Inequalities in RISC-V

- **Testing equalities**
  - == and != in C
  - beq and bne in RISC-V
- **What about <, >= ?**
  - blt t1, s1, TGT # if (t1 < s1) goto TGT
  - bge t1, s1, TGT # if (t1 >= s1) goto TGT
- **No ble or bgt. Why?**
  - RISC-V goal: Simpler is Better

>, ≤ in RISC-V

- **Compiler swaps the logic of the comparison to fit the available branch operators**

if (c > d) goto Less;	blt s3, s2, Less # if (d<c) goto Less
if (c <= d) goto Less2;	bge s3, s2, Less2 # if (d>=c) goto Less2

c: s2; d: s3

Complete the RISC-V Code

```
for(i = 0; i < 100; i++)
   b = b + A[i];
```

Assume:
i is in t1
b is in s1
base address of A is in s0

Label	Instruction
	addi t1, x0, 0
	addi t2, x0, 100
Loop:	lw t0, 0(s0)
	addi t1, t1, 1
	add s1, s1, t0
	addi s0, s0, 4
	blt _____ t1, t2, Loop

Exercise

Given: for (i = 50; i > 10; --i), where i is in s0, and 10 is in t0, which branch is the most appropriate?

a) blt s0, t0, Loop
b) blti s0, 10, Loop
c) bge s0, t0, Loop
d) blt t0, s0, Loop

Signed vs. Unsigned Comparison

- **Comparison instructions must deal with dichotomy between signed and unsigned numbers: blt vs. bltu**
  - MSB = 1
    - Signed: negative
    - Unsigned: large number
- **Example**
  - s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  - s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - blt s0, s1, L1 # -1<1, hence branch to L1
  - bltu s0, s1, L1 # 2^{32}-1 >1, hence do not branch

Bounds Check Shortcut

- **if ((k>=size) || (k<0)) goto IndexOutofBounds**
- **a1:k; a2: size, assuming size>0**

bgeu a1, a2, IndexOutofBounds # if k>=size or k<0

- **bgeu can do BOTH out-of-bounds check and a negative check**
  - All negative numbers, if interpreted as unsigned, are greater than positive numbers

RISC-V Decision-Making Instructions

beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends

Topics for Chapter 2

- **Part 1**
  - RISC-V Instruction Set (Chapter 2.1)
  - Arithmetic instructions (Chapter 2.2)
  - Introduction to RARS
  - Memory access instructions (Chapters 2.3)
  - Bitwise instructions (Chapter 2.6)
  - Decision making instructions (Chapter 2.7)
  - ➔ Arrays vs. pointers (Chapter 2.14)
- **Part 2:**
  - Procedure Calls (Chapters 2.8)
- **Part 3:**
  - RISC-V Instruction Format (Chapter 2.5)
  - RISC-V Addressing Modes (Chapter 2.10)

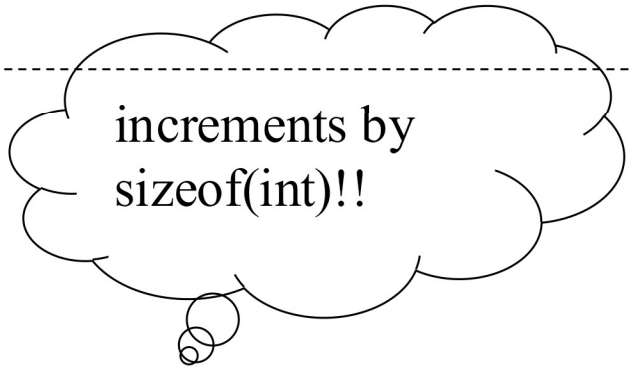
C Pointers

- **Each variable is assigned to memory or a register**
  - If assigned to memory, it is a stored value and has an address
  - Registers do not have addresses
- **In C**
  - & (reference operator): used to get the memory address of a stored value
  - * (dereference operator): used to get the value at a memory address
- **Array access is really just dereference**
  - *var is equivalent to var[0]
  - *(var+n) is equivalent to var[n]
  - Compiler knows how big the entries are, so it does the correct arithmetic to get the correct offset (i.e., n*size of the entry)
  - You are the compiler in this class

