CCM (Square and Cube Roots)

Team 25
Keegan Kavanagh - V00900571
Jonathon Squire - V00863838

Contents

Introduction 3
Requirements 4
Design Process 4
Optimization 7
  C-Level Optimizations 7
  Assembly optimizations 12
Conclusion 13
Appendix A 14
  Proof of Correctness and Breadth of Testing 14
  Non-Optimized Assembly 16
  Optimized Assembly 21

Figure 1 - Pseudocode 5
Figure 2 - Floating point code 5
Figure 3 - Fixed point arithmetic code 6
Figure 4 - Timing with just fixed point 7
Figure 5 - Fully optimized without flag yet 9
Figure 6 - Code fully optimized for square root 10
Figure 7 – Optimizations 12
Figure 8 - Os flag timings 13
Figure 9 - Proof of correctness 14
Figure 10 - Cube Root fully optimized 15

Table 1 - Optimizations 12

Introduction

The project we selected for the summer 2021 session of SENG 440 was Convergence Computing Method (CCM) for square root and cube root. The goal of this project was to find the square roots and cube roots of numbers and then to reduce the run-time and memory usage of the CCM on an embedded ARM system. Then we were to compare the times between the square root and cube root.

The solution is written in C and was optimized first in C and then the assembly code was broken down and optimized in the bottleneck areas that were discovered through testing. The optimizations we made were specific for the ARM machine that was used. The ARM machine used is from UVic Engineering which was used with remote access (SSH) because the school was closed due to COVID-19 pandemic.

The contribution to the work was split evenly between the pair of us. Since this was done remotely, collaboration tools were necessary to work adequately on this project. The tools used for this were MobaXTerm, GitHub, Discord and Google Docs. MobaXTerm was used as a terminal to connect to the UVic network. MobaXTerm allowed us to work on our local machines but have all changes saved to the documents on the UVic network. GitHub was used for version control and code management while Discord was used for communication. Lastly, Google docs were used for real-time collaboration to work on this report.

In this report, the requirements are introduced which were created by Dr. Sima. The designs will be discussed and show which one was ultimately chosen and why. Next, the optimizations will be talked about once the design was chosen. The optimizations section will go over which optimizations were chosen and why. Finally, the assembly code will be shown before and after the optimizations to show just how much was changed.

Requirements

The requirements for this project are discussed in this section of the report. This section describes the requirements given to use by Dr. Sima at the start of the project. The Design Process section will talk about how those requirements were implemented and achieved.

The initial requirements are listed below:

- Must be able to take 16-bit values as input
- Must be able to output a 32-bit value
- Must use fixed point arithmetic for calculations
- Square root must take all inputs from 1 to 4 but not including 4
- Cube root must take all inputs from 1 to 8 but not including 8

Design Process

The design for this project changed many times during this project. Many decisions were made in order to meet the requirements listed in the section above. This section describes the decisions that were made to make the calculations correct and to satisfy all the requirements.

The first design was made using floats and pseudo code (Figure 1) for CCM with square roots and cube roots. This design allowed us to ensure CCM was working and created a code base to later optimize. The code is shown below in Figure 2.

Calculation of Square Root – Pseudocode

```
                                        ▷ √M with K bits of precision
f = 1.0
f_sqrt = 1.0
for i = 0 to K − 1 do
    μ = f · (1 + 2^−i) · (1 + 2^−i)     ▷ potential multiplication by A_i^2
    μ_sqrt = f_sqrt · (1 + 2^−i)        ▷ potential multiplication by A_i
    if μ ≤ M then
        f = μ                           ▷ if product is less than M accept iteration,
        f_sqrt = μ_sqrt                 ▷ otherwise reject it (do nothing)
return f_sqrt
```

Figure 1 - Pseudocode

```
void squareRoot(double inputVal) {
    double f = 1.0;
    double f_squareRoot = 1.0;
    int i = 0;
    for (i; i <= 31; i++)
    {
        double precision = f * (1.0 + powerArray[i]) * (1.0 +
powerArray[i]);
        double precision_squareRoot = f_squareRoot * (1.0 +
powerArray[i]);

        if (precision <= inputVal)
        {
            f = precision;
            f_squareRoot = precision_squareRoot;
        }
    }
}
```

Figure 2 - Floating point code

A lookup table created in Python was used for the powerArray. This was our first try at optimizing to make the code run faster.

