Instruction sets
MIPS assembly language
Part 2

CS207, Fall 2004
September 17, 2004

Arithmetic instructions

- Add
- Subtract
- The rest can be fashioned from these

MIPS

- Add and subtract have three operands
  - Simpler to implement a fixed number of operands in hardware than a variable number
  - Multiple adds, subtracts to accomplish more advanced tasks
- add a, b, c: adds the contents of memory locations a and b, and places the answer in memory location c

Add immediate

- Allows you to add (subtract) a constant value from a register
- MIPS: addi
  addi $s0, $s0, -4
  # subtract 4 from the value in $s0
A diversion: machine code

- Assembly language is just a mnemonic device for us
- Instructions actually compiled into machine code
- Machine code: binary representation of operations, operands, data, register addresses, etc.

MIPS machine code format

- 32 bits total
- opcode: 6 bits
- registers 1-3: 5 bits each
  - r1 = source1, r2 = source2, r3 = destination
- “shift” amount: 5 bits
- function code: 6 bits

Definitions

- opcode: the “name” of the instruction (add, sub, addi, lw, etc.)
- shift amount: some operations require bits to be shifted left or right; this says how many bits (later)
- function code: some operations have variations; if so, this field contains the code for the variation

MIPS examples

add $s0, $s3, $s4
000000 10011 10100 10000 00000 100000
sub $s0, $s3, $s4
000000 10011 10100 10000 00000 100010
sub $s0, $s0, $t2
000000 10000 01010 10000 01010 100010
Q:

What about the memory access instructions, like lw, sw, and addi?

Instruction types

- **R-format**: used by register-based instructions
  - add, sub
- **I-format**: used by memory-access instructions
  - lw, sw, addi
  - instruction format:
    - opcode: 6 bits
    - registers 1, 2: 5 bits each
    - constant/address: 16 bits

MIPS examples

```plaintext
lw $t0, 12($s4)
100011 10100 01000 0000000000001010
sw $t0, 12($s4)
101011 10100 01000 0000000000001010
addi $s0, $s0, -4
001000 10000 10000 1111111111111100
```

(note: 2's complement for negative numbers)

Logical operations

- **AND**
  - 1 if both operands are 1, 0 otherwise
- **OR**
  - 1 if either or both operands are 1, 0 otherwise
- **NOT**
  - “flip” the bit
- **shift {left, right}**
  - move the bits {left, right} by the specified amount
  - e.g.: shift 0000001010001101 right by 2: 0000000001010011
MIPS logical instructions

- \texttt{sll}: shift left logical
- \texttt{srl}: shift right logical
- \texttt{and}
- \texttt{andi}: AND immediate (AND a register's contents with a constant)
- \texttt{or}
- \texttt{ori}: or immediate (OR a register's contents with a constant)
- \texttt{nor}: NOT OR (A NOR 0 = NOT A)
  - preserves 2-operand format

MIPS logical instructions: examples

\begin{verbatim}
sll $s1, $s1, 10  
000000 00000 10011 01010 000000
and $s0, $t1, $s2  
000000 01001 10010 10000 00000 011000
ori $s3, $t1, 15  
01101 01001 10011 0000000000001111
nor $s4, $t1, $t3  # assume $t3 contains 0  
000000 01101 01011 10100 00000 011011
\end{verbatim}

Decision-making operations

- High-level languages: if-else, for, while, etc.
- Instruction set: can be accomplished with a combination of the following:
  - branch on comparison
  - goto (“jump”)
  - labels

MIPS decision-making instructions

- \texttt{beq}: branch if equal
- \texttt{bne}: branch if not equal
- \texttt{j}: jump (unconditional)
- \texttt{slt/slti}: set on less than \{register, immediate\}
  - compares the contents of 2 registers; 3rd register is set to 1 if first register is less than second register, 0 otherwise
- jump address table: list of addresses for various program possibilities