nst.eecs.berkeley.edu/~cs61c **CS61C**: Machine Structures

Lecture 12 – Introduction to MIPS Procedures II, Logical and Shift Ops

2004-09-27

Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Gotta love Sept/Oct! ⇒

Pennant races heating up! SF, As trying to make the post-season. Yankees trying to hold off Boston. Ichiro soon to beat 84yr hit record!

espn.com/mlb/

Review

- Functions called with jal, return with jr \$ra.
- The stack is your friend: Use it to save anything you need. Just be sure to leave it the way you found it.
- Instructions we know so far

Arithmetic: add, addi, sub, addu, addiu, subu

Memory: lw, sw

Decision: beq, bne, slt, slti, sltu, sltiu Unconditional Branches (Jumps): j, jal, jr

- · Registers we know so far
 - · All of them!

There are CONVENTIONS when calling procedures!

Register Conventions (1/4)

- CalleR: the calling function
- CalleE: the function being called
- When callee returns from executing, the caller needs to know which registers may have changed and which are guaranteed to be unchanged.
- Register Conventions: A set of generally accepted rules as to which registers will be unchanged after a procedure call (jal) and which may be changed.



CS 61C L12 Introduction to MIPS: Procedures II, logical & shift ops (3)

Register Conventions (2/4) - saved

- \$0: No Change. Always 0.
- •\$s0-\$s7: Restore if you change. Very important, that's why they're called saved registers. If the callee changes these in any way, it must restore the original values before returning.
- \$sp: Restore if you change. The stack pointer must point to the same place before and after the jal call, or else the caller won't be able to restore values from the stack.

,• HINT -- All saved registers start with S!

Register Conventions (3/4) - volatile

- \$ra: Can Change. The jal call itself will change this register. Caller needs to save on stack if nested call.
- \$v0-\$v1: Can Change. These will contain the new returned values.
- \$a0-\$a3: Can change. These are volatile argument registers. Caller needs to save if they il need them after the call.
- \$±0-\$±9: Can change. That's why they're called temporary: any procedure may change them at any time. Caller needs to save if they'll need them afterwards.

Register Conventions (4/4)

- What do these conventions mean?
 - · If function R calls function E. then function R must save any temporary registers that it may be using onto the stack before making a jal call.
 - Function E must save any S (saved) registers it intends to use before garbling up their values
 - · Remember: Caller/callee need to save only temporary/saved registers they are using, not all registers.



Parents leaving for weekend analogy (1/5)

- Parents (main) leaving for weekend
- •They (caller) give keys to the house to kid (callee) with the rules (calling conventions):
 - · You can trash the temporary room(s), like the den and basement (registers) if you want, we don't care about it
 - BUT you'd better leave the rooms (registers) that we want to save for the guests untouched. "these rooms better look the same when we return!"



Parents leaving for weekend analogy (2/5)

- Kid now "owns" rooms (registers)
- Kid wants to use the saved rooms for a wild, wild party (computation)
- What does kid (callee) do?

ction to MIPS: Procedures II, logical & shift ops (8)

- · Kid takes what was in these rooms and puts them in the garage (memory)
- · Kid throws the party, trashes everything (except garage, who goes there?)
- · Kid restores the rooms the parents wanted saved after the party by replacing the items from the garage (memory) back into those saved rooms

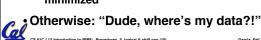
Parents leaving for weekend analogy (3/5)

- Same scenario, except before parents return and kid replaces saved rooms...
- Kid (callee) has left valuable stuff (data) all over.
 - Kid's friend (another callee) wants the house for a party when the kid is away
 - · Kid knows that friend might trash the place destroying valuable stuff!
 - Kid remembers rule parents taught and now becomes the "heavy" (caller), instructing friend (callee) on good rules (conventions) of house.



Parents leaving for weekend analogy (4/5)

- If kid had data in temporary rooms (which were going to be trashed), there are three options:
 - Move items directly to garage (memory)
 - Move items to saved rooms whose contents have already been moved to the garage (memory)
 - Optimize lifestyle (code) so that the amount you've got to shlep stuff back and forth from garage (memory) is minimized



Parents leaving for weekend analogy (5/5)

- Friend now "owns" rooms (registers)
- Friend wants to use the saved rooms for a wild, wild party (computation)
- What does friend (callee) do?
 - Friend takes what was in these rooms and puts them in the garage (memory)
 - Friend throws the party, trashes everything (except garage)
 - Friend restores the rooms the kid wanted saved after the party by replacing the items from the garage (memory) back into those saved rooms
 C L12 Introduction to MIPS: Procedures II, logical & shift ops (11)

Bitwise Operations

- · Up until now, we've done arithmetic (add, sub, addi), memory access (1w and sw), and branches and jumps.
- · All of these instructions view contents of register as a single quantity (such as a signed or unsigned integer)
- New Perspective: View register as 32 raw bits rather than as a single 32-bit number
- Since registers are composed of 32 bits, we may want to access individual bits (or groups of bits) rather than the whole.
- Introduce two new classes of instructions:

Logical & Shift Ops

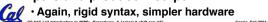
Logical Operators (1/3)

- Two basic logical operators:
 - · AND: outputs 1 only if both inputs are 1
 - · OR: outputs 1 if at least one input is 1
- Truth Table: standard table listing all possible combinations of inputs and resultant output for each. E.g.,

	A	В	A AND B	A OR B
	0	0	0	0
	0	1	0	1
	1	0	0	1
Cal	1	1	1	_ 1

Logical Operators (2/3)