The next design we tried to implement was to get fixed point arithmetic working. We failed a couple times and had to ask Dr. Sima for some help. After discussing the problem with Dr. Sima, we were able to create a design (Figure 3) that worked with fixed point arithmetic and allowed us to get rid of the lookup table. This design used only shifting and addition for calculating the square root and cube root using CCM. The solution that employed fixed point arithmetic reduced the time taken from about 48 seconds to just over 4 seconds (Figure 4), which is a massive reduction in the time it takes to calculate the cube root 1 million times.

```
int32_t squareRoot(int32_t M) {
    int32_t f = 16384;
    int32_t  f_sqrt = 16384;
    int32_t local_M = M;

    for(int i = 0; i < 16;i++)
        int32_t u = f + (f >> i);
        int32_t u_sqrt =  f_sqrt + (f_sqrt >> i);
        u = u + (u >> i);

        if (u <= local_M)
        {
            f = u;
            f_sqrt = u_sqrt;
        }
    }
    return  f_sqrt;
}
```

Figure 3 - Fixed point arithmetic code

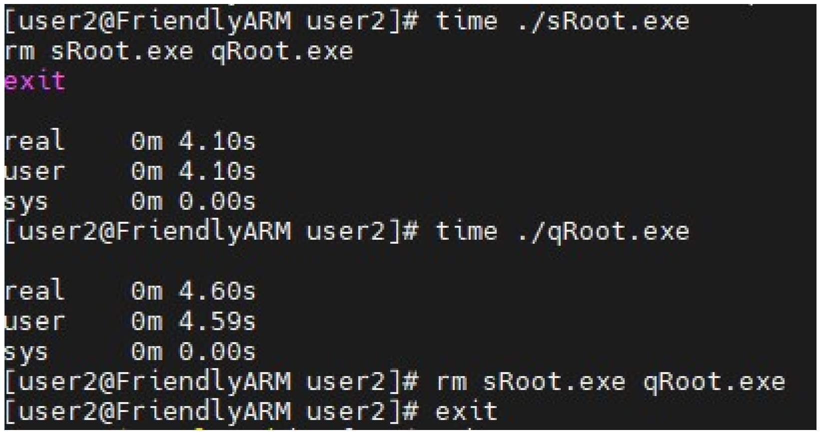

Figure 4 - Timing with just fixed point

This design was the one that became our final design to which we optimize as best we could. The optimization of this design will be talked about in the next section called “Optimization”.

Optimization

The optimizing for this project started after fixed point arithmetic was implemented. To do the optimization we ran the code on the ARM machine using a macro that would compile the code using the provided arm-linux-gcc compiler on the UVic network and transfer and run the resulting executable onto the ARM controller using the Time flag. The time calculated using this method was then used for our comparisons between the square root and cube root, and to compare what optimizations performed better. After the code had been executed and our times were calculated the macro would then delete all the files from both the ARM and temp folder in the UVic server.

C-Level Optimizations

Once the code was converted from using floating point arithmetic to fixed point, we began optimizing by altering the c code itself. All major optimizations can be found side-by-side with their original instructions in Table 1, as well as their resulting effects to efficiency in Figure 7.

To begin, we began by storing all variables locally in int32_t variables with the “register” type. This was to ensure the variables would be stored in the processor’s registers opposed to being stored in memory. Because the cube root function required an additional access to these variables than the square root function, the cube root function benefited more from this change.

Next, we attempted to introduce predicates to both internal and external loops. For the external loop, we applied a simple bit mask to trigger a termination. The internal loop that determined if our “u” variable was less than or equal to the initial input, significantly more challenging. For the predicate we subtracted “u” from the initial input, then masked the output to 2147483648 to determine if the signed bit had been triggered. This worked well for all inputs except when they were equal, resulting in 0 without triggering the signed bit. To combat this edge case, we subtracted 1 from “u” before the process. Unfortunately, most of this work was for naught as most predicates for both functions resulted in slower return times than simply using the comparator. The only exception was in the square root function, the external loop predicate offered approximately 0.2 milliseconds of efficiency per attempt. As such, we reverted all other predicates to their initial comparators.

Both the square root and cube root functions have several hard dependencies that are integral to the execution. It was at this point we saw these dependencies as the true bottleneck of the execution. As we attempted to optimize the pipeline of the code by making small tweaks to instruction locations, as well as adding preprocessing and post processing instructions. The change we made that made the largest difference was breaking down the first iteration with no shifts into its simplest form: shifting by 2 and 3 for the square root and cube root functions respectively. Once we did that, we altered the external loop to begin at 1 opposed to zero and begin with the comparison opposed to the calculations. By doing so, however, we were forced to include a post-process comparator for the final iteration to retain the integrity of the formula.

Due to the nature of the algorithm, we were unable to implement any loop unrolling. By adding a single additional iteration into the algorithm, we saw a significant increase in performance. The square root function especially saw an increase in efficiency. We suspect that is because the square root function has less instructions inside the external for loop and therefore the overhead from the comparator per iteration is more predominant. Increasing the instructions per iteration mitigates this unintentional overhead.

Finally, due to some vigorous testing and slight mishaps, we also reduced the bits of precision from 32 to 16 as we found no difference in output when we tested it with our testing suite. By adjusting the bits of precision, we effectively halved the computation time at the cost of precision (Figure 5). This of course can be reverted should the client prefer accuracy over efficiency.

