Optimizing Integer Division: A Study on the Unsigned Newton-Raphson Method

Niping Yi

Abstract

Current instruction set architectures (ISAs) sometimes lack support for integer division, necessitating the synthesis of these operations in software. This paper investigates functional iteration methods, particularly the improved Unsigned Newton-Raphson (UNR) recurrence method, to enhance the efficiency of software-based integer division. We provide an in-depth analysis of existing methods, including supplementary explanations for the reference work by Rodeheffer et al. (2008). The paper presents a theoretical demonstration of the UNR method for (W = 32) and extends this approach to unsigned integer division for (W = 64).

Keywords: Integer Division, Functional Iteration, Newton-Raphson Method

1. Introduction

In contemporary computing environments, many instruction set architectures (ISAs) lack native support for integer division, which necessitates the implementation of these operations through software. In this paper, we leverage the properties of different ISAs to explore functional iteration methods.

In this paper, we primarily address three key aspects. First, we provide supplementary explanations for the reference[1]. Second, we theoretically demonstrate the improved UNR recurrence method for $W = 32$. Finally, we extend this method to unsigned integer division for $W = 64$.

We adopt the same notation as in [1]

Email address: yiniping123@gmail.com (Niping Yi)

Symbol	Definition
W	Number of bits in a machine word (typically $W = 32$)
U	Set of unsigned integers: $\{0, \ldots, 2^W - 1\}$
U^+	Set of strictly positive unsigned integers: $\{1, \ldots, 2^W - 1\}$
x	Dividend, where $x \in U$
y	Divisor, where $y \in U^+$
q	Quotient, computed as $q = \left\lfloor \frac{x}{y} \right\rfloor$
r	Remainder, computed as $r = x - q \cdot y$
$\text{inv}(y)$	$\max\{z \in U^+ \mid y \cdot z < 2^W\}$
LBy	$\{z \in U^+ \mid 1 \leq y \cdot z < 2^W\}$
$DLBy$	$\{z \in U^+ \mid 2^{W-1} \leq y \cdot z < 2^W\}$
$TYLBy$	$\{z \in U^+ \mid 2^W - 2 \cdot y \leq y \cdot z < 2^W\}$

Table 1: Notation and Definitions

2. Related Work

Section 2 describes the methods based on functional iteration as presented in reference[1]. We also provide explanations and clarifications for some details that were not mentioned in the reference.

Recall that the steps to find $\frac{1}{y}$ using the Newton-Raphson method are as follows:

1. **Initial Guess**: Start with an initial guess x_0 for $\frac{1}{y}$.
2. **Iteration Formula**: Use the iteration formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

   where $f(x) = \frac{1}{x} - y$. Apply this formula iteratively.
3. **Convergence**: Repeat the iteration until the value converges to the desired level of accuracy.

Now, we present the Newton-Raphson method for calculating $\frac{x}{y}$ as described in Reference[1]. Then, we will understand how it modifies the above Newton-Raphson method to obtain this algorithm.

1. Start with an initial guess $z_0 \in LB_y$

2. Use the iteration formula $z_{n+1} = z_n - \left\lfloor \frac{g(z_n)}{g'(z_n)} \right\rfloor$, where $g(z) = \frac{1}{z} - \frac{y}{2^W}$, which can be expressed as

$$z_{i+1} = z_i + \left\lfloor \frac{z_i \cdot (2^W - y \cdot z_i)}{2^W} \right\rfloor, \tag{1}$$

   to refine the guess. Repeat the iteration until the value converges to the desired level of accuracy.
3. Estimate the quotient q and the remainder r as follows:

$$q = \left\lfloor \frac{x}{y} \right\rfloor$$

   and

$$r = x - q \cdot y.$$

4. q refinement. If the remainder r is greater than or equal to y, r is reduced by y and q is incremented by 1. This process is repeated at most twice.

