### CSEE 3827: Fundamentals of Computer Systems, Spring 2022

Lecture 8

Prof. Dan Rubenstein (danr@cs.columbia.edu)

#### Assembly Code vs. Machine Code

- An instruction has two forms: Assembly and Machine
  - Assembly: human-readable form, e.g., an instruction: add \$t1, \$s0, \$s2
    - says take values in registers s0 and s2, add them together, store result in register t1
  - Machine: bits that actually store the instruction that feed into the various MUXs, decoders, selector bits to produce the desired computation and/or operation:
    - instruction add \$t1, \$s0, \$s2 is 00000010 00110010 01000000 00100000
       in binary or 02 32 40 28 in hex
- An assembler is software that converts a text file of assembly code (instructions) into a binary file of machine code
  - Mostly a very straightforward (trivial) process: each instruction converts quite easily
  - One "smart" thing assembler does is permit labels for branches and jumps

#### Assembly Instruction → Machine Code Instruction

- Each assembly instruction (e.g., add \$t1, \$s0, \$s2)
- Has a corresponding 32-bit machine code representation,
   (e.g., 00000010 00110010 01000000 00100000 (binary) or 02 32 40 28 (hex))
- How is this 1-1 mapping performed?
  - Some parts are easy:
    - e.g., Each register is numbered (between 0 and 31), hence can be described with 5 bits
    - The operation (e.g., add) also easily described by a few bits
    - Constants (e.g., the -10 in adde \$t1, \$s0, -10) described using 16 bits or less (e.g., the 7 in sll \$t1, \$s0, 7 requires 5 bits)

# Addressing in an Instruction (for branches and jumps)

	£ 1			
	fact	:		
40000		addi	\$sp,	\$sp, -8
40004		sw	\$ra,	4(\$sp)
40008		sw	\$a0,	0(\$sp)
40012		slti	\$t0,	\$a0, 1
40016		beq	\$t0,	\$zero, L1
40020		addi	\$v0,	\$zero, 1
40024		addi	\$sp,	\$sp, 8
40028		jr	\$ra	
40032	L1:	addi	\$a0,	\$a0, -1
40036		jal	fact	
40040		lw	\$a0,	0(\$sp)
40044		lw	\$ra,	4(\$sp)
40048		addi	\$sp,	\$sp, 8
40052		mul	\$v0,	\$a0, \$v0
40056		jr	\$ra	

- 2 ways to interpret label:
  - Indirect addressing:
    - info in instruction explains how how far from current address
    - e.g., "From current location, go 10 miles East and 3 miles North"
  - Direct addressing:
    - Specify exact address to go to
    - e.g., "Go to 500 W 120th Street, New York NY 10027 USA"

	fact	:		
40000		addi	\$sp,	\$sp, -8
40004		sw	\$ra,	4(\$sp)
40008		sw	\$a0,	0(\$sp)
40012		slti	\$t0,	\$a0, 1
40016		beq	\$t0,	\$zero, L1
40020		addi	\$v0,	\$zero, 1
40024		addi	\$sp,	\$sp, 8
40028		jr	\$ra	
40028 40032	L1:	jr	\$ra	\$a0, -1
	L1:	jr	<pre>\$ra \$a0,</pre>	_
40032	L1:	jr addi	\$ra \$a0, fact	_
40032 40036	L1:	jr addi jal	\$ra \$a0, fact \$a0,	\$a0, -1
40032 40036 40040	L1:	jr addi jal lw lw	\$ra \$a0, fact \$a0, \$ra,	\$a0, -1 0(\$sp)
40032 40036 40040 40044	L1:	jr addi jal lw lw	\$ra \$a0, fact \$a0, \$ra, \$sp,	\$a0, -1 0(\$sp) 4(\$sp)
40032 40036 40040 40044 40048	L1:	jr addi jal lw lw addi	\$ra \$a0, fact \$a0, \$ra, \$sp,	\$a0, -1 0(\$sp) 4(\$sp) \$sp, 8

- Branches use Indirect addressing:
  - e.g., for beq instruction in fact, "If conditional true, skip 3 instructions" from where we would have otherwise been.
  - Uses a 16-bit (signed) constant to indicate # instructions to skip