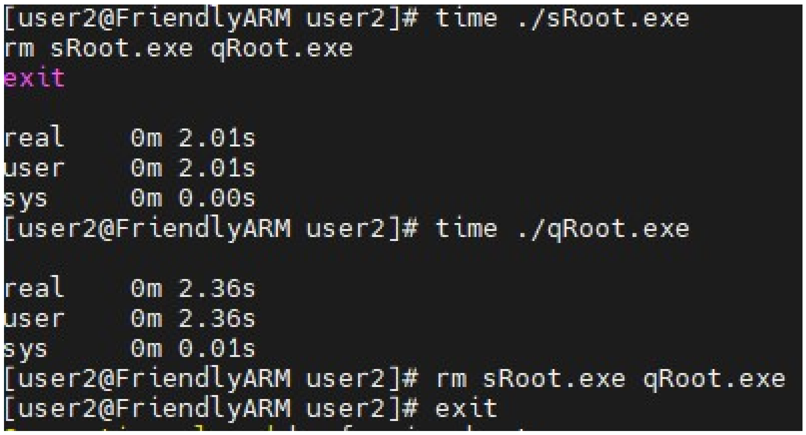

Figure 5 - Fully optimized without flag yet

```c
int32_t squareRoot(int32_t M)
{
    register int32_t f = 16384;
    register int32_t  f_sqrt = 16384;
    register int32_t local_M = M;

    register int i;

    register int32_t u = f<<2; // == 4*f
    register int32_t u_sqrt =  f_sqrt<<1; // 2*f_sqrt

    // for (i=1; i<16; i++)
    for(i=1;!(i&16);i++) //prefered
    {
        if (u <= local_M) // prefered
        // if ((u-local_M-1)&2147483648)
        {
            f = u;
            f_sqrt = u_sqrt;
        }

        u = f + (f >> i);
        u_sqrt =  f_sqrt + (f_sqrt >> i);
        u = u + (u >> i);

        if (u <= local_M) // prefered
        // if ((u-local_M-1)&2147483648)
        {
            f = u;
            f_sqrt = u_sqrt;
        }

        i++;
        u = f + (f >> i);
        u_sqrt =  f_sqrt + (f_sqrt >> i);
        u = u + (u >> i);

    }
    if (u <= local_M) // prefered
    // if ((u-local_M-1)&2147483648)
    {
        f = u;
        f_sqrt = u_sqrt;
    }

    return  f_sqrt;
}
```

Figure 6 - Code fully optimized for square root

Method	Before Optimization	After Optimization
Optimizing for loop	for (i^=i; i<32; i++)	for(i^=i;!(i&32);i++)
Variable Data Types	Int32_t f, f_sqrt, local_M, u;	Register int32_t f, f_sqrt, local_M, u;
Bit Masking Predicate for comparing u to local_M	if (u <= local_M)	if((u-local_M-1)&2147483648)

Pipelining 1: instruction reordering	u = f + (f >> i);
	u = u + (u >> i);
	u_sqrt = f_sqrt + (f_sqrt >> i);
		u = f + (f >> i);
		u_sqrt = f_sqrt + (f_sqrt >> i);
		u = u + (u >> i);

Pipelining:Preprocess and optimize first iteration	for(i^=i;!(i&32);i++){ ...	register int32_t u = f<<2; register int32_t u_sqrt = f_sqrt<<1;
		
		for(i=1;!(i&32);i++) {...

Loop unrolling	...for (i=1; i<16; i++)
	{
	if (u <= local_M)
	{
	f = u;
	f_sqrt = u_sqrt;
	}
	
	u = f + (f >> i);
	u_sqrt = f_sqrt + (f_sqrt >> i);
	u = u + (u >> i);
	u = u + (u >> i);
	} ...
		...for (i=1; i<16; i++)
		{
		if (u <= local_M)
		{
		f = u;
		f_sqrt = u_sqrt;
		}
		
		u = f + (f >> i);
		u_sqrt = f_sqrt + (f_sqrt >> i);
		u = u + (u >> i);
		u = u + (u >> i);
		if (u <= local_M)
		{
		f = u;
		f_sqrt = u_sqrt;
		}
		
		i++;
		u = f + (f >> i);
		u_sqrt = f_sqrt + (f_sqrt >> i);
		u = u + (u >> i);
		u = u + (u >> i);
		} ...

Remove redundant move instructions	add r2, r3, r3, lsr #1 movge r3, r2	add r3, r3, r3

