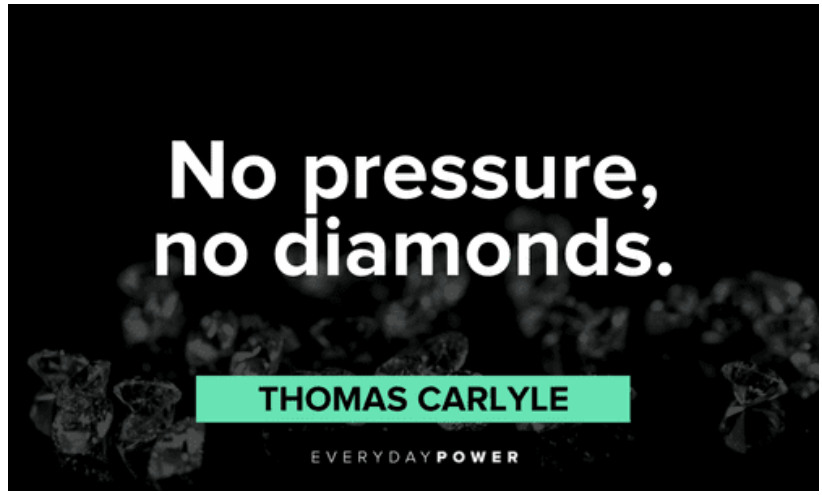
MIPS Main Memory Usage

Address	Directive	Memory Usage
0x0000 0000 -0x003F FFFF	.vect	Reserved by kernel
0x0040 0000 -0x1000 0000	.text	Code segment
0x1000 0000 -0x1001 0000	.data	Static data
0x1001 0000 -		Dynamic data
0x xxxx xxxx -0x7FFF FFFC	.stack	Heap <=> Stack
0x8000 0180 -0x9000 0000	.ktext	Reserved, kernel code
0x9000 0000 -0x9001 0000	.kdata	Reserved, kernel data

MARS:

$gp = 0x1000 8000	Global pointer, points to global data area
$sp =0x7FFF FFFC	User stack pointer

Quote of the Day



Appendix

MIPS Assembly Language Instructions

MIPS Core Instructions	Operation	Operands	Size/Clock Cycles
Add:	add	Rd, Rs, Rt	1/1
Add Immediate:	addi	Rt, Rs, Imm	1/1
Add Immediate Unsigned:	addiu	Rt, Rs, Imm	1/1
Add Unsigned:	addu	Rd, Rs, Rt	1/1
And:	and	Rd, Rs, Rt	1/1
And Immediate:	andi	Rt, Rs, Imm	1/1
Branch if Equal:	beq	Rs, Rt, Label	1/1
Branch if Not Equal:	bne	Rs, Rt, Label	1/1
Jump:	j	Label	1/1
Jump and Link:	jal	Label	1/1
Jump Register:	jr	Rs	1/1
Load Byte:	lb	Rt, offset(Rs)	1/1
Load Byte Unsigned:	lbu	Rt, offset(Rs)	1/1
Load Upper Immediate:	lui	Rt, Imm	1/1
Load Word:	lw	Rt, offset(Rs)	1/1
Or:	or	Rd, Rs, Rt	1/1
Or Immediate:	ori	Rt, Rs, Imm	1/1
Set on Less Than:	slt	Rd, Rt, Rs	1/1
Set on Less Than Immediate:	slti	Rt, Rs, Imm	1/1
Set on Less Than Immediate Unsigned:	sltiu	Rt, Rs, Imm	1/1
Set on Less Than Unsigned:	sltu	Rd, Rt, Rs	1/1
Shift Left Logical:	sll	Rd, Rt, sa	1/1
Shift Right Logical:	srl	Rd, Rt, sa	1/1
Subtract:	sub	Rd, Rs, Rt	1/1
Subtract Unsigned:	subu	Rd, Rs, Rt	1/1
Store Byte:	sb	Rt, offset(Rs)	1/1
Store Word:	sw	Rt, offset(Rs)	1/1

MIPS Arithmetic Core Instructions	Operation	Operands	Size/Clock Cycles
Divide:	div	Rs, Rt	1/38
Divide Unsigned:	divu	Rs, Rt	1/38
Move From High:	mfhi	Rd	1/1
Move From Low:	mflo	Rd	1/1
Multiply:	mult	Rs, Rt	1/32
Multiply Unsigned:	multu	Rs, Rt	1/32