- Logical Instruction Syntax:
 - 1 2,3,4
 - where
 - 1) operation name
 - 2) register that will receive value
 - 3) first operand (register)
 - 4) second operand (register) or immediate (numerical constant)
- In general, can define them to accept > 2 inputs, but in the case of MIPS assembly, these accept exactly 2 inputs and produce 1 output



Logical Operators (3/3)

- Instruction Names:
 - •and, or: Both of these expect the third argument to be a register
 - ·andi, ori: Both of these expect the third argument to be an immediate
- MIPS Logical Operators are all bitwise, meaning that bit 0 of the output is produced by the respective bit 0's of the inputs, bit 1 by the bit 1's, etc.
 - C: Bitwise AND is & (e.g., z = x & y;)
 - C: Bitwise OR is | (e.g., z = x | y;)



Uses for Logical Operators (1/3)

- Note that anding a bit with 0 produces a 0 at the output while anding a bit with 1 produces the original bit.
- This can be used to create a mask.
 - · Example:

1011 0110 1010 0100 0011 1101 1001 1010

mask:0000 0000 0000 0000 1111 1111 1111

The result of anding these:

0000 0000 0000 0000 1101 1001 1010



Uses for Logical Operators (2/3)

- The second bitstring in the example is called a mask. It is used to isolate the rightmost 12 bits of the first bitstring by masking out the rest of the string (e.g. setting it to all 0s).
- Thus, the and operator can be used to set certain portions of a bitstring to 0s, while leaving the rest alone.
 - · In particular, if the first bitstring in the above example were in \$t0, then the following instruction would mask it:

andi \$t0,\$t0,0xFFF

CS 61C L12 Introduction to MIPS: Procedures II, logical & shift ops (18)

Uses for Logical Operators (3/3)

- Similarly, note that oring a bit with 1 produces a 1 at the output while oring a bit with 0 produces the original bit.
- This can be used to force certain bits of a string to 1s.
 - · For example, if \$t0 contains 0x12345678, then after this instruction:

ori \$t0, \$t0, 0xFFFF

· ... \$t0 contains 0x1234FFFF (e.g. the high-order 16 bits are untouched, while the low-order 16 bits are forced to 1s).

CS 61C L12 Introduction to MIPS: Procedures II, logical & shift ops (19)

Shift Instructions (1/4)

- · Move (shift) all the bits in a word to the left or right by a number of bits.
 - Example: shift right by 8 bits 0001 0010 0011 0100 0101 0110 0111 1000

0000 0000 0001 0010 0011 0100 0101 0110

· Example: shift left by 8 bits 0001 0010 0011 0100 0101 0110 0111 1000

Shift Instructions (2/4)

- Shift Instruction Syntax:
 - 1 2,3,4
 - · where
 - 1) operation name
 - 2) register that will receive value
 - 3) first operand (register)
 - 4) shift amount (constant < 32)
- MIPS shift instructions:
 - 1. sll (shift left logical): shifts left and fills emptied bits with 0s
 - 2. srl (shift right logical): shifts right and fills emptied bits with 0s
 - 3. sra (shift right arithmetic): shifts right and fills

emptied bits by sign extending
CS 61C L12 Introduction to MIPS: Procedures II, logical & shift ops (21)

Shift Instructions (3/4)

- Example: shift right arith by 8 bits
- **0001 0010 0011 0100 0101 0110 0111 1000**

0000 0000 0001 0010 0011 0100 0101 0110

- Example: shift right arith by 8 bits
- **1001 0010 0011 0100 0101 0110 0111 1000**

1111 1111 1001 0010 0011 0100 0101 0110



Shift Instructions (4/4)

 Since shifting may be faster than multiplication, a good compiler usually notices when C code multiplies by a power of 2 and compiles it to a shift instruction:

a *= 8; (in C)

would compile to:

\$s0,\$s0,3 (in MIPS)

- · Likewise, shift right to divide by powers of 2
 - · remember to use sra



Peer Instruction

```
# R/W $s0,$v0,$t0,$a0,$sp,$ra,mem
              SH REGISTER(S) TO STACK?
         Call e
R/W $s0,$v0,$t0,$a0,$sp,$ra,mem
jal e
jr $ra # Return to caller of r
       # R/W $s0,$v0,$t0,$a0,$sp,$ra,mem
jr $ra # Return to r
```

What does r have to push on the stack before "jal e"?

```
Nothing
       1 of ($s0,$sp,$v0,$t0,$a0,$ra)
2 of ($s0,$sp,$v0,$t0,$a0,$ra)
3 of ($s0,$sp,$v0,$t0,$a0,$ra)
4 of ($s0,$sp,$v0,$t0,$a0,$ra)
5:
       5 of ($s0,$sp,$v0,$t0,$a0,$ra)
6:
```

CS 61C L12 Introduction to MIPS: Procedures II, logical & shift ops (24)

"And in Conclusion..."

- Register Conventions: Each register has a purpose and limits to its usage. Learn these and follow them, even if you're writing all the code yourself all the code yourself.
- Logical and Shift Instructions
 - · Operate on bits individually, unlike arithmetic, which operate on entire word.
 - · Use to isolate fields, either by masking or by shifting back and forth.
 - Use shift left logical, s11, for multiplication by powers of 2
 - Use <u>shift right arithmetic</u>, sra, for division by powers of 2.

and, andi, or, ori, sll, srl, sra

CS 61C L12 Introduction to MIPS: Procedures II, logical & shift ops (26)