Table 1 - Optimizations

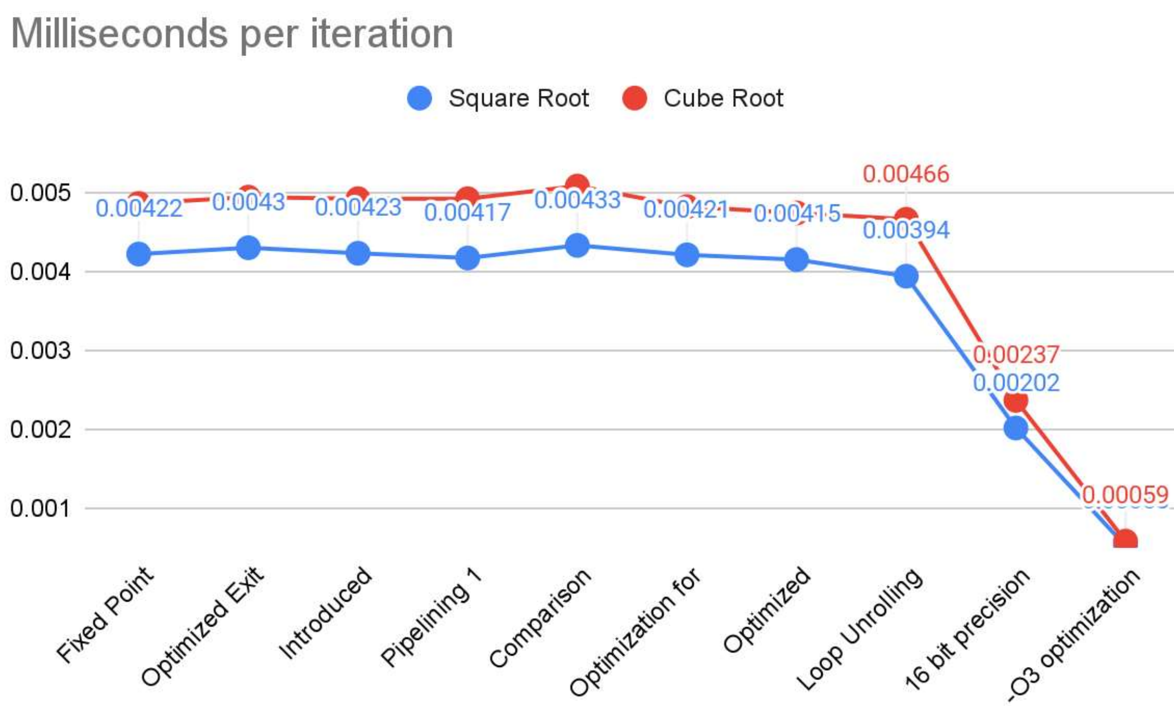

Figure 7 – Optimizations

Assembly optimizations

Once we optimized the c code to the best of our abilities, we attempted to optimize the assembly itself. When we ran the compiler as such “arm-linux-gcc -S sRoot.c”, we received the assembly code outlined in Appendix A. We could see there was an excessive amount of load and store instructions, however we could not figure out how to best optimize them. In response, we compiled the c code again with the -Os flag to receive the second assembly set outlined in Appendix A which decreased the execution time by about 15%.

Upon further inspection of the -Os optimized assembly, we discovered several redundant move instructions, outlined in table 1**.** We estimated there to be approximately 31 instances of these redundant move instructions in the -Os optimized code. Averaging 5-11 cycles per move, these additional instructions introduce approximately 155 - 341 unnecessary cycles per iteration.

```
time ./qRoot.exe
rm sRoot.exe qRoot.exe
exit

real    0m 0.53s
user    0m 0.52s
sys     0m 0.01s
[user3@FriendlyARM user3]# time ./qRoot.exe

real    0m 0.59s
user    0m 0.58s
sys     0m 0.00s
[user3@FriendlyARM user3]# rm sRoot.exe qRoot.exe
```

Figure 8 - Os flag timings

Conclusion

Using the various techniques learned in SENG 440, the run-time to calculate the square root and cube root using CCM was greatly reduced. The design process involved getting the program to correctly calculate the roots and then once that was accomplished the program was modified repeatedly until we were happy with the optimizations. This required a lot of C code changes and adding additional flags to the compiling.

At the very beginning the cube root was running at 48.05 seconds, this was without fix-point arithmetic or any other optimizations. Once all the optimizations were made, cube root took only 0.59 seconds to run. This was an increase of 8144% in performance. The performance difference between the square root and cube root function ended up being smaller than thought. It was only a couple instructions in C different, but it ended up being 0.06 seconds difference in execution time for 1 million runs of the functions. Therefore, cube root takes 60 nanoseconds longer to calculate than square root when fully optimized.

With more time, more optimizations would be tried with the hopes of increasing performance more. Something like removing the moves in the assembly as mentioned at the end of the optimization section. Ultimately, this project was a success, and the code was optimized well, and a lot was learned.

Appendix A

Proof of Correctness and Breadth of Testing

```
[jonathonsquire@seng440 src]$ ./sRoot.exe
The Square Root of 1.000 is 1.000000
The Square Root of 1.750 is 1.322937
The Square Root of 2.000 is 1.414246
The Square Root of 2.250 is 1.500000
The Square Root of 3.000 is 1.731995
The Square Root of 3.500 is 1.870728
The Square Root of 4.000 is 1.999939

[jonathonsquire@seng440 src]$ ./qRoot.exe
The Square Root of 1.000 is 1.000000
The Square Root of 2.750 is 1.400940
The Square Root of 3.000 is 1.442200
The Square Root of 4.250 is 1.619751
The Square Root of 5.000 is 1.709900
The Square Root of 6.500 is 1.866089
The Square Root of 7.999 is 1.999512
```