	٠.			
	fact	:		
40000		addi	\$sp,	\$sp, -8
40004		sw	\$ra,	4(\$sp)
40008		sw 🚄	\$a0,	0(\$sp)
40012		slti	\$t0,	\$a0, 1
40016		beq	\$t0,	\$zero, fact
40020		addi	\$v0,	\$zero, 1
40024		addi	\$sp,	\$sp, 8
40028		jr	\$ra	
40032	L1:	addi	\$a0,	\$a0, -1
40036		jal	fact	
40040		lw	\$a0,	0(\$sp)
40044		lw	\$ra,	4(\$sp)
40048		addi	\$sp,	\$sp, 8
40052		mul	\$v0,	\$a0, \$v0
40056		jr	\$ra	

- Branches use Indirect addressing:
  - e.g., for beq instruction in fact, "If conditional true, skip 3 instructions" from where we would have otherwise been.
  - Uses a 16-bit (signed) constant to indicate # instructions to skip
  - Can branch backwards as well
    - e.g., here it says "If conditional true, skip -5 instructions"

		_		
	fact	:		
40000		addi	\$sp,	\$sp, -8
40004		sw	\$ra,	4(\$sp)
40008		sw	\$a0,	0(\$sp)
40012		slti	\$t0,	\$a0, 1
40016		beq	\$t0,	\$zero, L1
40020		addi	\$v0,	\$zero, 1
40024		addi	\$sp,	\$sp, 8
40028		jr	\$ra	
40032	L1:	addi	\$a0,	\$a0, -1
40036		jal	fact	
40040		lw	\$a0,	0(\$sp)
40044		lw	\$ra,	4(\$sp)
40048		addi	\$sp,	\$sp, 8
40052		mul	\$v0,	\$a0, \$v0
40056		jr	\$ra	

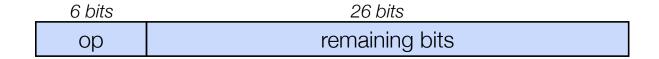
- Branches use Indirect addressing:
  - e.g., for beq instruction in fact, "If conditional true, skip 3 instructions" from where we would have otherwise been.
  - Uses a 16-bit (signed) constant to indicate # instructions to skip
  - · Can branch backwards as well
    - e.g., here it says "If conditional true, skip -5 instructions"

- Jumps use Direct addressing:
  - Requires that assembler know where (i.e., 32-bit addresses) in memory of code to be known by assembler
  - e.g., for jal instruction: "The next instruction to execute will be at address 40,000.

#### Instruction Formats

#### Representing Instructions

General form of machine code instruction:



- op is the 6-bit op-code: indicates how remaining 26 bits will be interpreted
- There are 3 basic formats
  - R-format: instructions with 3 register parameters (e.g., add \$s1, \$s2, \$s3)
    - Shifting (sll, slr) are also R-format even though only have 2 params
  - I-format / mem-access / (indirect addressing) conditional branches
  - (direct addressing) jumps

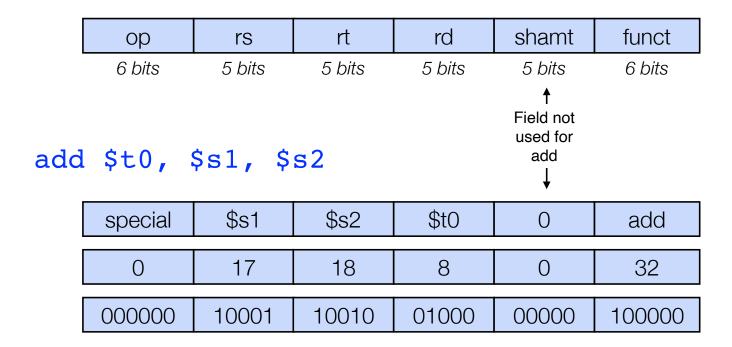
#### Representing Instructions

#### R-format (R-type): instructions that have 3 register parameters

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
op=000000	rs	rt	rd	shamt	funct

- e.g., add \$t0, \$s1, \$s2: must indicate 3 registers, "+" operation
- op-code always 000000 for R-format instruction
- **rs** = 5-bit description of 1st register to be read from
- rt = 5-bit description of 2nd register to be read from
  - rs = rt permitted
- rd = 5-bit description of register to write to (desination reg)
- shamt: used only for shift ops, 5-bit specification for how much to shift by
- funct: the specific operation (e.g., add, sub, or, sll, etc. to perform)
  - e.g., add is 100000, addu is 100001, subtract is 100010, etc.