Theorem 1. *[1] Given $y \in U^+$, $z_i \in LB_y$, and the UNR recurrence*

$$z_{i+1} = z_i + \left\lfloor \frac{z_i \cdot (2^W - y \cdot z_i)}{2^W} \right\rfloor$$

it follows that $z_i < z_{i+1}$ and $z_{i+1} \in LB_y$.

From Theorem 1, we know that when we choose the initial value $z_0 \in LB_y$ and use the equation 1, we obtain a monotonically increasing and bounded sequence $z_0, z_1, z_2, \ldots$.

Theorem 2. *[1] Given $z_0 \in DLB_y$, and computing z_1, z_2, and so on using the UNR recurrence, it follows that the final value $z_n \in TYLB_y$.*

Theorem 3. *[1] Given $x \in TYLB$, $x \in U$, and $q = \left\lfloor \frac{x}{y} \right\rfloor$, it follows that*

$$q - 2 \leq \left\lfloor \frac{x \cdot z}{2^W} \right\rfloor < q.$$

Since a monotonic and bounded sequence converges, we know that the non-decreasing sequence $z_0, z_1, z_2, \ldots$ will eventually reach a final value. From Theorem 2, we know that the final value $z_n \in TYLB_y$. From Theorem 3, it

follows that at most two test subtractions of y from $\hat{r}$ are required to obtain the final quotient and remainder.

Specifically, this paper[1] sets $z_0 = 2^{W-k-1}$, where $k = \lfloor \log_2(y) \rfloor$. It is easy to check that $z_0 \in DLB_y$. According to Reference[1], five iterations are always sufficient for $W = 32$. Below is the pseudocode for this algorithm.

Algorithm 1: UNR recurrence method for W=32.

```
// step I:  decent start
z = lsl(1, clz(y))

// step II:  z recurrence, 5 iterations
Repeat 5 times:
    z = z + umulh(z, mul(-y, z))

// step III:  q estimate
q = umulh(x, z)
r = x - mul(y, q)

// step IV:  q refinement
If r >= y:
    r = r - y
    q = q + 1
    If r >= y:
        r = r - y
        q = q + 1
```

3. UNR recurrence method for W=32

In this chapter, we consider architectures such as AMD architecture, which have added single-precision floating-point multiplication and reciprocal operations. In this section, we utilize these additional operations to improve the algorithm, thereby obtaining better initial values and reducing the number of iterations.

First, we present the pseudocode as follows:

Algorithm 2: UNR recurrence method for W=32.

```
// step I:  decent start
    constant_part = bit_cast<float>(0x4F7FFFFE)
    z = (constant_part * v_rcp_f32(y as float)) as unsigned
```

```
// step II:  One round of UNR (Unsigned integer Newton-Raphson)
z = z + umulh(z, -y * z)

// step III:  q estimate
q = umulh(x, z)
r = x - q * y

// step IV:  q estimate
If r >= y:
    q = q + 1
    r = r - y
If r >= y:
    q = q + 1
    r = r - y
```

Let's examine how the initial value is chosen. We take a floating-point number close to 2^{32} and convert y to a floating-point number. Dividing these gives us an initial value. For this initial value, a single iteration is sufficient to obtain a value that meets the conditions. Below, we provide a theoretical proof of the specific content. Through rigorous argumentation, we aim to extend this method to the case where $w = 64$.

The floating-point representation of 2^{32} can be obtained as `0x4F800000`. However, when using this value, due to the sparsity of floating-point numbers at higher magnitudes, the function `v_rcp_f32` may yield a result that is either greater than or less than the actual value of y. This discrepancy is particularly problematic when `v_rcp_f32` is less than y, as it can cause issues in the computation of `umulh(z, -y * z)`. Therefore, we need to choose a floating-point number that is as close as possible to 2^{32} while avoiding the aforementioned issues.