Figure 9 - Proof of correctness

```c
int32_t cubeRoot(int32_t M)
{
    register int32_t f = 16384;
    register int32_t  f_sqrt = 16384;
    register int32_t local_M = M;

    register int i;

    register int32_t u = f<<3; // == f*8
    register int32_t u_sqrt =  f_sqrt<<1; // == u_sqrt*2 == u_sqrt + (u_sqrt>>0)

    for (i=1; i<16; i++) // prefered
    // for(i=1;!(i&16);i++)
    {
        if (u <= local_M) //prefered
        // if ((u-local_M-1)&2147483648)
        {
            f = u;
            f_sqrt = u_sqrt;
        }

        u = f + (f >> i);
        u_sqrt =  f_sqrt + (f_sqrt >> i);
        u = u + (u >> i);
        u = u + (u >> i);

        if (u <= local_M) //prefered
        // if ((u-local_M-1)&2147483648)
        {
            f = u;
            f_sqrt = u_sqrt;
        }

        i++;
        u = f + (f >> i);
        u_sqrt =  f_sqrt + (f_sqrt >> i);
        u = u + (u >> i);
        u = u + (u >> i);
    }

    if (u <= local_M) //prefered
    // if ((u-local_M-1)&2147483648)
    {
        f = u;
        f_sqrt = u_sqrt;
    }
    return  f_sqrt;
}
```

