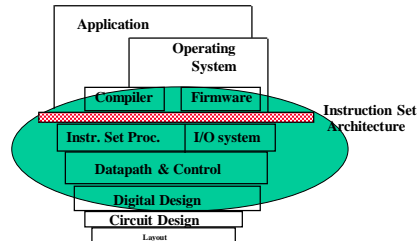
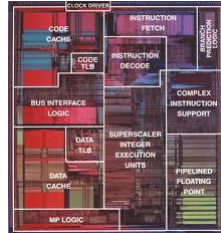


CS/SE 3340

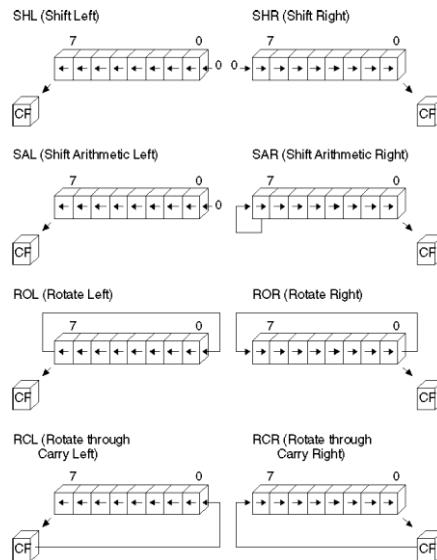
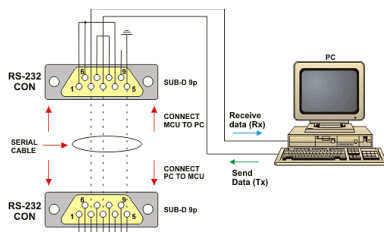
Computer Architecture



Bits and Bytes Manipulation & Other Instructions

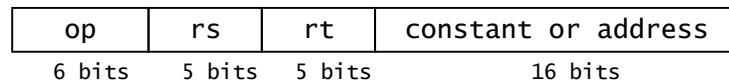
Adapted from slides by Profs. D. Patterson and J. Hennessey

X	Y	O
0	0	0
0	1	1
1	0	1
1	1	0



Branch Addressing - recap

- Branch instructions specify
 - opcode, two registers, target address
- Most branch targets are near branch (relative)
 - Forward or backward

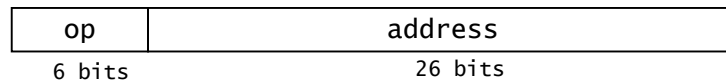


- PC-relative addressing
 - Target address = $PC + \text{offset} \times 4$
 - Note: PC *already* incremented by 4 by this time

3

Jump Addressing - recap

- Jump (j and jal) targets could be anywhere in text segment
 - Need to encode full address in instruction!



- (Pseudo) Direct addressing
 - Target address = $PC_{31...28} : (\text{address} \times 4)$

4

Target Addressing Example

- Loop code from earlier example
 - Assume Loop at location 0x800

Loop: sll \$t1, \$s3, 2	0x800	0	0	19	9	2	0
add \$t1, \$t1, \$s6	0x804	0	9	22	9	0	32
lw \$t0, 0(\$t1)	0x808	35	9	8			0
bne \$t0, \$s5, Exit	0x80C	5	8	21			2
addi \$s3, \$s3, 1	0x810	8	19	19			1
j Loop	0x814	2					0x200
Exit: ...	0x818						

5

Branching Far Away

- If branch target is too far to encode with 16-bit offset, assembler rewrites the code
- Example

```

    beq $s0,$s1, L1
      ↓
    bne $s0,$s1, L2
    j  L1
L2: ...
    
```

6

Logical Operations

- Instructions for bitwise manipulation

Operation	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR			or, ori
Bitwise NOT	~	~	nor

- Useful for manipulating (extracting and inserting) *groups of bits* in a word

7

Shift Operations

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

- **shamt**: how many positions to shift
- Shift left logical
 - Shift left and fill with 0 bits
 - **sll** by i bits *multiplies* by 2^i (unsigned only)
- Shift right logical
 - Shift right and fill with 0 bits
 - **srl** by i bits *divides* by 2^i (unsigned only)