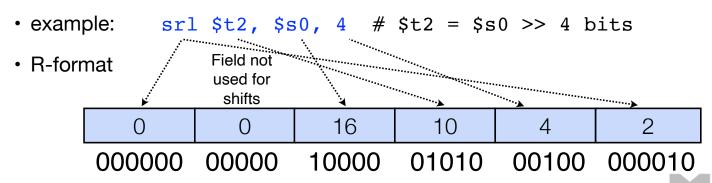
#### R-format Example: add





#### R-Format Example 2: shift right logical (srl)

- e.g., shift left logical (i.e., instruction is = s11)
  - Shift left and fill with 0s
  - sll by *i* bits multiplies by 2 (unsigned only)
- Shift right logical (op = srl)
  - Shift right and fill with 0s
  - srl by i bits divides by  $2^{i}$  (for unsigned values only)
- shamt is 5-bit description of how far to shift by



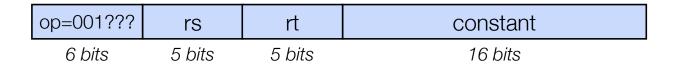
#### 2-register Instructions

#### I-format / mem-access / (indirect addressing) conditional branches

6 bits	5 bits	5 bits	16 bits
op	rs	rt	constant

- op-code describes what the instruction will do (no funct field here)
  - op = 1?????: memory access op (e.g., lw, sw)
  - op = 001???: I-format: immediate (uses constant) arithmetic/logic op (e.g., addi, ori, etc.
  - op = 0001?? or 000001: conditional branch
- rs = 5-bit description of a register to read from
- rt = 5-bit description of 2nd register (some instructions write to, some read to)
- constant: 16-bit constant (usually unsigned)

#### MIPS I-format Instructions



- Includes immediate arithmetic and load/store operations
  - op: operation code (opcode): addi [001000], addiu [001001], etc.
  - rs: source register number (register to read from)
  - rt: destination register number (register to write to)
  - constant: offset added to base value in rs, its interpretation depends on opcode (i.e., signed or unsigned)



#### I-format Example: add

op=001???	rs	rt	constant
6 bits	5 bits	5 bits	16 bits

#### addi \$t0, \$s1, -1

addi	\$s1	\$tO	-1
8	17	8	-1
001000	10001	01000	111111111111111



#### Memory Access

	base addr register	register w/ data	
op=1?????	rs	rt	constant
6 bits	5 bits	5 bits	16 bits

- e.g., lw [100011], sw [101011]
- Let A = value in register #rs, B = value in register #rt, C = value of constant
- lw: loads word in memory at address A+C into register #rt
- sw: stores B into memory at address A+C



#### Iw Example

op=100011	rs	rt	constant
6 bits	5 bits	5 bits	16 bits



lw	(\$s1)	\$tO	8
35	17	8	000000000001000
100011	10001	01000	000000000001000



#### sw Example

op=101011	rs	rt	constant
6 bits	5 bits	5 bits	16 bits

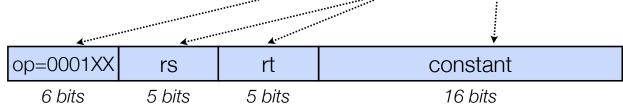
#### sw \$t0, 8(\$s1)

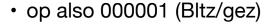
SW	\$s1	\$t0	8
43	17	8	000000000001000
101011	10001	01000	000000000001000



#### **Branch Addressing**

Branch instructions specify: opcode, two registers, branch target







- Uses both registers: beq [000100], bne [000101]
- Uses just rs: blez [000110], bgtz [000111]
- Most branch targets are near branch (either forwards or backwards)
- Recall branches use relative addressing: instruction's constant specifies offset from next instruction (current address + 4)
  - target address = next address + (offset \* 4) (NOTE constant counts instructions (words), not bytes - hence the need to \* 4)
  - Q: Why is offset computed from next (+4) address instead of current?