Figure 10 - Cube Root fully optimized

Non-Optimized Assembly

```
	.arch armv4t
	.fpu softvfp
	.eabi_attribute 20, 1
	.eabi_attribute 21, 1
	.eabi_attribute 23, 3
	.eabi_attribute 24, 1
	.eabi_attribute 25, 1
	.eabi_attribute 26, 2
	.eabi_attribute 30, 6
	.eabi_attribute 18, 4
	.file	"sRoot.c"
	.global testingResults
	.bss
	.align	2
	.type	testingResults, %object
	.size	testingResults, 4
testingResults:
	.space 4
	.text
	.align	2
	.global squareRoot
	.type	squareRoot, %function
squareRoot:
	@ Function supports interworking.
	@ args = 0, pretend = 0, frame = 32
	@ frame_needed = 1, uses_anonymous_args = 0
	@ link register save eliminated.
	str	fp, [sp, #-4]!
	add	fp, sp, #0
	sub	sp, sp, #36
	str	r0, [fp, #-8]
	mov	r1, #16384
	str	r1, [fp, #-32]
	mov	r2, #16384
	str	r2, [fp, #-28]
	ldr	r3, [fp, #-8]
	str	r3, [fp, #-24]
	ldr	r1, [fp, #-32]
	mov	r1, r1, asl #2
	str	r1, [fp, #-16]
	ldr	r2, [fp, #-28]
	mov	r2, r2, asl #1
	str	r2, [fp, #-12]
	mov	r3, #1
	str	r3, [fp, #-20]
	b	.L2
```

```
.L5:
        ldr     r1, [fp, #-16]
        ldr     r2, [fp, #-24]
        cmp     r1, r2
        bgt     .L3
        ldr     r3, [fp, #-16]
        str     r3, [fp, #-32]
        ldr     r1, [fp, #-12]
        str     r1, [fp, #-28]
.L3:
        ldr     r2, [fp, #-32]
        ldr     r1, [fp, #-20]
        mov     r3, r2, asr r1
        ldr     r2, [fp, #-32]
        add     r3, r3, r2
        str     r3, [fp, #-16]
        ldr     r1, [fp, #-28]
        ldr     r2, [fp, #-20]
        mov     r3, r1, asr r2
        ldr     r1, [fp, #-28]
        add     r3, r3, r1
        str     r3, [fp, #-12]
        ldr     r2, [fp, #-16]
        ldr     r1, [fp, #-20]
        mov     r3, r2, asr r1
        ldr     r2, [fp, #-16]
        add     r2, r2, r3
        str     r2, [fp, #-16]
        ldr     r3, [fp, #-16]
        ldr     r1, [fp, #-24]
        cmp     r3, r1
        bgt     .L4
        ldr     r2, [fp, #-16]
        str     r2, [fp, #-32]
        ldr     r3, [fp, #-12]
        str     r3, [fp, #-28]
.L4:
        ldr     r1, [fp, #-20]
        add     r1, r1, #1
        str     r1, [fp, #-20]
        ldr     r2, [fp, #-32]
        ldr     r1, [fp, #-20]
        mov     r3, r2, asr r1
        ldr     r2, [fp, #-32]
        add     r3, r3, r2
        str     r3, [fp, #-16]
        ldr     r1, [fp, #-28]
        ldr     r2, [fp, #-20]
        mov     r3, r1, asr r2
        ldr     r1, [fp, #-28]
        add     r3, r3, r1
```

```
        str     r3, [fp, #-12]
        ldr     r2, [fp, #-16]
        ldr     r1, [fp, #-20]
        mov     r3, r2, asr r1
        ldr     r2, [fp, #-16]
        add     r2, r2, r3
        str     r2, [fp, #-16]
        ldr     r3, [fp, #-20]
        add     r3, r3, #1
        str     r3, [fp, #-20]
.L2:
        ldr     r1, [fp, #-20]
        and     r3, r1, #16
        cmp     r3, #0
        beq     .L5
        ldr     r2, [fp, #-16]
        ldr     r3, [fp, #-24]
        cmp     r2, r3
        bgt     .L6
        ldr     r1, [fp, #-16]
        str     r1, [fp, #-32]
        ldr     r2, [fp, #-12]
        str     r2, [fp, #-28]
.L6:
        ldr     r3, [fp, #-28]
        mov     r0, r3
        add     sp, fp, #0
        ldmfd   sp!, {fp}
        bx      lr
        .size   squareRoot, .-squareRoot
        .section        .rodata
        .align  2
        .type   C.0.2136, %object
        .size   C.0.2136, 28
C.0.2136:
        .word   1065353216
        .word   1071644672
        .word   1073741824
        .word   1074790400
        .word   1077936128
        .word   1080033280
        .word   1082130428
        .global __aeabi_fmul
        .global __aeabi_f2iz
        .global __aeabi_i2f
        .global __aeabi_fdiv
        .global __aeabi_f2d
        .align  2
.LC0:
        .ascii  "The Square Root of %1.3f is %f\012\000"
        .text
```

```
	.align	2
	.global	main
	.type	main, %function
main:
	@ Function supports interworking.
	@ args = 0, pretend = 0, frame = 64
	@ frame_needed = 1, uses_anonymous_args = 0
	stmfd	sp!, {r4, r5, r6, fp, lr}
	add	fp, sp, #16
	sub	sp, sp, #76
	str	r0, [fp, #-80]
	str	r1, [fp, #-84]
	ldr	r3, .L16
	sub	lr, fp, #72
	mov	ip, r3
	ldmia	ip!, {r0, r1, r2, r3}
	stmia	lr!, {r0, r1, r2, r3}
	ldmia	ip, {r0, r1, r2}
	stmia	lr, {r0, r1, r2}
	mov	r3, #0
	str	r3, [fp, #-44]
	ldr	r3, .L16+4
	ldr	r3, [r3, #0]
	cmp	r3, #0
	beq	.L9
	b	.L10
.L11:
	ldr	r3, [fp, #-44]
	mvn	r2, #51
	mov	r3, r3, asl #2
	sub	r1, fp, #20
	add	r3, r1, r3
	add	r3, r3, r2
	ldr	r3, [r3, #0]	@ float
	str	r3, [fp, #-40]	@ float
	ldr	r3, [fp, #-40]	@ float
	ldr	r2, .L16+8	@ float
	mov	r0, r3
	mov	r1, r2
	bl	__aeabi_fmul
	mov	r3, r0
	mov	r0, r3
	bl	__aeabi_f2iz
	mov	r3, r0
	str	r3, [fp, #-36]
	ldr	r0, [fp, #-36]
	bl	squareRoot
	mov	r3, r0
	str	r3, [fp, #-32]
	ldr	r0, [fp, #-32]
	bl	__aeabi_i2f
```

```
        mov     r3, r0
        ldr     r2, .L16+8      @ float
        mov     r0, r3
        mov     r1, r2
        bl      __aeabi_fdiv
        mov     r3, r0
        str     r3, [fp, #-28]  @ float
        ldr     r0, [fp, #-40]  @ float
        bl      __aeabi_f2d
        mov     r5, r0
        mov     r6, r1
        ldr     r0, [fp, #-28]  @ float
        bl      __aeabi_f2d
        mov     r3, r0
        mov     r4, r1
        stmia   sp, {r3-r4}
        ldr     r0, .L16+12
        mov     r2, r5
        mov     r3, r6
        bl      printf
        ldr     r3, [fp, #-44]
        add     r3, r3, #1
        str     r3, [fp, #-44]
.L10:
        ldr     r3, [fp, #-44]
        cmp     r3, #6
        ble     .L11
        b       .L12
.L9:
        mov     r3, #28672
        str     r3, [fp, #-24]
        b       .L13
.L14:
        ldr     r0, [fp, #-24]
        bl      squareRoot
        ldr     r3, [fp, #-44]
        add     r3, r3, #1
        str     r3, [fp, #-44]
.L13:
        ldr     r2, [fp, #-44]
        mov     r3, #999424
        add     r3, r3, #572
        add     r3, r3, #3
        cmp     r2, r3
        ble     .L14
.L12:
        mov     r0, #10
        bl      putchar
        mov     r3, #0
        mov     r0, r3
        sub     sp, fp, #16
```

```
        ldmfd   sp!, {r4, r5, r6, fp, lr}
        bx      lr
.L17:
        .align  2
.L16:
        .word   C.0.2136
        .word   testingResults
        .word   1182793728
        .word   .LC0
        .size   main, .-main
        .ident  "GCC: (Sourcery G++ Lite 2008q3-72) 4.3.2"
        .section        .note.GNU-stack,"",%progbits
```

