
# RISC V ARCHITECTURE

**Rajeshwari B**
Department of Electronics and Communication Engineering

PES UNIVERSITY ONLINE

# RISC V ARCHITECTURE

## Instructions: The Language of Computer –Part-B

**Rajeshwari B**
Department of Electronics and Communication Engineering

## Arrays versus Pointers

**Array** indexing involves

- Multiplying index by element size
- Adding to array base address

**Pointers** correspond directly to memory addresses

- Can avoid indexing complexity

Comparing assembly code that uses arrays and array indices to the assembly code that uses pointers offers insights about pointers.

Examples below show  how pointers map into RISC-V instructions,

## Example: Clearing an Array

C Code

```
clear1(int array[], int size) {
  int i;
  for (i = 0; i < size; i += 1)
    array[i] = 0;
}
```

Let the two parameters array is mapped to x10
size is mapped to x11,
and i is allocated to register x5.

## Arrays versus Pointers

### Example: Clearing an Array

Assembly code

```
        li   x5,0       // i = 0
loop1: slli  x6, x5 ,2    // x6 = i * 4
        add  x7, x10, x6  // x7 = address  of array[i]
        sw   x0, 0(x7)   // array[i] = 0
        addi  x5, x5, 1    // i = i + 1
         blt  x5,x11,loop1  // if (i<size) go to loop1
```

## Arrays versus Pointers

### Pointer Version of Clear

```
clear2(int *array, int size) {
  int *p;
  for (p = &array[0]; p < &array[size];
       p = p + 1)
    *p = 0;
}
```

Let the two parameters array is mapped to x10
size is mapped to x11,
and p is allocated to register x5

## Arrays versus Pointers

### Pointer Version of Clear

Assembly code

```
    mv x5,x10       // p = address of array[0]
       OR
    add x5,x10,x0
    slli x6,x11,2   // x6 = size * 4
    add x7,x10,x6   // x7 = address of array[size]

loop2:
    sw x0,0(x5)     // Memory[p] = 0
    addi x5,x5,4    // p = p + 4
    bltu x5,x7,loop2 // if (p<&array[size]) go to loop2
```

## Arrays versus Pointers

- In Array version “multiply” and add inside the loop because i is incremented and each address must be recalculated from the new index.
- The memory pointer version increments the pointer p directly.
- The pointer version moves the scaling shift and the array bound addition outside the loop, thereby reducing the instructions executed per iteration from five to three.
- Compiler can achieve same effect as manual use of pointers
  - strength reduction (shift instead of multiply)
  - induction variable elimination (eliminating array address calculations within loops).


# THANK YOU

**Rajeshwari B**
Department of Electronics and Communication Engineering
**rajeshwari@pes.edu**