#### Target Addressing Example

- Loop code from earlier example
- Assume Loop label placed in memory at address 80000

#### Loop: sll \$t1, \$s3, 2 add \$t1, \$t1, \$s5 lw \$t0, 0(\$t1) bne \$t0, \$s4, Exit addi \$s3, \$s3, 1 j Loop Exit:

#### Machine code

80000	0	0	19	9	4	0
80004	0	9	21	9	0	32
80008	35	9	8		0	
80012	5	8	20		2	<b>→</b>
80016	8	19	19		1	
80020	2		20	00	0	·
80024						

Note: branch constant measures distance in words (i.e., instructions), not bytes (whereas memory access is in bytes, not words)



#### Jump Addressing

 Jump (j and jal) targets could be anywhere in a text segment, so, encode the full address in the instruction

op=00001X	address=A
6 bits	26 bits

- jump (j): 000010, jump and link (jal): 000011
- · Pseudo-direct addressing: address, not offset, is specified
- target address: how to build 32-bit address from 26 bits included in instruction?
- Ans:
  - bottom 2 bits are 00 (instruction (word) address is always a multiple of 4)
  - top 4 bits stay same as current (entire program should fit within a 2<sup>28</sup> byte
     = 2<sup>26</sup> word > 67 million instruction block)
- e.g., @ addr 0101 1110 0101 1100 0011 1100 1010 1100:
  - j 1010 0101 1000 0010 1010 1111 10
  - new addr: 0101 1010 0101 1000 0010 1010 1111 1000



#### Target Addressing Example

- Loop code from earlier example
- Assume loop at location 80000

```
80000
                                              19
Loop: sll $t1, $s3, 2
                                           0
                                                       0
       add $t1, $t1, $s5
                                80004
                                              21
                                                       32
                                        35
       lw $t0, 0($t1)
                                80008
                                        5
       bne $t0, $s4, Exit
                                80012
                                              20
                                                           Note: divided by 4
       addi $s3, $s3, 1
                                80016
                                        8
                                                           because last 2 bits
                                                              omitted
                                              20000 ←
                                80020
       j Loop
Exit:
                                80024
```



#### Relative v. Direct Addressing

- Why are branches addressed in a relative manner while jumps are (pseudo)direct?
  - branch distance is usually short (e.g., while or for loop), so relative values will usually be small (easily fit in 16 bits)
  - jumps often used to reach external code (e.g., standard procedures at fixed locations, like a sqrt() procedure)
    - using direct addressing means just using a fixed value rather than computing offset

#### That f\*\*\*\*\*g offset by 4

- Data Memory access (e.g., lw, sw): there are times a single byte access (in the middle of the word) might be desired, hence constants, registers are in address (byte) units.
- Instruction memory access (i.e., jumps and branches): since instructions always start an address A where A mod 4 = 0, offsets (in branches) and addresses (in jumps) within instruction are in instruction (word) units.
  - Note: if they were in address units, the 2 lowest order bits would always be 0 anyhow, why waste the bits?

# Where to find various opcodes and funcs

				op(31:26)		MINE .		
28-26	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
31-29	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	sltiu	andi	ori	xori	load upper imm
2(010)	TLB	FlPt						
3(011)			1000	a traible igns		The P		
4(100)	load byte	load half	lwl	load word	1bu	1hu	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	1wc0	1wc1				10 110		
7(111)	swc0	swc1	(Laphine A.	A Janoele Dy Gill	Welenblin.			
		entral marie in	Removed 238	debute of or	describb A			
			op(31:26)=	:010000 (TLB), i	rs(25:21)			
23–21 25–24	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(00)	mfc0		cfc0	by a thirteen and	mtc0	I KI I II	ctc0	
1(01)			Contract of the contract of th		Length of L		le force by	al drawn when he was
2(10)						1 1111		
3(11)	MD MAKANDUCHS		yrenyd Ma	A Sungle Alfale	WINNE HOLDE	14	0 100 0000	