Optimized Assembly

```
        .arch armv4t
        .fpu softvfp
        .eabi_attribute 20, 1
        .eabi_attribute 21, 1
        .eabi_attribute 23, 3
        .eabi_attribute 24, 1
        .eabi_attribute 25, 1
        .eabi_attribute 26, 2
        .eabi_attribute 30, 2
        .eabi_attribute 18, 4
        .file   "sRoot.c"
        .text
        .align  2
        .global squareRoot
        .type   squareRoot, %function
squareRoot:
        @ Function supports interworking.
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        mov     r3, #65536
        sub     r3, r3, #1
        cmp     r0, r3
        mov     ip, r0
        movle   r0, #16384
        movgt   r0, #65536
        add     r2, r0, r0, lsr #1
        add     r2, r2, r2, lsr #1
        movle   r3, #16384
        movgt   r3, #32768
        cmp     ip, r2
        movge r0, r2
        add     r1, r0, r0, lsr #2
        add     r1, r1, r1, lsr #2
```

```
add     r2, r3, r3, lsr #1
movge r3, r2
cmp     ip, r1
movge r0, r1
add     r2, r0, r0, lsr #3
add     r2, r2, r2, lsr #3
add     r1, r3, r3, lsr #2
movge r3, r1
cmp     ip, r2
movge r0, r2
add     r1, r0, r0, lsr #4
add     r1, r1, r1, lsr #4
add     r2, r3, r3, lsr #3
movge r3, r2
cmp     ip, r1
movge r0, r1
add     r2, r0, r0, lsr #5
add     r2, r2, r2, lsr #5
add     r1, r3, r3, lsr #4
movge r3, r1
cmp     ip, r2
movge r0, r2
add     r1, r0, r0, lsr #6
add     r1, r1, r1, lsr #6
add     r2, r3, r3, lsr #5
movge r3, r2
cmp     ip, r1
movge r0, r1
add     r2, r0, r0, lsr #7
add     r2, r2, r2, lsr #7
add     r1, r3, r3, lsr #6
movge r3, r1
cmp     ip, r2
movge r0, r2
add     r1, r0, r0, lsr #8
add     r1, r1, r1, asr #8
add     r2, r3, r3, lsr #7
movge r3, r2
cmp     ip, r1
movge r0, r1
add     r2, r0, r0, asr #9
add     r2, r2, r2, asr #9
add     r1, r3, r3, lsr #8
movge r3, r1
cmp     ip, r2
movge r0, r2
add     r1, r0, r0, asr #10
add     r1, r1, r1, asr #10
add     r2, r3, r3, lsr #9
movge r3, r2
cmp     ip, r1
```

```
        movge r0, r1
        add   r2, r0, r0, asr #11
        add   r2, r2, r2, asr #11
        add   r1, r3, r3, lsr #10
        movge r3, r1
        cmp   ip, r2
        movge r0, r2
        add   r1, r0, r0, asr #12
        add   r1, r1, r1, asr #12
        add   r2, r3, r3, lsr #11
        movge r3, r2
        cmp   ip, r1
        movge r0, r1
        add   r2, r0, r0, asr #13
        add   r2, r2, r2, asr #13
        add   r1, r3, r3, lsr #12
        movge r3, r1
        cmp   ip, r2
        movge r0, r2
        add   r1, r0, r0, asr #14
        add   r1, r1, r1, asr #14
        add   r2, r3, r3, lsr #13
        movge r3, r2
        cmp   ip, r1
        movge r0, r1
        add   r2, r0, r0, asr #15
        add   r2, r2, r2, asr #15
        add   r1, r3, r3, lsr #14
        movge r3, r1
        cmp   ip, r2
        movge r0, r2
        add   r1, r3, r3, lsr #15
        add   r0, r0, r0, asr #16
        movge r3, r1
        add   r0, r0, r0, asr #16
        cmp   ip, r0
        add   r2, r3, r3, lsr #16
        movlt r0, r3
        movge r0, r2
        bx    lr
        .size squareRoot, .-squareRoot
        .global __aeabi_fmul
        .global __aeabi_f2iz
        .global __aeabi_f2d
        .global __aeabi_i2f
        .align 2
        .global main
        .type main, %function
main:
        @ Function supports interworking.
        @ args = 0, pretend = 0, frame = 32
```

```
	@ frame_needed = 0, uses_anonymous_args = 0
	stmfd	sp!, {r4, r5, r6, r7, r8, sl, lr}
	ldr	ip, .L143
	sub	sp, sp, #44
	mov	lr, ip
	ldmia	lr!, {r0, r1, r2, r3}
	add	r8, sp, #12
	mov	ip, r8
	stmia	ip!, {r0, r1, r2, r3}
	ldr	r3, .L143+4
	ldr	r3, [r3, #0]
	ldmia	lr, {r0, r1, r2}
	cmp	r3, #0
	stmia	ip, {r0, r1, r2}
	moveq	r2, #999424
	addeq	r2, r2, #576
	beq	.L137
	mov	sl, #65536
	sub	sl, sl, #1
	mov	r7, #0
.L135:
	ldr	r5, [r8, r7]	@ float
	mov	r1, #1174405120
	add	r1, r1, #8388608
	mov	r0, r5
	bl	__aeabi_fmul
	bl	__aeabi_f2iz
	cmp	r0, sl
