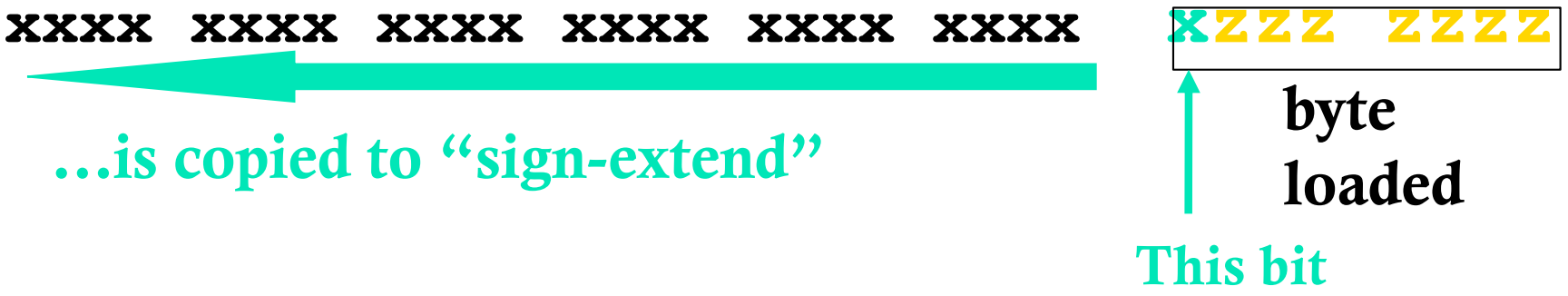
Review – MIPS Basics

- Arithmetic/logical/comparison instructions: add, addi, and, andi, sub, sll, srl, sra, slt, slti
  - Opcode dest, src1, src2
- Control instructions: bne, beq, j
  - beq/bne src1, src2, LABEL
  - j LABEL
- Data transfer instructions: lw, sw, lb, sb, lbu
  - Opcode data_reg, offset(addr_reg)

Byte/Halfword Operations

- MIPS byte/halfword load/store
  - Used in string processing: character in C is a byte
  - lb/sb/lh/sh: same format as lw, sw
- What to do with other 24/16 bits in the 32 bit register?
  - Sign-extend: lb, lh



  - Zero-extend: lbu, lhu
  - Store ops just update with the low byte/halfword

MIPS Loads

- lw: load word
  - Example: lw $t0, 4($s0)
  - Other data sizes: lb, lh, lbu, lhu
- la: load address (pseudo-instruction)
  - Example: la $t0, INPUT
- li: load immediate (pseudo-instruction)
  - Example: li $t0, 1024

Example: Loop & Memory

- Demo

Inequality Checking/Branching

- Equality checking: beq/bne; how about branch on other conditions?
  - `if (g<h) goto Less #g:$s0, h:$s1`
- Option one: similar to beq/bne, add more instructions blt, ble, bgt, bge …
- Option two: keep branch instructions very simple (only beq and bne), but add some instructions to compare and set the result
  - → MIPS's choice

Branch Instruction Design

- Why not option one (blt, bge, etc)?
- Hardware for inequality checking (>, >=, …) is slower than equality checking (== or !=)
  - Combining with branch involves more work per instruction, requiring a slower clock
  - All instructions penalized!
- beq and bne are quite common
  - Keep them to deal with the common cases but avoid adding other branching variations
  - A design compromise!

MIPS Inequality Instructions

- Set result to 1 if a condition is true
  - Otherwise, set to 0
- `slt rd,rs,rt`
  - "Set on Less Than"
  - if (rs < rt) rd = 1; else rd = 0;
- `slti rt,rs,constant`
  - if (rs < constant) rt = 1; else rt = 0;

MIPS Inequality Instructions

- How do we use this? Compile by hand:
  `if (g<h) goto Less #g:$s0, h:$s1`
- Answer: compiled MIPS code…

```
slt $t0,$s0,$s1 # $t0 = 1 if g<h
bne $t0,$0,Less # goto Less
                # if $t0!=0
                # (if (g<h)) Less:
```

  - Branch if $t0 != 0 ➔ (g < h)
  - Register $0 always contains the value 0, so bne and beq often use it for comparison after an slt instruction
- A slt ➔ bne pair means if(… < …)goto…

MIPS Inequality Instructions

- We use slt+bne to implement <, but how do we implement >, ≥and ≤?
- MIPS/RISC design principle: keep the set of instructions minimal
  - Do not ask the hardware to support more branch or set operations
  - Implement sgt/sge/sle and bge/bgt/ble/blt as pseudo instructions
    - Still use slt and bne/beq to do the work
- Exercise
  - How to implement > using just slt and bne/beq?
  - How about ≥ ?
  - How about ≤ ?

Signed vs. Unsigned

- Signed comparison: `slt`, `slti`
- Unsigned comparison: `sltu`, `sltui`
- Example
  - $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  - $s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - `slt  $t0, $s0, $s1  # signed`
    - –1 < +1 ⇒ $t0 = 1
  - `sltu $t0, $s0, $s1  # unsigned`
    - +4,294,967,295 > +1 ⇒ $t0 = 0

Core vs. Pseudo-Instructions

- Core MIPS instructions: directly implemented by hardware
  - Small group, regular format
- Pseudo-instructions
  - Not directly implemented by hardware
  - Assembler needs to translate each into equivalent core instruction(s) before generating machine code
    - Register $1: reserved for assembler use
    - Register $0: constant zero

NOT Operation

- Useful to invert bits in a word (*pseudo-instruction*)
  - Change 0 to 1, and 1 to 0
- MIPS has NOR 3-operand instruction
  - `nor $t0,$t1,$t2 #$t0= not($t1 or $t2)`
  - How to implement NOT?
    - `nor $t0, $t1, $zero`

      `#$t0= not($t1 or 0) = not($t1)`

NEG Operation

- Negate a value (*pseudo-instruction*)
  - neg $dest, $src
  - Implementation using core instructions?

  - sub $dest, $0, $src

ABS Operation

- Absolute value (*pseudo-instruction*)
  - abs $dest, $src
  - Implementation using core instructions?

Summary

- MIPS basic instructions
  - Arithmetic instructions: add, addi, sub
  - Logical operations: and, andi, or, ori, nor, sll, srl
  - Data transfer instructions: lw, sw, lb, sb, lbu
  - Control instructions: slt, slti, beq, bne, j
- MIPS formats
  - Opcode dest, src1, src2
  - Opcode data_reg, offset(addr_reg)
  - Opcode src1, src2, Label
  - j Label