	op(31:26)=000000 (R-format), funct(5:0)										
2-0 5-3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)			
0(000)	shift left logical		shift right logical	sra	sllv	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	srlv	srav			
1(001)	jump reg.	jalr	- 1 - 000000	4	syscall	break					
2(010)	mfhi	mthi	mflo	mtlo							
3(011)	mult	multu	div	divu	1 1,27						
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)			
5(101)	I Charles to		set l.t.	sltu	Ted school		The state of the	a legionem per per			
6(110)				1000000		1000					
7(111)		The same			7 12 1101						

FIGURE 2.25 MIPS instruction encoding. This notation gives the value of a field by row and by column. For example, the top portion of the figure shows 10 ad word in row number 4 (100<sub>two</sub> for bits 31–29 of the instruction) and column number 3 (011<sub>two</sub> for bits 28–26 of the instruction) and column number 3 (011<sub>two</sub> for bits 28–26 of the instruction) and column of (op = 000000<sub>two</sub>) is defined in the bottom part of the figure. Hence, Subtract in row 4 and column 2 of the bottom section means that the funct field (bits 5–0) of the instruction is 100010<sub>two</sub> and the op field (bits 31–26) is 000000<sub>two</sub>. The F1Pt value in row 2, column 1 is defined in Figure 3.20 in Chapter 3. Bltz/gez is the opcode for four instructions found in Appendix A: bltz, bgez, bltzal, and bgezal. Chapter 2 describes instructions given in full name using color, while Chapter 3 describes instructions given in memonics using color. Appendix A covers all instructions.

				op(31:26)				
28-26	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
31-29	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	sltiu	andi	ori	xori	load upper imm
2(010)	TLB	FIPt						
3(011)	Maria Maria		Land America			The P		
4(100)	load byte	load half	lwl	load word	1bu	1hu	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	1wc0	1wc1						
7(111)	swc0	swc1	ILagolido JE	A Tringe of the	estentia.			
	San State of the S	estal out to a	Keined 28	detailed at	control like			
			op(31:26)=	:010000 (TLB), I	rs(25:21)			
23-21	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
25-24			The Man	or is the converse	CEASIDIAN.		n lest mits	ri samulata
0(00)	mfc0		cfc0	he to the second	mtc0	I GO THE	ctc0	Harry March
L(O1)	M. Ki _							
2(10)	R-type	e instruc	tions wit	th opcode	e=0000	)00, wl	hat is f	unct?
3(11)	MI I PARAMAGANA		4 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6					m Paradalla (1957)

	op(31:26)=000000 (R-format), funct(5:0)										
2-0 5-3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)			
0(000)	shift left logical		shift right logical	sra	sllv	1300	srlv	srav			
1(001)	jump reg.	jalr	- s- comin n	*	syscall	break					
2(010)	mfhi	mthi	mflo	mtlo							
3(011)	mult	multu	div	divu	1 2 2 2 1 1 1 1						
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)			
5(101)	The state of the		set l.t.	sltu	Ted telinbu		The state of the	a legitarini il li li li li			
6(110)				FLLDVILLE)	The same	268-11					
7(111)											

FIGURE 2.25 MIPS instruction encoding. This notation gives the value of a field by row and by column. For example, the top portion of the figure shows 10ad word in row number 4 (100<sub>two</sub> for bits 31–29 of the instruction) and column number 3 (011<sub>two</sub> for bits 28–26 of the instruction) so the corresponding value of the op field (bits 31–26) is 100011<sub>two</sub>. Underscore means the field is used elsewhere. For example, R-format in row 0 and column 0 (op = 000000<sub>two</sub>) is defined in the bottom part of the figure. Hence, Subtract in row 4 and column 2 of the bottom section means that the funct field (bits 5–0) of the instruction is 100010<sub>two</sub> and the op field (bits 31–26) is 000000<sub>two</sub>. The F1Pt value in row 2, column 1 is defined in Figure 3.20 in Chapter 3. B1tz/gez is the opcode for four instructions found in Appendix A: b1tz, bgez, b1tzal, and bgezal. Chapter 2 describes instructions given in full name using color, while Chapter 3 describes instructions given in mnemonics using color. Appendix A covers all instructions.