	movle	r3, #16384
	movgt	r3, #65536
	add	r2, r3, r3, lsr #1
	add	r2, r2, r2, lsr #1
	movle	r4, #16384
	movgt	r4, #32768
	cmp	r0, r2
	movge	r3, r2
	add	r1, r3, r3, lsr #2
	add	r1, r1, r1, lsr #2
	add	r2, r4, r4, lsr #1
	movge	r4, r2
	cmp	r0, r1
	movge	r3, r1
	add	r2, r3, r3, lsr #3
	add	r2, r2, r2, lsr #3
	add	r1, r4, r4, lsr #2
	movge	r4, r1
	cmp	r0, r2
	movge	r3, r2
	add	r1, r3, r3, lsr #4
	add	r1, r1, r1, lsr #4
	add	r2, r4, r4, lsr #3
```

```
movge r4, r2
cmp     r0, r1
movge r3, r1
add     r2, r3, r3, lsr #5
add     r2, r2, r2, lsr #5
add     r1, r4, r4, lsr #4
movge r4, r1
cmp     r0, r2
movge r3, r2
add     r1, r3, r3, lsr #6
add     r1, r1, r1, lsr #6
add     r2, r4, r4, lsr #5
movge r4, r2
cmp     r0, r1
movge r3, r1
add     r2, r3, r3, lsr #7
add     r2, r2, r2, lsr #7
add     r1, r4, r4, lsr #6
movge r4, r1
cmp     r0, r2
movge r3, r2
add     r1, r3, r3, lsr #8
add     r1, r1, r1, asr #8
add     r2, r4, r4, lsr #7
movge r4, r2
cmp     r0, r1
movge r3, r1
add     r2, r3, r3, asr #9
add     r2, r2, r2, asr #9
add     r1, r4, r4, lsr #8
movge r4, r1
cmp     r0, r2
movge r3, r2
add     r1, r3, r3, asr #10
add     r1, r1, r1, asr #10
add     r2, r4, r4, lsr #9
movge r4, r2
cmp     r0, r1
movge r3, r1
add     r2, r3, r3, asr #11
add     r2, r2, r2, asr #11
add     r1, r4, r4, lsr #10
movge r4, r1
cmp     r0, r2
movge r3, r2
add     r1, r3, r3, asr #12
add     r1, r1, r1, asr #12
add     r2, r4, r4, lsr #11
movge r4, r2
cmp     r0, r1
movge r3, r1
```

```
        add     r2, r3, r3, asr #13
        add     r2, r2, r2, asr #13
        add     r1, r4, r4, lsr #12
        movge   r4, r1
        cmp     r0, r2
        movge   r3, r2
        add     r1, r3, r3, asr #14
        add     r1, r1, r1, asr #14
        add     r2, r4, r4, lsr #13
        movge   r4, r2
        cmp     r0, r1
        movge   r3, r1
        add     r2, r3, r3, asr #15
        add     r2, r2, r2, asr #15
        add     r1, r4, r4, lsr #14
        movge   r4, r1
        cmp     r0, r2
        movge   r3, r2
        add     r1, r4, r4, lsr #15
        add     r3, r3, r3, asr #16
        add     r3, r3, r3, asr #16
        movge   r4, r1
        add     r2, r4, r4, lsr #16
        cmp     r0, r3
        mov     r0, r5
        movge   r4, r2
        bl      __aeabi_f2d
        mov     r5, r0
        mov     r0, r4
        mov     r6, r1
        bl      __aeabi_i2f
        mov     r1, #947912704
        bl      __aeabi_fmul
        bl      __aeabi_f2d
        add     r7, r7, #4
        stmia   sp, {r0-r1}
        mov     r2, r5
        mov     r3, r6
        ldr     r0, .L143+8
        bl      printf
        cmp     r7, #28
        bne     .L135
.L136:
        mov     r0, #10
        bl      putchar
        mov     r0, #0
        add     sp, sp, #44
        ldmfd   sp!, {r4, r5, r6, r7, r8, sl, lr}
        bx      lr
.L137:
        add     r3, r3, #1
```

```
        cmp     r3, r2
        beq     .L136
        add     r3, r3, #1
        cmp     r3, r2
        bne     .L137
        b       .L136
.L144:
        .align  2
.L143:
        .word   .LANCHOR0
        .word   .LANCHOR1
        .word   .LC0
        .size   main, .-main
        .global testingResults
        .section        .rodata
        .align  2
.LANCHOR0 = . + 0
        .type   C.11.2298, %object
        .size   C.11.2298, 28
C.11.2298:
        .word   1065353216
        .word   1071644672
        .word   1073741824
        .word   1074790400
        .word   1077936128
        .word   1080033280
        .word   1082130428
        .section        .rodata.str1.4,"aMS",%progbits,1
        .align  2
.LC0:
        .ascii  "The Square Root of %1.3f is %f\012\000"
        .bss
        .align  2
.LANCHOR1 = . + 0
        .type   testingResults, %object
        .size   testingResults, 4
testingResults:
        .space 4
        .ident  "GCC: (Sourcery G++ Lite 2008q3-72) 4.3.2"
        .section        .note.GNU-stack,"",%progbits
```