Theorem 4. *In Algorithm 2, we have $z_0 \in LB_y$.*

Proof. When $y \geq 2^9$, let k be the difference between y and its floating-point representation. When $k < 0$, it is easy to obtain:

$$1 \leq \frac{2^{32} - 2^9}{y - k} \cdot y < 2^{32}.$$

When $k \geq 0$, due to the format of floating-point numbers, we have:

$$\frac{k}{y} < \frac{1}{2^{24}}.$$

Therefore,

$$\frac{2^{32} \cdot k}{y} < 2^9,$$

which implies

$$2^{32} - \frac{2^{32} \cdot (y-k)}{y} < 2^9,$$

and thus

$$2^{32} - 2^9 < \frac{2^{32} \cdot (y-k)}{y}.$$

Therefore,

$$\frac{2^{32} - 2^9}{y-k} \cdot y < 2^{32}.$$

When $y < 2^9$, let k' be the difference between $\frac{2^{32}-2^9}{y}$ and its floating-point representation. When $k' < 0$, it is easy to obtain:

$$(\frac{2^{32} - 2^9}{y} + k') \cdot y < 2^{32}.$$

When $k' \geq 0$, due to the format of floating-point numbers, we have:

$$k' \cdot y < 2^9.$$

Therefore,

$$(\frac{2^{32} - 2^9}{y} + k') \cdot y < 2^{32}.$$

Then we have $z_0 \in LB_y$.

□

Theorem 5. *Algorithm 2 requires only one round of UNR (Unsigned integer Newton-Raphson).*

Proof. Let $\delta = 2^W - y \cdot z_0$. It is easy to prove that

$$\delta < 2^{10}.$$

Given

$$z_1 = z_0 + \left\lfloor \frac{z_0 \cdot (2^W - y \cdot z_0)}{2^W} \right\rfloor$$

We have:

$$z_1 > z_0 + \frac{z_0 \cdot (2^W - y \cdot z_0)}{2^W} - 1$$

Therefore,

$$\begin{aligned} 2^W - y \cdot z_1 &< 2^W - y \cdot z_0 - \frac{y \cdot z_0 \cdot (2^W - y \cdot z_0)}{2^W} + y \\ &< \frac{(2^W - y \cdot z_0)^2}{2^W} + y \\ &< 2 \cdot y \end{aligned}$$

□

4. UNR recurrence method for W=64

We attempt to extend this method to the case where $W = 64$, while keeping the architecture unchanged. In the previous chapter, the theoretical arguments we presented helped us design the algorithm for $W = 64$. Below, we first present the algorithm.

Algorithm 3: UNR recurrence method for W=64.

```
// step I:  decent start
    constant_part = bit_cast<float>(0x5f7ffffd)
    z = (constant_part * v_rcp_f32(y as float)) as unsigned

// step II:  Two round of UNR (Unsigned integer Newton-Raphson)
Repeat 2 times:
z = z + umulh(z, -y * z)

// step III:  q estimate
q = umulh(x, z)
r = x - q * y

// step IV:  q estimate
If r >= y:
    q = q + 1
    r = r - y
If r >= y:
    q = q + 1
    r = r - y
```

The initial value selection and the proof of the number of iterations for $W = 64$ are similar to those for $W = 32$. Therefore, we will not elaborate further on this.

5. Conclusion

In this paper, we have explored the challenges associated with integer division in instruction set architectures (ISAs) that lack native support for this operation. By investigating functional iteration methods, specifically the improved Unsigned Newton-Raphson (UNR) recurrence method, we have demonstrated a robust approach to efficiently perform integer division in software. Our theoretical analysis for $W = 32$ establishes the effectiveness of the improved UNR method, providing a solid foundation for its extension to $W = 64$.

6. Acknowledge

This paper was written during my tenure at the Central Software Institute, Huawei 2012 Laboratories. The research conducted in this area facilitated my transition from purely theoretical research in academia to practical applications in my work. During this period, my colleagues helped me become familiar with various software tools. I would like to express my gratitude for their invaluable support and assistance.

References

[1] T. Rodeheffer, "Software Integer Division," MSR-TR-2008-141, Microsoft Research, 2008. [Online]. Available: `http://www.microsoft.com/en-us/research/publication/software-int`