				op(31:26)				
28-26	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
31-29	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	sltiu	andi	ori	xori	load upper imm
(010)	TLB	FIPt						
(011)			Land HIRES	marketoly less		The state of	diam'r.	
(100)	load byte	load half	lwl	load word	1bu	1hu	lwr	
(101)	store byte	store half	swl	store word			swr	and the land
(110)	1wc0	1wc1	IGNO	ORE THIS	SIII	10 - 10		
(111)	swc0	swc1		J 1111	MIA.			

	op(31:26)=010000 (TLB), rs(25:21)											
23–21 25–24	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)				
0(00)	mfc0		cfc0	li i i remedia	mtc0	IS I Im	ctc0	Harry Wall				
1(01)			The second				September 1	dramatical spirit				
2(10)						10 110	d to 2 con num					
3(11)	MI Wasayayana	n italitatic in	Is prefyd lyfas.	Jungo RAN	A WAR LOWER	14 - 15	10 117 100000					

14	op(31:26)=000000 (R-format), funct(5:0)										
2–0 5–3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)			
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav			
1(001)	jump reg.	jalr	- 1 onoi-n	*	syscall	break					
2(010)	mfhi	mthi	mflo	mtlo							
3(011)	mult	multu	div	divu	1 1/2						
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)			
5(101)	I CHANNEL TO		set l.t.	sltu	Ted school		The state of the	a Contract of Land			
6(110)				FILL WELLEN	The same of	TOTAL STATE					
7(111)					7 12 1101						

FIGURE 2.25 MIPS instruction encoding. This notation gives the value of a field by row and by column. For example, the top portion of the figure shows 10ad word in row number 4 (100<sub>two</sub> for bits 31–29 of the instruction) and column number 3 (011<sub>two</sub> for bits 28–26 of the instruction), so the corresponding value of the op field (bits 31–26) is 100011<sub>two</sub>. Underscore means the field is used elsewhere. For example, R-format in row 0 and column 0 (op = 000000two) is defined in the bottom part of the figure. Hence, Subtract in row 4 and column 2 of the bottom section means that the funct field (bits 5–0) of the instruction is 100010<sub>two</sub> and the op field (bits 31–26) is 000000<sub>two</sub>. The F1Pt value in row 2, column 1 is defined in Figure 3.20 in Chapter 3. B1tz/gez is the opcode for four instructions found in Appendix A: b1tz, bgez, b1tzal, and bgezal. Chapter 2 describes instructions given in full name using color, while Chapter 3 describes instructions given in mnemonics using color. Appendix A covers all instructions.

0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
The second second							dentitle la initial
R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
add immediate	addiu	set less than imm.	sltiu	andi	ori	xori	load upper im
TLB	FlPt						
		Land Hitters	a retentativique	lafe	The second		
load byte	load half	lwl	load word	1bu	1hu	lwr	
store byte	store half	swl	store word			swr	ed ter in v
1wc0	1wc1				19 21 119		
swc0	swc1	Hambled JE	A Limeter of the	Make a block			

23-21	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
23- <mark>21</mark> 25- <mark>24</mark>			All th	ne rest!	TOTAL DATE	THE PARTY OF THE P	Ken merke	tabellation.
O(O <mark>O</mark> )	mfc0		cfc0		mtc0		ctc0	
1(01)					LESSON OF A		Section of the	
2(10)				7.000	Service Control	THE RESERVE		
3(11)	MINI TAKANGUCHEN	n staudys in	Details Name	Sundb still	Same to the		de mil doedin	

an/21/26)=000000 (R-format), funct(5:0)									
3 high-order bits of op-code 10)			3(011)	4(100)	5(101)	6(110)	7(111)		
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav	
1(001)	jump reg.	jalr		*	syscall	break			
2(010)	mfhi	mthi	mflo	mtlo					
3(011)	mult	multu	div	divu	1 2 2 2 1 1 1				
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)	
5(101)	DA CHANNEL TO		set l.t.	sltu	THO A RIVER		The state of	a con union i la	
6(110)				1000000		1000			
7(111)			2 7 3 4 4 4		TRanen				