8

AND Operations

- Useful to mask bits in a word
 - Select some bits, clear others to 0

and \$t0, \$t1, \$t2

\$t2	1100	1001	0110	0101	0000	1101	1100	0001
\$t1	0000	0000	0000	0000	0011	1100	0000	0000
\$t0	0000	0000	0000	0000	0000	1100	0000	0000

9

OR Operations

- Useful to include bits in a word
 - Set some bits to 1, leave others unchanged

or \$t0, \$t1, \$t2

\$t2	1100	1001	0110	0101	0000	1101	1100	0001
\$t1	0000	0000	0000	0000	0011	1100	0000	0000
\$t0	1100	1001	0110	0101	0011	1101	1100	0001

10

NOT Operations

- Useful to invert bits in a word
 - Change 0 to 1, and 1 to 0
- MIPS has NOR 3-operand instruction
 - $a \text{ NOR } b == \text{NOT} (a \text{ OR } b)$

`nor $t0, $t1, $zero`

Register 0: always read as zero

\$t1 0000 0000 0000 0000 0011 1100 0000 0000

\$t0 1111 1111 1111 1111 1100 0011 1111 1111

Why using NOR 3-operand instruction for NOT?

11

Character Data Revisited

- Byte-encoded character sets
 - ASCII: 128 characters
 - 95 graphic, 33 control
 - Latin-1: 256 characters
 - ASCII, +96 more graphic characters
- Unicode: 32-bit character set
 - Used in Java, C++ wide characters, ...
 - Most of the world's alphabets, plus symbols
 - UTF-8, UTF-16: variable-length encodings

12

Byte/Halfword Operations

- Could use bitwise operations
- MIPS byte/halfword load/store instructions
 - String processing is a common case

```
lb rt, offset(rs)          lh rt, offset(rs)
    – Sign extend to 32 bits in rt
lbu rt, offset(rs)         lhu rt, offset(rs)
    – Zero extend to 32 bits in rt
sb rt, offset(rs)          sh rt, offset(rs)
    – Store just rightmost byte/halfword
```

13

Serial (UART) Programming

- C code for setting a new baud rate

```
void setBaudRate (int newRate)
{
    int divisorLatch = 11520 / newRate;

    portData[3] |= 0x80; // set DLAB
    portData[1]  = divisorLatch >> 8;
    portData[0]  = divisorLatch & 0xFF;
    portData[3] &= 0x7F; // reset DLAB
}
```

▪ base of `portData` is `0xFFEE00F0`

14

Serial (UART) Programming

- MIPS code:

```
setBaudRate:
    la    $s0, maxRate
    lw    $t0, ($s0)          # t0 = maxRate
    div   $t0, $t0, $a0       # divide by newRate
    la    $s0, portData
    lw    $s1, ($s0)          # s1 = portData
    lb    $t2, 3($s1)         # get 3rd byte
    ori   $t2, $t2, 0x80      # set DLAB
    sb    $t2, 3($s1)
    srl   $t3, $t0, 8         # left byte of divisor
    sb    $t3, 1($s1)
    sb    $t0, ($s1)          # right byte of divisor
    andi  $t2, $t2, 0x7F      # reset DLAB
    sb    $t2, 3($s1)
    jr    $ra                 # and return
.data
portData: .word 0xFFEE00F0
maxRate:  .word 11520
```

15

String Copy Example

- C code (naïve):

- Null-terminated string

```
void strcpy (char x[], char y[])
{
    int i;
    i = 0;
    while ((x[i]=y[i])!='\0')
        i += 1;
}
```

