

LOOPS (1)

- A **loop** is a block of statements or instructions that may be executed more than once
 - ▷ A fundamental programming structure
 - Present in all programming languages
 - Necessary for all matrix-vector (linear algebra) algorithms
 - Necessary in all differential-equation solvers
 - ▷ Loop control statements or instructions
 - Loop initialization
 - Incrementing the induction variable (if there is one)
 - Exiting from the loop

LOOPS (2)

- Loop syntax in C:

- ▷ For loop:

```
for ( init; test; update )  
{  
    statements  
}
```

- ▷ While loop:

```
while ( expression )  
{  
    statements  
}
```

LOOPS (3)

- Loops in assembly languages:
 - ▷ Initialization before loop
 - Set a counter to control the number of times the loop is executed
 - ▷ Inside the loop:
 - ALU or floating-point instructions
 - Decrement or increment the counter
 - ▷ Test and branch
 - If yes, then transfer control to the top of the loop instruction
 - If no, then fall through the bottom of the loop
 - SPIM/SAL instructions:
 - ◇ `blt`, `beq`, `bgt` compare two numbers and branch if “true”
 - ◇ `bltz`, `beqz`, `bgtz` compare a number to 0 and branch if “true”
 - ◇ If $(\$s0) < (\$s1)$, `slt $t0,$s0,$s1` followed by `bne $t0,$0,Target` sets $(\$t0)$ to 1 and then branches to Target

LOOPS (4)

- Example of loop programming in SAL:

```
# SAL code to initialize the elements of a 1-d array
# with the value stored in location val
.data
ar1:      .word    0:20    # array of 20 integers (4 bytes each)
val:      .float   1.e-3  # value to store in array elements
base:     .word    0      # will hold address of ar1[0]
index:    .word    0      # array index; initialized here to 0
nmax:     .word    19     # no. of array elements - 1; max. value
                    #   of array index
size:     .word    4      # size of an array element, in bytes
addr:     .word    0      # absolute address of ar1[index]
offset:   .word    0      # offset from first array element
n:        .word
#
.text
__start:
    la base,ar1          # get absolute address of ar1[0]
                        # addresses of other array elements
                        # will be computed as base + offset
loop: mul   offset,index,size # loop begins here
      add  addr,offset,base   # compute absolute address of ar1[index]
      move M[addr],val       # store val in ar1[index]
      add  index,index,1     # increment array index
      sub  n,nmax,index      # subtract array index from nelems
      bgez n, loop          # branch back to loop if diff >= 0
      done
```