Arrays vs. Pointers

```
ClearArray (int array[ ], int size)
{
    int i;
    for (i=0; i<size; i+=1)
        array[ i ]=0;
}
```

```
ClearPtr(int *array, int size)
{
    int *p;
    for (p= &array[0]; p< &array[size]; p=p+1)
        *p=0;
}
```



Arrays Version (C)

```
ClearArray (int array[ ], int size)
{
    int i;
    for (i=0; i<size; i+=1)
        array[ i ]=0;
}
```

Assume size>0
array: a0
size: a1
i: t0

Translated C Code:

```
                i = 0;
loop1:          array[i] = 0;
                i++;
                if (i < size) goto loop1;
```

Arrays Version (RISC-V)

Translated C Code:

```
        i = 0;
loop1:  array[i] = 0;
        i++;
        if (i < size) goto Loop1;
```

Assume size>0
array: a0
size: a1
i: t0

```
        add   t0, zero, zero     # i=0
loop1:  slli  t1, t0, 2          # t1=i*4
        add   t2, a0, t1         # t2=&array[i]
        sw    zero, 0 (t2)       # array[i]=0
        addi  t0, t0, 1          # i= i+1
        blt   t0, a1, loop1      # if () goto loop1
```

Pointer Version (C)

```
ClearPtr(int *array, int size)
{
    int *p;
    for (p= &array[0]; p< &array[size]; p=p+1)
        *p=0;
}
```

Assume size>0
array: a0
size: a1
p: t0

Translated C Code:

```
                p = &array[0];
loop2:          *p = 0;
                p ++;
                if (p < &array[size]) goto loop2;
```

Pointer Version (RISC-V)

Translated C Code:

```
        p = &array[0];
loop2:  *p = 0;
        p ++;
        if (p < &array[size]) goto loop2;
```

Assume size>0
array: a0
size: a1
p: t0

```
        add   t0, a0, x0     # p=address of array[0]
loop2:  sw    zero, 0(t0)    # memory[p]=0
        addi  t0, t0, 4      # p= p+4
        slli   t1, a1, 2     # t1=size *4
        add    t2, a0, t1    # t2=address of array[size]
        blt   t0, t2, loop2  # if (p<&array[size]) goto loop2
```

Arrays vs. Pointers (RISC-V)

```
        add t0, zero, zero # i=0
loop1:  sll   t1, t0, 2              # t1=i*4
        add   t2, a0, t1     # t2=&array[i]
        sw    zero, 0 (t2)   # array[i]=0
        addi  t0, t0, 1              # i= i+1
        blt   t0, a1, loop1  # if (i<size) goto loop1
```

```
        slli    t1, a1, 2            # t1=size *4
        add     t2, a0, t1   # t2=address of array[size]
        add     t0, a0, 0    # p=address of array[0]
loop2:  sw      zero, 0(t0)  # memory[p]=0
        addi    t0, t0, 4            # p= p+4
        blt     t0,t2, loop2 # if (p<&array[size]) goto loop2
```

Summary

- **In RISC-V Assembly Language:**
  - Registers replace C variables
  - One Instruction (simple operation) per line
- **No types in RISC-V**
- **Memory is byte-addressable, but lw and sw access one word at a time.**
- **A pointer (used by lw and sw) is just a memory address, so we can add to it or subtract from it (using offset).**

Summary

- **RISC-V Instructions:**
  - Arithmetic: add, addi, sub,
  - Data transfer: lw, sw, lb, sb, lbu, lh,sh,lhu
  - Bitwise: and,andi, or,ori, sll,slli, srli,sra, srai
  - Branch: bne, beq, bge, bgeu, blt, bltu, jal
  - Special: ecall, la
- **Registers:**
  - C Variables: s0 – s11
  - Temporary Variables: t0 – t6
  - Zero: x0/zero
  - Arguments to functions: a0-a7
  - Return values: a0, a1