- Addresses of **x**, **y** in **\$a0**, **\$a1**
- **i** in **\$s0**

16

String Copy Example

- MIPS code:

strcpy:		
	addi \$sp, \$sp, -4	# adjust stack for 1 item
	sw \$s0, 0(\$sp)	# save \$s0
	add \$s0, \$zero, \$zero	# i = 0
L1:	add \$t1, \$s0, \$a1	# addr of y[i] in \$t1
	lbu \$t2, 0(\$t1)	# \$t2 = y[i]
	add \$t3, \$s0, \$a0	# addr of x[i] in \$t3
	sb \$t2, 0(\$t3)	# x[i] = y[i]
	beq \$t2, \$zero, L2	# exit loop if y[i] == 0
	addi \$s0, \$s0, 1	# i = i + 1
	j L1	# next iteration of loop
L2:	lw \$s0, 0(\$sp)	# restore saved \$s0
	addi \$sp, \$sp, 4	# pop 1 item from stack
	jr \$ra	# and return

17

32-bit Constants

- Most constants are small
 - 16-bit immediate is sufficient
- For the occasional 32-bit constant
 - lui rt, constant** # *load upper immediate*
 - Copies 16-bit constant to left 16 bits of **rt**
 - Clears right 16 bits of **rt** to 0

lui \$s0, 61

0000 0000 0111 1101	0000 0000 0000 0000
---------------------	---------------------

ori \$s0, \$s0, 2304

0000 0000 0111 1101	0000 1001 0000 0000
---------------------	---------------------

18

Synchronization

- Two processors sharing an area of memory
 - P1 writes, then P2 reads
 - Data race if P1 and P2 don't synchronize
 - Result depends of order of accesses (**race condition!**)
- Hardware support required
 - Atomic read/write memory operation
 - No other access to the location allowed between the read and write
- Could be a single instruction
 - E.g., atomic swap of register \leftrightarrow memory
 - Or an atomic pair of instructions

19

Synchronization in MIPS

- Load linked: **ll rt, offset(rs)**
- Store conditional: **sc rt, offset(rs)**
 - Succeeds if location not changed since the **ll**
 - Returns 1 in rt
 - Fails if location is changed
 - Returns 0 in rt
- Example: atomic swap (to test/set lock variable)

```
try: add $t0,$zero,$s4 ;copy exchange value
      ll  $t1,0($s1)    ;load linked
      sc  $t0,0($s1)    ;store conditional
      beq $t0,$zero,try ;branch store fails
      add $s4,$zero,$t1 ;put load value in $s4
```

20

MIPS Instructions - recap

Category	Instruction	Op Code	Example	Meaning
Arithmetic (R format)	add	0 and 32	add \$s1, \$s2, \$s3	$\$s1 = \$s2 + \$s3$
	subtract	0 and 34	sub \$s1, \$s2, \$s3	$\$s1 = \$s2 - \$s3$
Data transfer (I format)	load word	35	lw \$s1, 100(\$s2)	$\$s1 = \text{Memory}(\$s2+100)$
	store word	43	sw \$s1, 100(\$s2)	$\text{Memory}(\$s2+100) = \$s1$
	load byte	32	lb \$s1, 101(\$s2)	$\$s1 = \text{Memory}(\$s2+101)$
	store byte	40	sb \$s1, 101(\$s2)	$\text{Memory}(\$s2+101) = \$s1$
Branch (conditional jump)	br on equal	4	beq \$s1, \$s2, L	if $(\$s1 == \$s2)$ go to L
	br on not equal	5	bne \$s1, \$s2, L	if $(\$s1 \neq \$s2)$ go to L
	set on less than	0 and 42	slt \$s1, \$s2, \$s3	if $(\$s2 < \$s3)$ $\$s1=1$ else $\$s1=0$
Unconditional jump	jump	2	j 2500	go to 10000
	jump register	0 and 8	jr \$t1	go to \$t1

21

Summary

- MIPS ISA provides instructions operating on bits
 - Logical operations: AND, OR, NOT, XOR etc..
 - Shift operations: shift left, shift right,...
- Instructions for manipulating bytes are also provided
 - Many applications require manipulations of bytes, e.g. characters handling
- Special instructions are also provided to address concurrency issues, e.g. synchronization

22