Outline

- Representing Instructions in the Computer
- Conditional and unconditional branches

Instructions in the Computer

- Map register names into numbers
- $to to $t7 map to 8 → 15 (in decimal)
- $s0 to $s7 map to 16 → 23 (in decimal)
- Instruction add $t0, $s1, $s2 What is the machine language in decimal?

0	17	18	8	0	32
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Each segment is a field
- First and last field (0, 32): imply add *(lookup back cover of book for complete codes)*
- Second field (17): first source operand ($s1) Third field (18): second source operand ($s2)
- Fourth field (8): destination operand ($t0) Fifth field (0): unused
- Layout of instruction is called instruction format
- All MIPS instructions are 32 bits long (simplicity favors regularity)

MIPS Fields

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

op: operation code

rs: first register source operand

rt: second register source operand

rd: register destination operand

shamt: shift amount (will not use it within chapter 3)

funct: Function. Selects specific variant of operation

How many operations? How many functions within an operation?

MIPS Fields

Instructions are divided into three types: R, I and J.

- Every instruction starts with a 6-bit opcode.
- R-type instructions specify 3 registers, a shift amount field, and a function field
- I-type instructions specify two registers and a 16-bit immediate value
- J-type instructions follow the opcode with a 26-bit jump target

Type	-31- format (bits) -0-					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

MIPS Fields

Different kinds of instructions formats for different kinds of instructions

Previous format is R-type (register-type) or R-format

I-type used by Data transfer instructions

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

rt changes to mean destination register

Implication of 16 bit address field? Offset restricted to $\pm 2^{15} = \pm$ 32,768 bytes

lw $t0, 32 ($s3) What is machine language in decimal?

35	19	8	32
6 bits	5 bits	5 bits	16 bits

MIPS machine language

Name	Format	Example						Comments
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
add	R	0	2	3	1	0	32	add $1,$2,$3
sub	R	0	2	3	1	0	34	sub $1,$2,$3
addi	I	8	2	1		100		addi $1,$2,100
addu	R	0	2	3	1	0	33	addu $1,$2,$3
subu	R	0	2	3	1	0	35	subu $1,$2,$3
addiu	I	9	2	1		100		addiu $1,$2,100
mfc0	R	16	0	1	14	0	0	mfc0 $1,$epc
mult	R	0	2	3	0	0	24	mult $2,$3
multu	R	0	2	3	0	0	25	multu $2,$3
div	R	0	2	3	0	0	26	div $2,$3
divu	R	0	2	3	0	0	27	divu $2,$3
mfhi	R	0	0	0	1	0	16	mfhi $1
mflo	R	0	0	0	1	0	18	mflo $1
and	R	0	2	3	1	0	36	and $1,$2,$3
or	R	0	2	3	1	0	37	or $1,$2,$3
andi	I	12	2	1		100		andi $1,$2,100
ori	I	13	2	1		100		ori $1,$2,100
sll	R	0	0	2	1	10	0	sll $1,$2,10
srl	R	0	0	2	1	10	2	srl $1,$2,10
lw	I	35	2	1		100		lw $1,100($2)
sw	I	43	2	1		100		sw $1,100($2)
lui	I	15	0	1		100		lui $1,100
beq	I	4	1	2		25		beq $1,$2,100
bne	I	5	1	2		25		bne $1,$2,100
slt	R	0	2	3	1	0	42	slt $1,$2,$3
slti	I	10	2	1		100		slti $1,$2,100
sltu	R	0	2	3	1	0	43	sltu $1,$2,$3
sltiu	I	11	2	1		100		sltiu $1,$2,100
j	J	2			2500			j 10000
jr	R	0	31	0	0	0	8	jr $31
jal	J	3			2500			jal 10000

MIPS instruction formats

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Main MIPS machine language. Formats and examples are shown, with values in each field: op and funct fields form the opcode (each 6 bits), rs field gives a source register (5 bits), rt is also normally a source register (5 bits), rd is the destination register (5 bits), and shamt supplies the shift amount (5 bits). The field values are all in decimal. Floating-point machine language instructions are shown in Figure 4.47 on page 291. Appendix A gives the full MIPS machine language.