MIPS Assembly Language Instructions – cont'd

MIPS Instructions (remaining)	Operations	Operands	Size/Clock Cycles
Branch if Greater Than or Equal to Zero:	bgez	Rs, Label	1/1
Branch if Greater Than or Equal to Zero and Link:	bgezal	Rs, Label	1/1
Branch if Greater Than Zero:	bgtz	Rs, Label	1/1
Branch if Less Than or Equal to Zero:	blez	Rs, Label	1/1
Branch if Less Than Zero and Link:	bltzal	Rs, Label	1/1
Branch if Less Than Zero:	bltz	Rs, Label	1/1
Cause Exception:	break		1/1
Exclusive Or:	xor	Rd, Rs, Rt	1/1
Exclusive Or Immediate:	xori	Rt, Rs, Imm	1/1
Jump and Link Register:	jalr	Rd, Rs	1/1
Load Halfword:	lh	Rt, offset(Rs)	1/1
Load Halfword Unsigned:	lhu	Rt, offset(Rs)	1/1
Load Word Left:	lwl	Rt, offset(Rs)	1/1
Load Word Right:	lwr	Rt, offset(Rs)	1/1
Move to High:	mthi	Rs	1/1
Move to Low:	mtlo	Rs	1/1
Nor:	nor	Rd, Rs, Rt	1/1
Return from Exception	rfe		1/1
Shift Left Logical Variable:	sllv	Rd, Rt, Rs	1/1
Shift Right Arithmetic:	sra	Rd, Rt, sa	1/1
Shift Right Arithmetic Variable:	srav	Rd, Rt, Rs	1/1
Shift Right Logical Variable:	srlv	Rd, Rt, Rs	1/1
Store Halfword:	sh	Rt, offset(Rs)	1/1
Store Word Left:	swl	Rt, offset(Rs)	1/1
Store Word Right:	swr	Rt, offset(Rs)	1/1
System Call:	syscall		1/1

Operands

1) Register names *Rd, Rs, Rt* (d=destination, s=source, t=second source/dest)
2) Immediate value Imm, sa, offset (Numeric expr= 16 bits,shift amount, offset)
3) Address label Label (28 or 16 bits)

27

27

MIPS Assembly Language Instructions – cont'd

Pseudo Instructions	Operations	Operands	Size/Clock Cycles
Absolute Value:	abs	Rd, Rs	3/3
Branch if Equal to Zero:	beqz	Rs, Label	1/1
Branch if Greater Than or Equal :	bge	Rs, Rt, Label	2/2
Branch if Greater Than or Equal Unsigned:	bgeu	Rs, Rt, Label	2/2
Branch if Greater Than:	bgt	Rs, Rt, Label	2/2
Branch if Greater Than Unsigned:	bgtu	Rs, Rt, Label	2/2
Branch if Less Than or Equal:	ble	Rs, Rt, Label	2/2
Branch if Less Than or Equal Unsigned:	bleu	Rs, Rt, Label	2/2
Branch if Less Than:	blt	Rs, Rt, Label	2/2
Branch if Less Than Unsigned:	bltu	Rs, Rt, Label	2/2
Branch if Not Equal to Zero:	bnez	Rs, Label	1/1
Branch Unconditional:	b	Label	1/1
Divide:	div	Rd, Rs, Rt	4/41
Divide Unsigned:	divu	Rd, Rs, Rt	4/41
Load Address:	la	Rd, Label	2/2
Load Double:	ld	Rd, Label	2/2
Load Immediate:	li	Rd, value	2/2
Move:	move	Rd, Rs	1/1
Multiply:	mul	Rd, Rs, Rt	1/33
Multiply (with overflow exception):	mulo	Rd, Rs, Rt	7/37
Multiply Unsigned (with overflow exception):	mulou	Rd, Rs, Rt	5/35

28

28

MIPS Assembly Language Instructions – cont'd

Pseudo Instructions	Operations	Operands	Size/Clock Cycles
Negate:	neg	Rd, Rs	1/1
Negate Unsigned:	negu	Rd, Rs	1/1
Not:	not	Rd, Rs	1/1
Nop:	nop		1/1
Remainder:	rem	Rd, Rs, Rt	4/41
Remainder Unsigned:	remu	Rd, Rs, Rt	4/41
Rotate Left:	rol	Rd, Rs, sa	3/3
Rotate Right	ror	Rd, Rs, sa	3/3
Rotate Left, variable:	rol	Rd, Rs, Rt	4/4
Rotate Right, variable	ror	Rd, Rs,Rt	4/4
Set on Equal:	seq	Rd, Rt, Rs	4/4
Set on Not Equal:	sne	Rd, Rt, Rs	4/4
Set on Greater Than:	sgt	Rd, Rt, Rs	1/1
Set on Greater Than Unsigned:	sgtu	Rd, Rt, Rs	1/1
Set on Greater Than or Equal:	sge	Rd, Rt, Rs	4/4
Set on Greater Than or Equal Unsigned:	sgeu	Rd, Rt, Rs	4/4
Set on Less Than or Equal:	slte	Rd, Rt, Rs	4/4
Set on Less Than or Equal Unsigned:	slteu	Rd, Rt, Rs	4/4
Store Double:	sd	Rd, Label	2/2
Unaligned Load Half Word:	ulh	Rd, Label	4/4
Unaligned Load Half Word Unsigned:	ulhu	Rd, Label	4/4
Unaligned Load Word:	ulw	Rd, Label	2/2
Unaligned Store Half Word:	ush	Rd, Label	3/3
Unaligned Store Word:	usw	Rd, Label	2/2

29