MITx 6.004.2x
Computation Structures 2: Computer Architecture

Help

Course Progress Dates Discussion

Course / 11. Compilers / Lecture Videos (34:57)

Previous ✓ ✓ ✓ ✓ ✓ Next

## LE11.3

Bookmark this page

Calculator

## LE11.3.1 Loop optimizations

0.3333333333333333/1 point (ungraded)

In block structured languages such as C or Java, the scope of a variable declared locally within a block extends only over that block, i.e., the value of the local variable cannot be accessed outside the block. Conceptually, storage is allocated for the variable when the block is entered and deallocated when the block is exited. In many cases, this means the compiler is free to use a register to hold the value of the local variable instead of a memory location.

Consider the following C fragment:

```
int sum = 0;
{ int i;
  for (i = 0; i < 10; i = i+1) sum += i;
}
```

If this loop is compiled such that intermediate results are stored in memory, then the resulting compiled code would look like this:

```
     ST(R31,sum)
     ST(R31,i)
_L7:
     LD(sum,R0)
     LD(i,R1)
     ADD(R0,R1,R0)
     ST(R0,sum)
     ADDC(R1,1,R1)
     ST(R1,i)
     CMPLTC(R1,10,R0)
     BT(R0,_L7)
```

If instead the compilation of this code is optimized by using registers to hold the values of the local variables i and sum, then the compiled code would look like this:

```
     MOVE(R31,R2)         // R2 holds sum
     ST(R2,sum)
     MOVE(R31,R1)         // R1 holds i
_L5:
     ADD(R2,R1,R2)
     ADDC(R1,1,R1)
     CMPLTC(R1,10,R0)
     BT(R0,_L5)
     ST(R2,sum)
```

Define a memory access as any access to memory, i.e., instruction fetch, data read (LD), or data write (ST). Compare the total number of memory accesses generated by executing the optimized loop with the total number of memory access for the unoptimized loop.

**Unoptimized loop:**
**Number of instructions in loop:**

✖ Answer: 8

**Number of data accesses per loop iteration:**

✖ Answer: 4

**Optimized loop:**
**Number of instructions in loop:**

✖ Answer: 4

Calculator

**Number of data accesses per loop iteration:**

✗ Answer: 0

Given that the loop is executed 10 times, what is the total number of memory accesses in the unoptimized loop versus the optimized loop?

**Unoptimized loop: Total memory accesses:**

✗ Answer: 120

**Optimized loop: Total memory accesses:**

✗ Answer: 40

Explanation
The unoptimized loop has 8 instructions in its loop plus 4 data accesses (2 loads and 2 stores). This means that across 10 iterations of the loop, there are a total of 120 memory accesses made.
In the optimized loop, the loop is reduced to 4 instructions with 0 data accesses within the loop. This means that across 10 iterations of this loop, there are a total of 40 memory accesses.

Some optimizing compilers "unroll" small loops to amortize the overhead of each loop iteration over more instructions in the body of the loop. For example, one unrolling of the loop above would be equivalent to rewriting the program as:

```
int sum = 0;
{ int i;
  for (i = 0; i < 10; i = i+2) { sum += i; sum += i+1; }
}
```

The hand-compiled unrolled loop looks like this:

```
     MOVE(R31,R2)          // R2 holds sum
     ST(R2,sum)
     MOVE(R31,R1)          // R1 holds i
_L5:
     ADD(R2,R1,R2)
     ADDC(R1,1,R0)
     ADD(R2,R0,R2)
     ADDC(R1,2,R1)
     CMPLTC(R1,10,R0)
     BT(R0,_L5)
     ST(R2,sum)
```

**Unrolled loop:**
**Number of instructions in loop:**

6 ✓ Answer: 6

**Number of data accesses per loop iteration:**

0 ✓ Answer: 0

**Unrolled loop: Total memory accesses:**

30 ✓ Answer: 30

Calculator

Explanation

The unrolled loop has 6 instructions in its loop plus 0 data accesses. Note that in this loop, since two values of i are handled in each iteration of the loop, i is incremented by 2 each time through the loop, so the total number of loop iterations is 5. This means that the 6*5 = 30 total memory accesses are made when using this unrolled loop. This demonstrates that as a compiler performs a larger number of optimizations, the resulting number of memory accesses decreases.

Submit

Answers are displayed within the problem

## Discussion

**Topic:** 11. Compilers / LE11.3

Hide Discussion

**Add a Post**

Show all posts by recent activity

[STAFF] Misleading
"Define a memory access as any access to memory, i.e., instruction fetch, data read (LD), or data write (ST). " followed by: "Compa... 3

CMP functions memory access
Do CMP functions access the memory twice, once for each operand? 2

Super Confused on Optimized loop
The course in my opinion is not very straight forward sometimes or the definitions I'm not very familiar with yet. Can some one clari... 5

implementing our 32-bit beta risc processor, more than just bsim?
are we implementing our 32-bit beta risc processor so we can run our beta assembly programs, more than just bsim? we implemen... 3

< Previous Next >

## edX

About
Affiliates
edX for Business
Open edX
Careers
News

## Legal

Terms of Service & Honor Code
Privacy Policy
Accessibility Policy
Trademark Policy
Sitemap
Cookie Policy
Your Privacy Choices

Calculator

## Connect

Idea Hub
Contact Us
Help Center
Security
Media Kit

GET IT ON Google play
Download on the App Store

Calculator