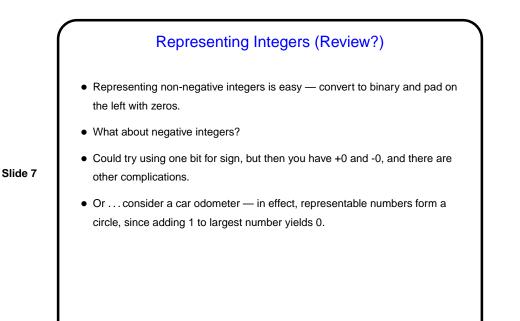
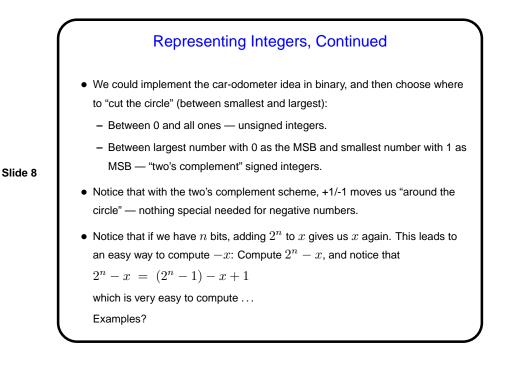
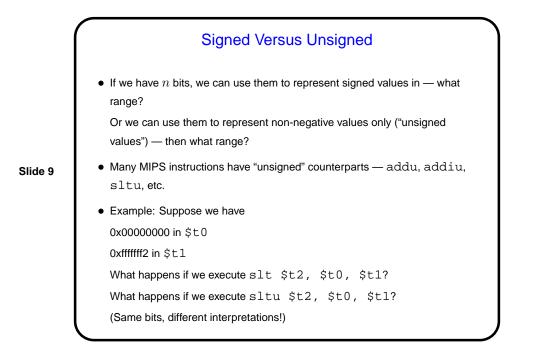


Binary Versus Decimal, Continued

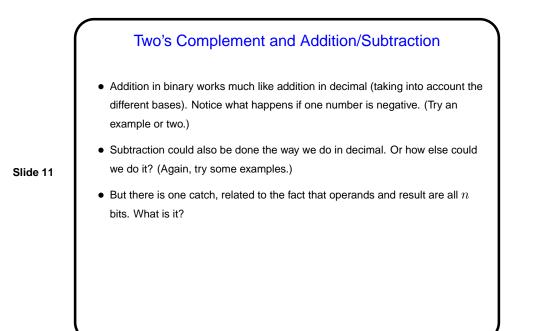
- Binary is useful for showing real internal state but not very compact. Decimal is compact but not so easy to convert to/from binary.
- We might notice easy to convert to/from a base that's a power of 2. Hence the use of "octal" (base 8) and "hexadecimal" (base 16). For the latter, we need more than 10 digits, so we use "A" through "F".
 Examples?
- Notice that we can also convert directly to/from decimal, much as we did for binary.

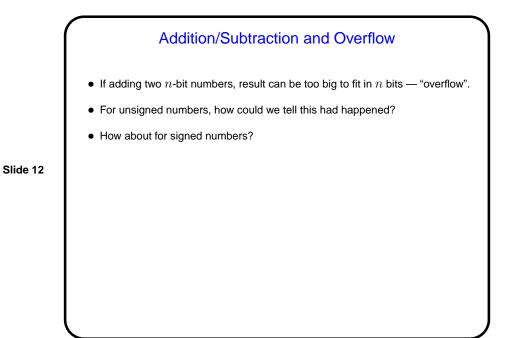


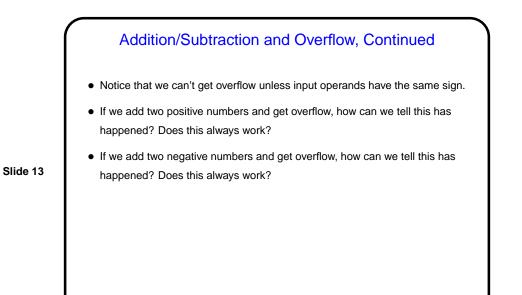




Sign Extension
If we have a number in 16-bit two's complement notation (e.g., the constant in an I-format instruction), do we know how to "extend" it into a 32-bit number? For non-negative numbers, easy.
For negative numbers, also not too hard — consider taking absolute value, extending it, then taking negative again.
In effect — "extend" by duplicating sign bit.
(Notice that not all instructions that include a 16-bit constant do this.)







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.