FIGURE 2.25 MIPS instruction encoding. This notation gives the value of a field by row and by column. For example, the top portion of the figure shows 10 ad word in row number 4 (100<sub>two</sub> for bits 31–29 of the instruction) and column number 3 (011<sub>two</sub> for bits 28–26 of the instruction) and column number 3 (011<sub>two</sub> for bits 28–26 of the instruction) and column of (op = 000000<sub>two</sub>) is defined in the bottom part of the figure. Hence, Subtract in row 4 and column 2 of the bottom section means that the funct field (bits 5–0) of the instruction is 100010<sub>two</sub> and the op field (bits 31–26) is 000000<sub>two</sub>. The F1Pt value in row 2, column 1 is defined in Figure 3.20 in Chapter 3. Bltz/gez is the opcode for four instructions found in Appendix A: bltz, bgez, bltzal, and bgezal. Chapter 2 describes instructions given in full name using color, while Chapter 3 describes instructions given in memonics using color. Appendix A covers all instructions.

3 low-order bits of op-code

0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	1(001)	2(010)	3(011)	4(100)	3(101)	0(110)	(111)
R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
add immediate	addiu	set less than imm.	sltiu	andi	ori	xori	load upper imm
TLB	F1Pt						
The state of the s		La differen	retentally Squa	late	Thu H	diameters -	
load byte	load half	lwl	load word	1bu	1hu	lwr	
store byte	store half	swl	store word			swr	editer in 1
1wc0	1wc1				10 11		
swc0	swc1	Ottombre JE	A Limitary (II)	Walen blink			

23-21	0(000)	1(001)	2(010)	3/011)	4(100)	5(101)	6(110)	7(111)
25-24			All th	e rest!		THE RESERVE	Alternations of	eared do.
0(00)	mfc0		cfc0		mtc0	I Second In	ctc0	
1(01)			ALS VINCE				I SECTION OF THE	
1(0 <mark>1)</mark> 2(1 <mark>0</mark> )					Service Total	THE LAND		
3(11)	Managuards	n stauring o	Depter March	Jungs etter	CAMP DAY	14 -11	To arm opening	

			on(21:26)=000	000 (R-forma	t), funct(5:0)			
3 high-order bits of op-code 10)		3(011)	4(100)	5(101)	6(110)	7(111)		
0(000)	shift left		shift right	sra	sllv		srlv	srav
4/00	logical	54 Lo 268	logical		11	Lucia di		3-5-0
1(001) 2(010)	jump reg.	jalr	mflo	mtlo	syscall	break		
3(011)	mfhi mult	mthi multu	div	divu	THE STATE OF			
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)
5(101)					I A HOLL		a particular to	e est unem
6(110)	e.g., load word is 100011					243		
7(111)			10 TE 3 W F 12		9 14 . 11/94			

FIGURE 2.25 MIPS instruction encoding. This notation gives the value of a field by row and by column. For example, the top portion of the figure shows 10 ad word in row number 4 (100<sub>two</sub> for bits 31–29 of the instruction) and column number 3 (011<sub>two</sub> for bits 28–26 of the instruction) and column number 3 (011<sub>two</sub> for bits 28–26 of the instruction) and column of (op = 000000<sub>two</sub>) is defined in the bottom part of the figure. Hence, Subtract in row 4 and column 2 of the bottom section means that the funct field (bits 5–0) of the instruction is 100010<sub>two</sub> and the op field (bits 31–26) is 000000<sub>two</sub>. The F1Pt value in row 2, column 1 is defined in Figure 3.20 in Chapter 3. Bltz/gez is the opcode for four instructions found in Appendix A: bltz, bgez, bltzal, and bgezal. Chapter 2 describes instructions given in full name using color, while Chapter 3 describes instructions given in memonics using color. Appendix A covers all instructions.

3 low-order bits of op-code