COPYRIGHT 1998 MORGAN KAUFMANN
PUBLISHERS, INC. ALL RIGHTS RESERVED

Back cover of Textbook

MIPS machine language and instruction formats

MIPS operands

Name	Example	Comments
32 registers	$s0–$s7, $t0–$t9, $gp, $fp, $zero, $sp, $ra, $at, Hi, Lo	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants. Hi and Lo contain the results of multiply and divide.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow detected
	add immediate	addi $s1,$s2,100	$s1 = $s2 + $s3	+ constant; overflow detected
	add unsigned	addu $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	subu $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow undetected
	add immediate unsigned	addiu $s1,$s2,100	$s1 = $s2 + $s3	+ constant; overflow undetected
	move from coprocessor register	mfc0 $s1,$epc	$s1 = $epc	Used to copy Exception PC plus other special registers
	multiply	mult $s2,$s3	Hi, Lo = $s2 × $s3	64-bit signed product in Hi, Lo
	multiply unsigned	multu $s2,$s3	Hi, Lo = $s2 × $s3	64-bit unsigned product in Hi, Lo
	divide	div $s2,$s3	Lo = $s2 / $s3, Hi = $s2 mod $s3	Lo = quotient, Hi = remainder
	divide unsigned	divu $s2,$s3	Lo = $s2 / $s3, Hi = $s2 mod $s3	Unsigned quotient and remainder
	move from Hi	mfhi $s1	$s1 = Hi	Used to get copy of Hi
	move from Lo	mflo $s1	$s1 = Lo	Used to get copy of Lo
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; logical AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; logical OR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Logical AND reg, constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Logical OR reg, constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2+100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte unsigned	lbu $s1,100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1,100	$s1 = 100 × 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1=0	Compare < constant; two's complement
	set less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1=0	Compare less than; natural numbers
	set less than immediate unsigned	sltiu $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

Main MIPS assembly language instruction set. The floating-point instructions are shown in Figure 4.47 on page 291. Appendix A gives the full MIPS assembly language instruction set.

COPYRIGHT 1998 MORGAN KAUFMANN
PUBLISHERS, INC. ALL RIGHTS RESERVED

Back cover of Textbook

MIPS operands and assembly language

From High-level to Machine Language

A[300] = h + A[300];

Assume $t1 has the base of the array A, and $s2 corresponds to h

Assembly

lw $t0, 1200 ($t1) # $t0 ← A[300]

add $t0, $s2, $t0 # $t0 ← h + A[300]

sw $t0, 1200 ($t1) # A[300] ← h + A[300]

Machine code in decimal

				address	
op	rs	rt	rd	shamt	funct
35	9	8		1200	
0	18	8	8	0	32
43	9	8		1200	

What we know so far

Fig. 3.6 page 121

MIPS operands

Name	Example	Comments
32 registers	$s0, $s1, ..., $s7 $t0, $t1, ..., $t7	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. Registers $s0–$s7 map to 16–23 and $t0–$t7 map to 8–15.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; data in registers
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Data from register to memory

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

FIGURE 3.6 MIPS architecture revealed through section 3.4. Highlighted portions show MIPS machine language structures introduced in section 3.4. The two MIPS instruction formats so far are R and I. The first 16 bits are the same: both contain an *op* field, giving the base operation; an *rs* field, giving one of the sources; and the *rt* field, which specifies the other source operand, except for load word, where it specifies the destination register. R-format divides the last 16 bits into an *rd* field, specifying the destination register; *shamt* field, which is unused in Chapter 3 and hence always is 0; and the *funct* field, which specifies the specific operation of R-format instructions. I-format keeps the last 16 bits as a single *address* field.

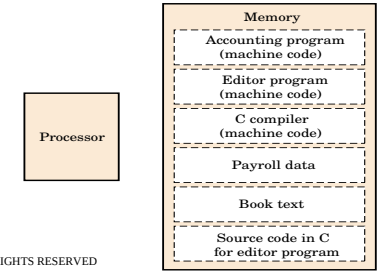
COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED

Stored-Program Concept

Two key principles

- Instructions are represented as numbers
- Programs can be stored in memory to be read or written just like numbers

Stored-Program concept



COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED

Decision Making: Branches

Decision making: *if* statement, sometimes combined with *goto* and *labels*

beq register1, register2, L1(beq: Branch if equal)

Go to the statement labeled L1 if the value in register1 equals the value in register2

bne register1, register2, L1(bne: Branch if not equal)

Go to the statement labeled L1 if the value in register1 does not equal the value in register2

beq and bne are termed Conditional branches

Compiling an If statement

```
        If (i == j) go to L1;
        f = g + h;
L1:     f = f-i;
```

f, g, h, i, and j correspond to five registers $s0 through $s4.

```
        beq $s3, $s4, L1        #go to L1 if i equals j
        add $s0, $s1, $s2       # f = g+h (skipped if i equals j)
L1:     sub $s0, $s0, $s3       # f = f –i (always executed)
```

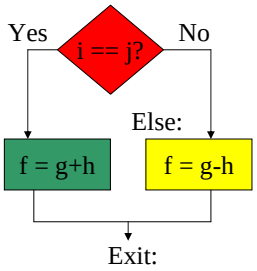
Instructions must have memory addresses

Label L1 corresponds to address of sub instruction

Compiling an if-then-else

if (i == j) f = g + h; else f = g-h;

Same variable/register mapping as previous example




```
        bne $s3, $s4, Else    # go to Else if i != j
        add $s0, $s1, $s2     # f = g + h (skipped if i != j)
        j Exit                # go to Exit (unconditional branch)
Else:   sub $s0, $s1, $s2     # f = g-h (skipped if i equals j)
Exit:
```

COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED