CENG311 Computer Architecture

Instructions: Language of the Computer

IZTECH, Fall 2023

09 November 2023

Arrays vs Pointers

<pre>clear1(int array[], int size) {
  int i;
  for (i = 0; i < size; i += 1)
    array[i] = 0;
}</pre>
	<pre>clear2(int *array, int size) {
	  int *p;
	  for (p = &array[0]; p < &array[size];
	       p = p + 1)
	    *p = 0;
	}</pre>

<pre>   move $t0,$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
       slt $t3,$t0,$a1 # $t3 =
                        #   (i < size)
       bne $t3,$zero,loop1 # if (…)
                           # goto loop1</pre>
	<pre> move $t0,$a0 # p = & array[0]
	       sll $t1,$a1,2 # $t1 = size * 4
	       add $t2,$a0,$t1 # $t2 =
	                       # &array[size]
	loop2: sw $zero,0($t0) # Memory[p] = 0
	       addi $t0,$t0,4 # p = p + 4
	       slt $t3,$t0,$t2 # $t3 =
	                       #(p<&array[size])
	       bne $t3,$zero,loop2 # if (…)
	                           # goto loop2</pre>

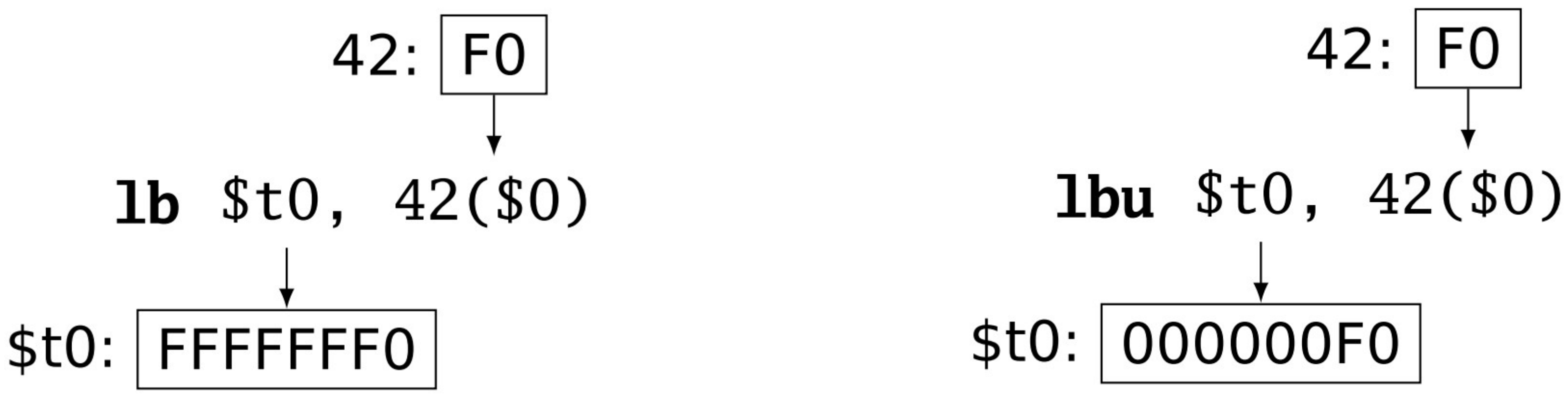
Byte/Halfword Operations

```
lb rt, offset(rs)      lh rt, offset(rs)
 (Sign extend to 32 bits in rt)

lbu rt, offset(rs)     lhu rt, offset(rs)
 (Zero extend to 32 bits in rt)

sb rt, offset(rs)      sh rt, offset(rs)
 (Store just rightmost byte/halfword)
```

Byte Load and Store



String Copy Example

C code:

```
void strcpy (char x[], char y[]){
   int i;
   i = 0;
   while ((x[i]=y[i])!='\0')
      i += 1;
}
```

Addresses of x, y in $a0, $a1

i in $s0

String Copy Example

MIPS code:

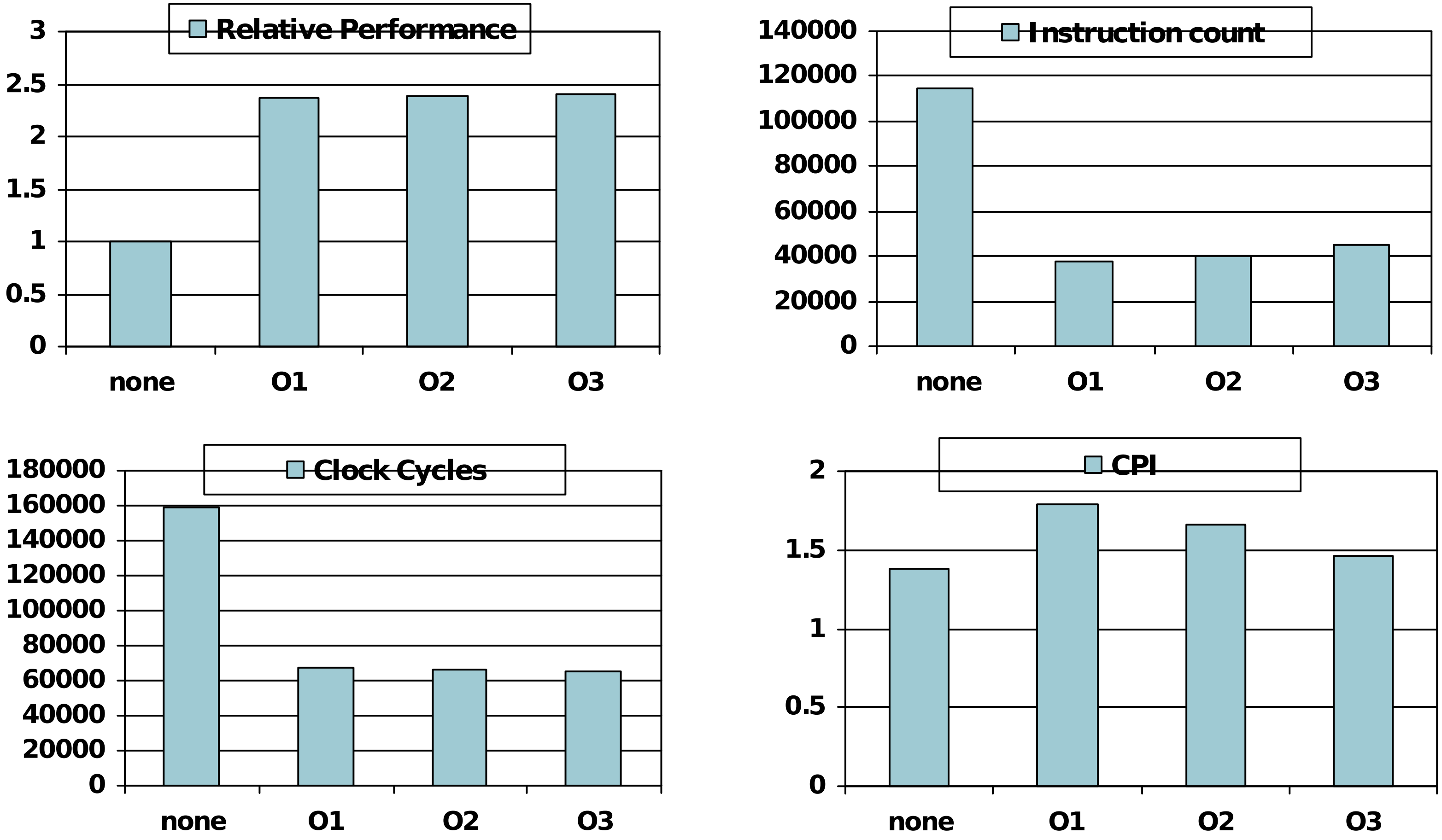
```
strcpy:
     addi $sp, $sp, -4      # adjust stack for 1 item
     sw   $s0, 0($sp)       # save $s0
     add  $s0, $zero, $zero # i = 0
L1:  add  $t1, $s0, $a1     # addr of y[i] in $t1
     lbu  $t2, 0($t1)       # $t2 = y[i]
     add  $t3, $s0, $a0     # addr of x[i] in $t3
     sb   $t2, 0($t3)       # x[i] = y[i]
     beq  $t2, $zero, L2    # exit loop if y[i] == 0
     addi $s0, $s0, 1       # i = i + 1
     j    L1                # next iteration of loop
L2:  lw   $s0, 0($sp)       # restore saved $s0
     addi $sp, $sp, 4       # pop 1 item from stack
     jr   $ra               # and return
```

gcc Optimization Levels

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

Effect of Compiler Optimization

Compiled with gcc for Pentium 4 under Linux



Compiler Optimizations

Register allocation

Dead code elimination

Instruction scheduling

Loop transformations

Register Allocation

Given a block of code, we want to choose assignments of variables to registers to <u>minimize</u> the total number of required registers

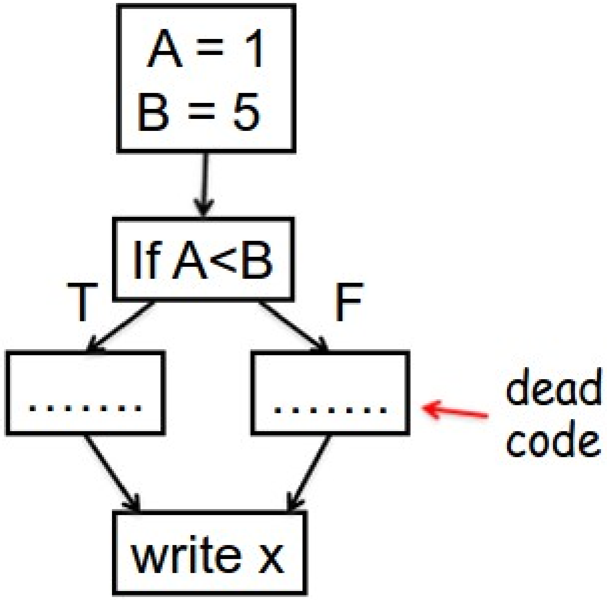
```
w = a + b; /* statement 1 */
x = c + w; /* statement 2 */
y = c + d; /* statement 3 */
```

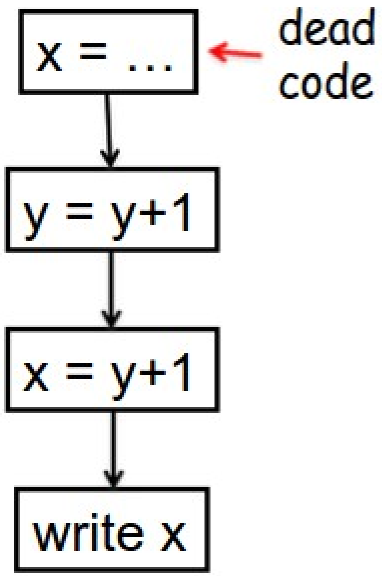
A naive register allocation assigns seven registers for the seven variables

But there is much better by reusing registers

Dead Code Elimination

Code that will never be executed can be safely removed from the program





Loop Transformations

Loop unrolling

```
for (i = 0; i < N; i++) {
        a[i] = b[i]*c[i];
}
```

If we have N=4;

Unroll 4 times:

```
a[0] = b[0]*c[0];
a[1] = b[1]*c[1];
a[2] = b[2]*c[2];
a[3] = b[3]*c[3];
```

Unroll twice:

```
for (i = 0; i < 2; i++) {
        a[i*2] = b[i*2]*c[i*2];
        a[i*2 + 1] = b[i*2 + 1]*c[i*2 + 1];
        }
```

Loop Transformations

Loop fusion: combines two or more loops into a single loop

```
for (i = 0; i < 300; i++)
  a[i] = a[i] + 3;

for (i = 0; i < 300; i++)
  b[i] = b[i] + 4;
```

→

```
for (i = 0; i < 300; i++)
  {
    a[i] = a[i] + 3;
    b[i] = b[i] + 4;
  }
```

+ Reduces loop overhead

- Memory architecture may provide better performance, if two arrays are initialized separately

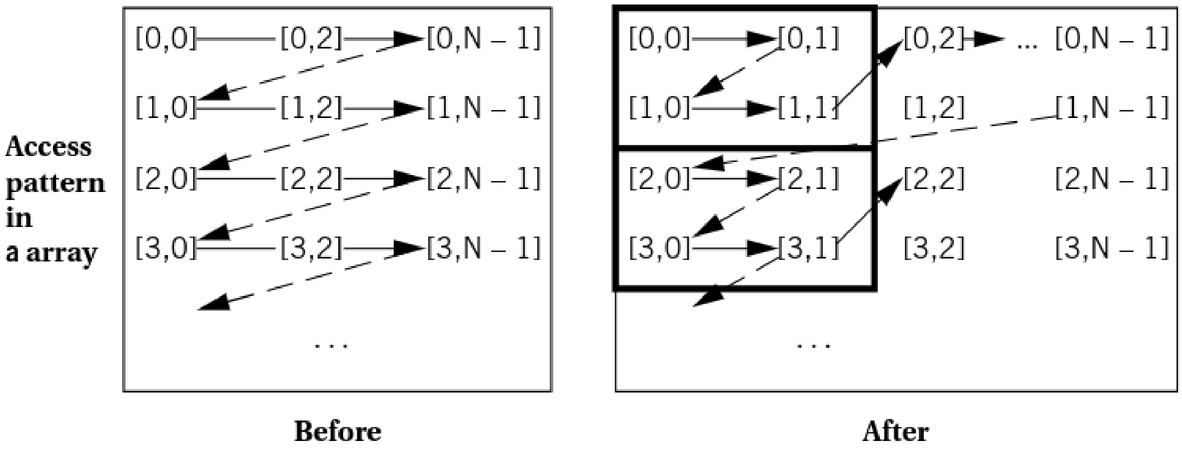
Loop Transformations

Loop tiling: breaks up a loop into a set of nested loops, with each inner loop performing the operations on a subset of the data

Code

```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        c[i] = a[i,j] * b[i];
```

```
for (i = 0; i < N; i += 2)
    for (j = 0; j < N; j += 2)
        for (ii = i; ii < min(i + 2,N); ii++)
            for (jj = j; jj < min(j + 2,N); jj++)
                c[ii] = a[ii,jj] * b[ii];
```



Procedure Inlining

The body of the procedure is substituted in place for the procedure call to eliminate procedure call overhead

$$\text{int foo(a,b,c) \{ return } a + b - c \text{ ; \}}$$

Function definition

$$z = \text{foo}(w,x,y);$$

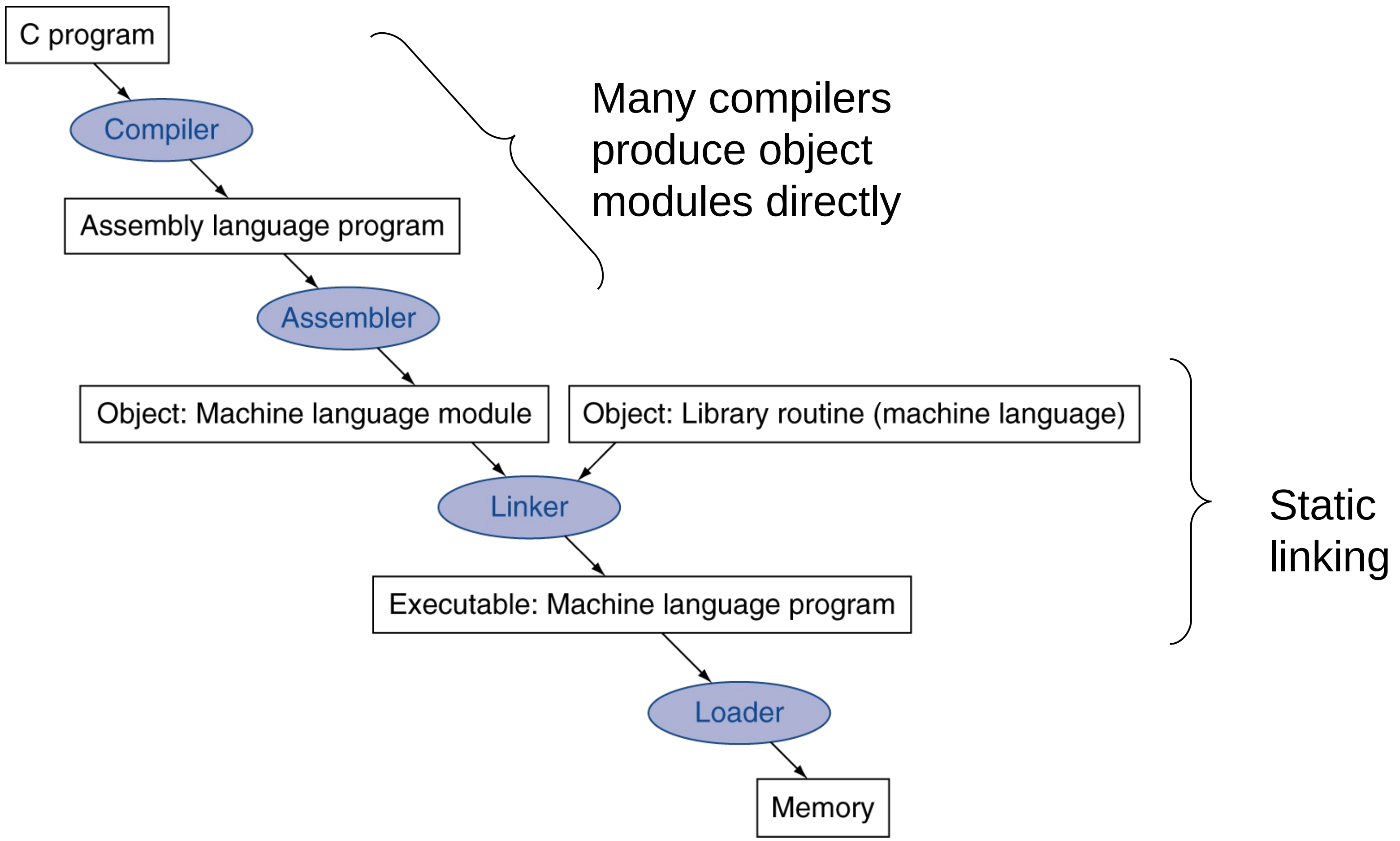
Function call

$$z = w + x - y;$$

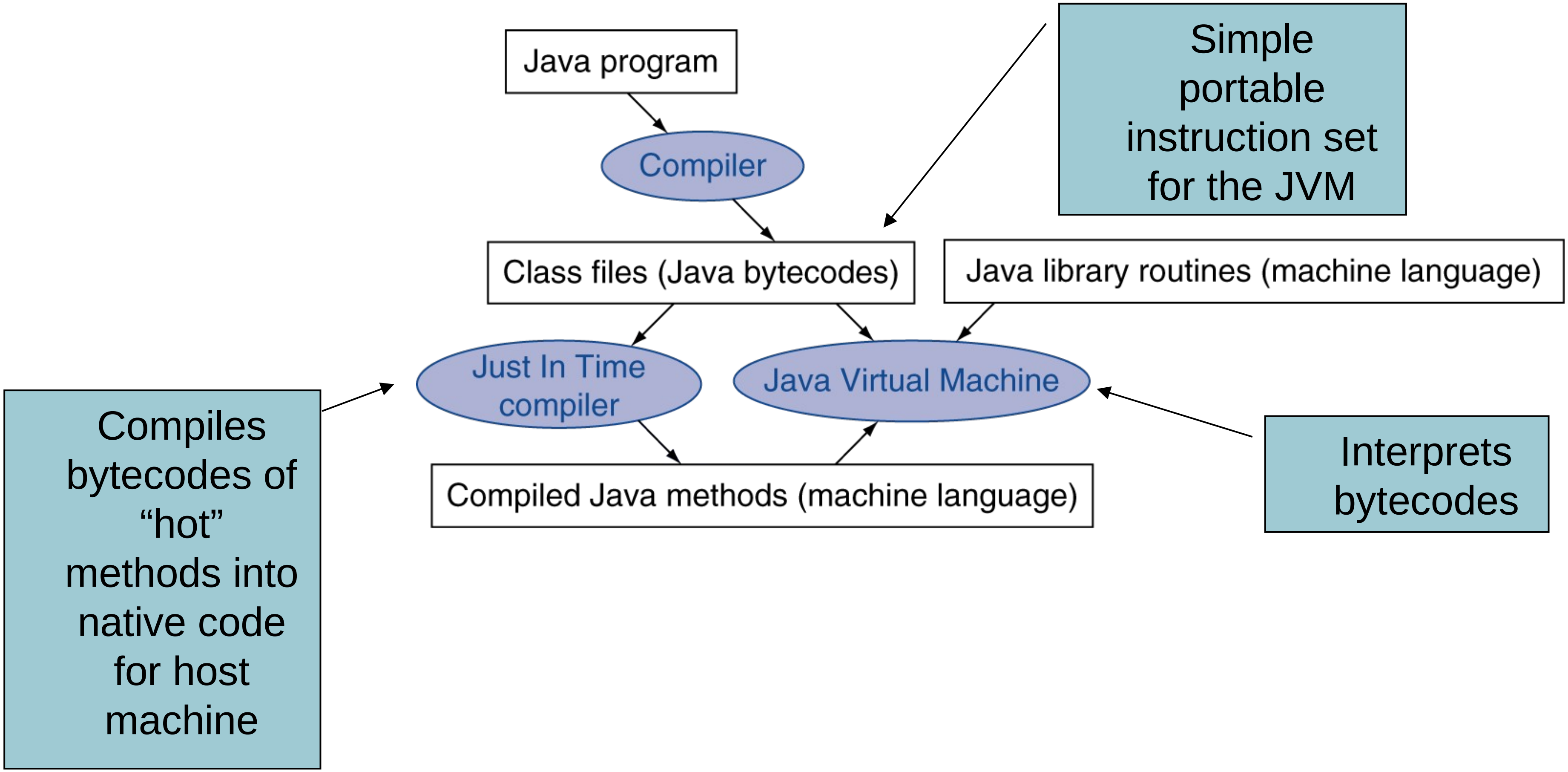
Inlining result

- Increases code size

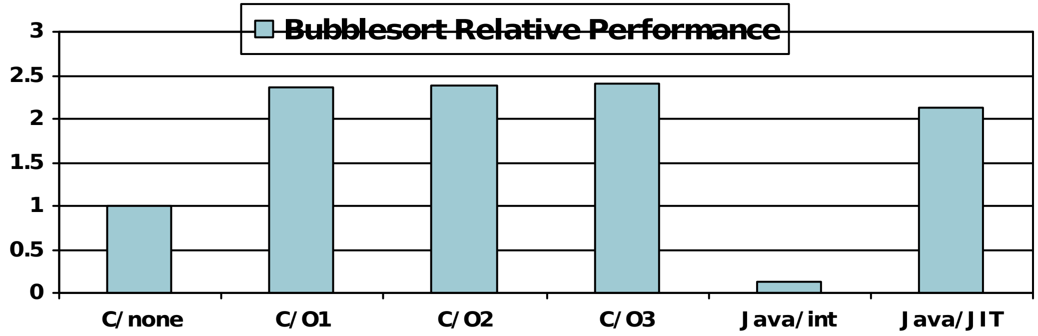
Translation and Starting a C Program

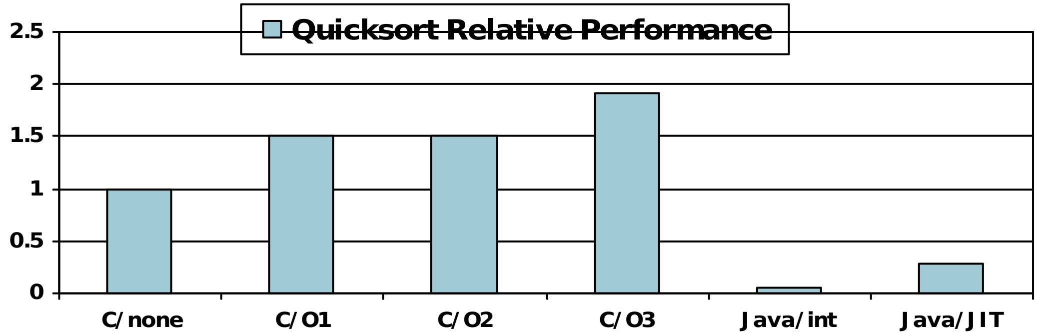


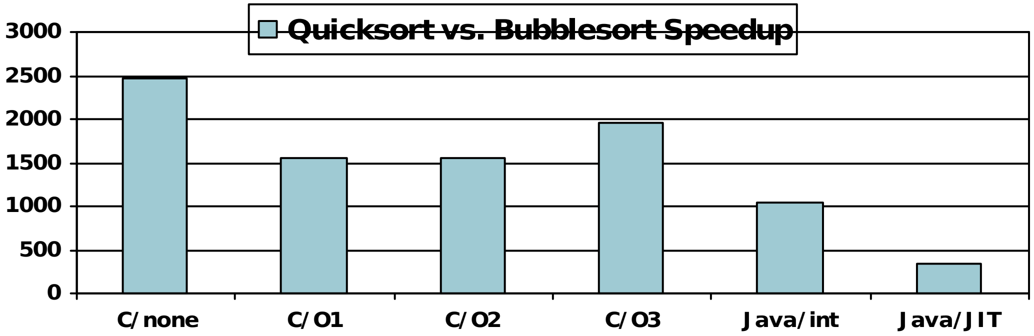
Starting a Java Program



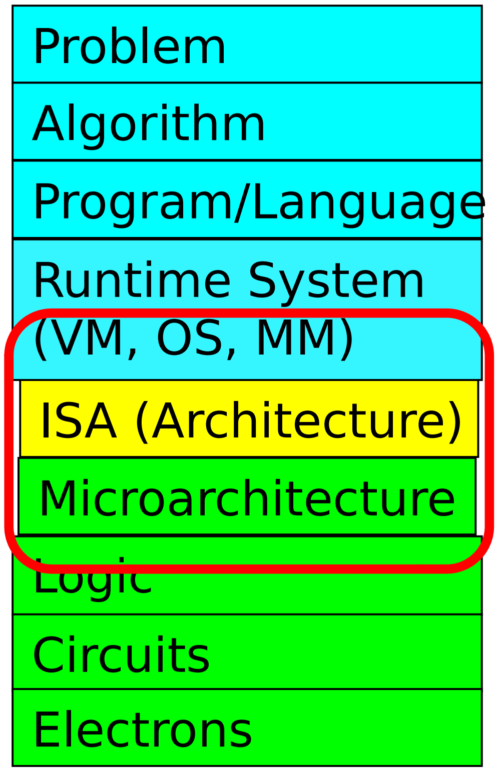
Effect of Language and Algorithm







Levels of Transformation



Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," Proceedings of the IEEE 2001.

Performance Optimization

Must optimize at multiple levels:

- algorithm, data representations, procedures, and loops

Must understand system to optimize performance

- How programs are compiled and executed
- How modern processors + memory systems operate
- How to measure program performance and identify bottlenecks
- How to improve performance without destroying code modularity and generality

References

Chapter 2.9

Chapter 2.12

Chapter 2.14

(Computer Organization and Design: The Hardware/Software Interface by Hennessy/Patterson, 5th edition)