Administrivia

Slide 1

- None really.
- Quiz 2 scores — high was 10 (5 of them), low was 4.5 (3 of them). Very strange distribution!

Addition/Subtraction and Overflow

Slide 2

- If adding two n-bit numbers, result can be too big to fit in n bits — "overflow".
- For unsigned numbers, how could we tell this had happened?
- How about for signed numbers?

Slide 3

Addition/Subtraction and Overflow, Continued

- Recall that we can't get overflow unless input operands have the same sign.
- If we add two positive numbers and get overflow, how can we tell this has happened? Does this always work?
- If we add two negative numbers and get overflow, how can we tell this has happened? Does this always work?

Slide 4

Addition/Subtraction and Overflow, Continued

- When we detect overflow, what do we do about it?
- From a HLL standpoint, we could ignore it, crash the program, set a flag, etc.
- To support various HLL choices, MIPS architecture includes two kinds of addition instructions:
  - Unsigned addition just ignores overflow.
  - Signed addition detects overflow and "generates an exception" (interrupt) — hardware branches to a fixed address ("exception handler"), usually containing operating system code to take appropriate action.

  This is why, if you look at MIPS assembler for C programs, the arithmetic is unsigned — C ignores overflow, so why bother to look for it.

Slide 5

Logical Operations

- Sometimes useful to be able to work with individual bits — e.g., to implement a compact array of boolean values.
- Thus, MIPS instruction set provides “logical operations”. Hard to say whether these exist to support C bit-manipulation operations, or C bit-manipulation operations exist because most ISAs provide such instructions!

Slide 6

“Shift” Instructions

- C << and >> (on unsigned numbers) are translated into `sll` (“shift left logical”) and `srl` (“shift right logical”).
- `sll` and `srl` do what the names imply — bits “fall off’ one side, and we add zeros at the other side. These are R-format instructions, and they use that “shift amount” field.
- When shifting left, filling with zeros makes sense. But when shifting right, we might want to extend the sign bit instead. `sra` (“shift right arithmetic”) does that.
- Examples?

Bitwise And and Or

Slide 7

- C `&` is translated into `and` or `andi`. C `|` is translated into `or` or `ori`. Format/operands are analogous to `add` and `addi`.

  (Notice/recall that C has two sets of and/or operators — logical and bitwise. These are the bitwise ones.)
- We could use these to test/set particular bits. Examples? Could we use them to, e.g., compute remainder when dividing by power of 2?

Minute Essay

Slide 8

- Suppose `$t0` contains `0xffffffff` and `$t1` contains `0x000000ff`. What is in `$t2`, `$t3`, `$t4` after the following instructions are executed? Answers in either binary or hexadecimal are fine.

```
sll $t2, $t0, 4
and $t3, $t0, $t1
or $t4, $t0, $t1
```

- Reminder: Homework 3 due by 5pm.