RISC V Architecture

Dr. Santhameena.S

Department of Electronics and Communication Engineering

RISC V Architecture

UNIT 3: Instructions: The Language of Computer

Dr. Santhameena.S
Department of Electronics and Communication Engineering

Instructions – Language of Computer

C Sort Example

- Illustrates use of assembly instructions for a C bubble sort function
  - Consider v in x10, k in x11, temp in x5
- Swap procedure (leaf)

```
void swap(long long int v[],
        long long int k)
{
  long long int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Instructions – Language of Computer

The Procedure Swap


```
swap:

  slli   x6, x11, 2   // reg x6 = k * 4
  add    x6, x10, x6  // reg x6 = v + (k * 4)
  lw     x5, 0(x6)    // reg x5 (temp) = v[k]
  lw     x7, 4(x6)    // reg x7 = v[k + 1]
  sw     x7, 0(x6)    // v[k] = reg x7
  sw     x5, 4(x6)    // v[k+1] = reg x5 (temp)
  jalr   x0, 0(x1)    // return to calling routine
```

Instructions – Language of Computer

The Sort Procedure in C

```
void sort (int v[], size_t int n)
{
    size_t i, j;
    for (i = 0; i < n; i += 1) {
        for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1) {
            swap(v,j);
        }
    }
}
```

– v in x10, n in x11, i in x19, j in x20

Instructions – Language of Computer

The Outer Loop

- Skeleton of outer loop:
  - for (i = 0; i <n; i += 1) {

```
    li   x19,0          // i = 0
for1tst:
    bge  x19,x11,exit1  // go to exit1 if x19 ≥ x11 (i≥n)

    (body of outer for-loop)

    addi x19,x19,1          // i += 1
    j for1tst           // branch to test of outer loop
exit1:
```

Instructions – Language of Computer

The Inner Loop

- Skeleton of inner loop:
  - for (j = i – 1; j >= 0 && v[j] > v[j + 1]; j – = 1) {

```
     addi x20,x19,-1     // j = i –1
for2tst:
     blt  x20,x0,exit2   // go to exit2 if X20 < 0 (j < 0)
     slli x5,x20,2       // reg x5 = j * 4
     add  x5,x10,x5      // reg x5 = v + (j * 4)
     lw   x6,0(x5)       // reg x6 = v[j]
     lw   x7,4(x5)       // reg x7 = v[j + 1]
     ble  x6,x7,exit2    // go to exit2 if x6 ≤ x7
     mv   x21, x10       // copy parameter x10 into x21
     mv   x22, x11       // copy parameter x11 into x22
     mv   x10, x21       // first swap parameter is v
     mv   x11, x20       // second swap parameter is j
     jal  x1,swap         // call swap
     addi x20,x20,-1      // j -= 1
     j  for2tst           // branch to test of inner loop
exit2:
```

Instructions – Language of Computer

Preserving Registers

- Preserve saved registers:

```
addi sp,sp,-20  // make room on stack for 5 regs
sw   x1,16(sp)  // save x1 on stack
sw   x22,12(sp) // save x22 on stack
sw   x21,8(sp) // save x21 on stack
sw   x20,4(sp)  // save x20 on stack
sw   x19,0(sp)  // save x19 on stack
```

- Restore saved registers:

```
exit1:
  lw   x19,0(sp)   // restore x19 from stack
  lw   x20,4(sp)  // restore x20 from stack
  lw   x21,8(sp) // restore x21 from stack
  lw   x22,12(sp) // restore x22 from stack
  lw   x1,16(sp)  // restore x1 from stack
  addi sp,sp, 20  // restore stack pointer
  jalr x0,0(x1)
```

Instructions – Language of Computer

RISC-V assembly version of procedure sort

Saving registers			
	sort:	addi sp, sp, -20	# make room on stack for 5 registers
		sw x1, 16(sp)	# save return address on stack
		sw x22, 12(sp)	# save x22 on stack
		sw x21, 8(sp)	# save x21 on stack
		sw x20, 4(sp)	# save x20 on stack
		sw x19, 0(sp)	# save x19 on stack
Procedure body			
Move parameters		addi x21, x10, 0	# copy parameter x10 into x21
		addi x22, x11, 0	# copy parameter x11 into x22
Outer loop		addi x19,x0, 0	# i = 0
	for1tst:	bge x19, x22, exit1	# go to exit1 if i >= n
Inner loop		addi x20, x19, -1	# j = i - 1
	for2tst:	blt x20, x0, exit2	# go to exit2 if j < 0
		slli x5, x20, 2	# x5 = j * 4
		add x5, x21, x5	# x5 = v + (j * 4
		lw x6, 0(x5)	# x6 = v[j]
		lw x7, 4(x5)	# x7 = v[j + 1]
		ble x6, x7, exit2	# go to exit2 if x6 < x7
Pass parameters and call		addi x10, x21, 0	# first swap parameter is v
		addi x11, x20, 0	# second swap parameter is j
		jal x1, swap	# call swap
Inner loop		addi x20, x20, -1	j for2tst
		jal, x0 for2tst	# go to for2tst
Outer loop	exit2:	addi x19, x19, 1	# i += 1
		jal, x0 for1tst	# go to for1tst
Restoring registers			
	exit1:	lw x19, 0(sp)	# restore x19 from stack
		lw x20, 4(sp)	# restore x20 from stack
		lw x21, 8(sp)	# restore x21 from stack
		lw x22, 12(sp)	# restore x22 from stack
		lw x1, 16(sp)	# restore return address from stack
		addi sp, sp, 20	# restore stack pointer
Procedure return			
		jalr x0, 0(x1)	# return to calling routine

Instructions – Language of Computer

Compiler optimization for Bubble Sort

- Unoptimized code has the best CPI, and O1 optimization has the lowest instruction count, but O3 is the fastest, reminding us that time is the only accurate measure of program performance.

gcc optimization	Relative performance	Clock cycles (millions)	Instruction count (millions)	CPI
None	1.00	158,615	114,938	1.38
O1 (medium)	2.37	66,990	37,470	1.79
O2 (full)	2.38	66,521	39,993	1.66
O3 (procedure integration)	2.41	65,747	44,993	1.46

Instructions – Language of Computer

The impact of programming languages, compilation versus interpretation, and algorithms on performance of sorts

Language	Execution method	Optimization	Bubble Sort relative performance	Quicksort relative performance	Speedup Quicksort vs. Bubble Sort
C	Compiler	None	1.00	1.00	2468
	Compiler	O1	2.37	1.50	1562
	Compiler	O2	2.38	1.50	1555
	Compiler	O3	2.41	1.91	1955
Java	Interpreter	–	0.12	0.05	1050
	JIT compiler	–	2.13	0.29	338

THANK YOU

Dr. Santhameena.S
Department of Electronics and Communication Engineering
santhameena.s@pes.edu