기초디지털회로설계 HW#7 32bit Multiplication

20161453 전자공학과

김규래

컴퓨터에서 연산을 구현하는 방법에는 크게 소프트웨어적인 방법과 하드웨어적인 방법이 있다. 하드웨어적인 연산이 구현되어 있지 않을 경우에는 지원되는 연산들을 기반으로 그것을 소프트웨어적으로 구현하면 된다.

곱셈 연산은 사칙연산중에서 가장 기본적인 것에 속하지만 오랜 기간 동안 마이크로프로세서들에서 Single Instruction으로 제공하지 않았다. 특히 대중적으로 가장 처음으로 광범위하게 배포됐던 MOS 6502 마이크로프로세서 기반 컴퓨터들은 곱셈 instruction이 제공되지 않았기 때문에 소프트웨어적으로 구현돼야 했다. 따라서 하드웨어적인 곱셈 뿐만 아니라 소프트웨어적인 곱셈의 구현 방법에 관해 각각 알아보았다.

1. Software Multiplication Algorithm: Standard Algorithm

일반적으로 초등학교에서 가르치는 곱셈 방법을 Long multiplication 또는 Standard Algorithm 이라고 부른다.

각 자리의 숫자끼리 곱셈을 한 다음 마지막에 전부 더하는 식이다. 이를 의사코드로 표현하면 아래와 같다.

multiply(a[1..p], b[1..q], base)

tot := 0

**for** ri := 1 **to** p + q – 1

**for** bi := MAX(1, ri - p + 1) **to** MIN(ri, q)

ai := ri − bi + 1

tot := tot + (a[ai] b[bi])

product[ri] := tot **mod** base

tot := floor(tot / base)

product[p+q] := tot **mod**

return product

이 알고리즘은 N길이의 두 숫자에 대해 시간 복잡도와 공간 복잡도를 가진다. 이를 공간복잡도에 대해 최적화할 경우 의 공간 복잡도 정도만 필요로 하게 할 수 있다.

1. Software Multiplication Algorithm: Shift and Add

Small Integer 곱셈이나 나눗셈의 경우 Bit Shifting을 이용해서 매우 효율적으로 구현할 수 있다.

X = X << 1

X = X << N

C언어의 표기법을 예로 들 경우, 첫 번째 줄은 X에 2를 곱하는 것에 비례하며, 단지 X의 비트들을 1자리씩 옮기는 것에 불과하다. 두 번째 줄에서 N 만큼 비트를 쉬프팅할 경우 X에 를 곱하는 것과 같다.

1. Karatsuba Algorithm

이 알고리즘은 1962년에 발견되어 N크기의 숫자에 대한 곱셈의 복잡도를 까지 줄였다.

곱셈의 대상이 되는 두 숫자 는

로, 어떤 Base Number B 에 대해 표현할 수 있다. 두 숫자의 곱셈은 이 때,

가 된다.

이 숫자들의 연산은 총 4번의 곱셈을 필요로 하며 앞서 찰스 베비지가 발견했었다. Karatsuba 알고리즘에서는 이 과정에서 과 의 연산 결과를 이용할 경우 연산 횟수를 줄일 수 있다는 것을 반영하였다.

디지털 연산에서는 베이스 B 를 2로 두고 곱셈을 재귀적으로 시행할 경우 덧셈으로부터 큰 수의 곱셈을 구현할 수 있다.

Karatsuba 알고리즘의 재귀적인 형태를 의사코드로 표현하면 아래와 같다. 이 알고리즘은 Divide and Conquer 의 일종이기에 Dynamic Programming 과 같은 기법으로 추가적인 최적화가 가능하며, 의사코드는 어떠한 최적화없이 Naïve Recursion 을 이용한 것이다. 이 때문에 특정 상황에서는 Stack overflow 같은 문제에 노출될 수도 있다.

karatsuba(x, y, Base)

**if** (x 2) or (y 2)

return

m := max(sizeInBase(x), sizeInBase(y))

:= Karatsuba( )

( )

(+ ) -

1. Software Multiplication Algorithm: Schönhage–Strassen algorithm

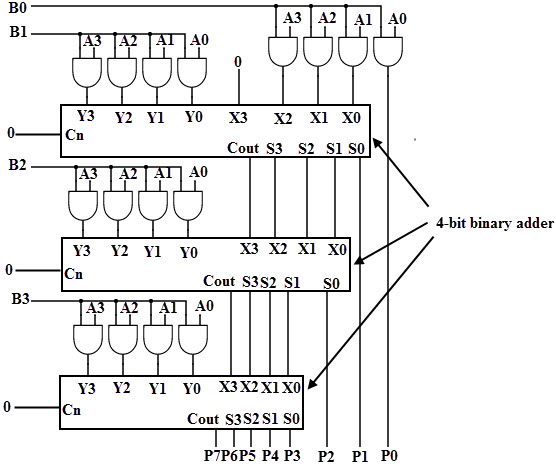
2007년 당시까지 최적의 곱셈 알고리즘은 재귀적인 Fast Fourrier Transform을 기반으로 한 Schönhage–Strassen algorithm 알고리즘이다. 시간 복잡도는 이다. 현재 이 이상으로 복잡도가 낮은 알고리즘은 Fürer's algorithm이 유일하며, 이 경우에는 천문학적 단위의 숫자에 관해서만 효과를 보이기 때문에 현실적으로는 Schönhage–Strassen algorithm이 가장 빠른 알고리즘이라고 할 수 있다.

Schönhage–Strassen algorithm은 기존의 Karatsuba 알고리즘과 Toom-Cook method 를 이용하며, 컴퓨터를 위해 최적화할 경우 Shift and Add를 사용하게 만들 수 있다. 또한 FFT 자체는 중간 과정에서 허수 연산이 필요하나, 허수 연산 자체는 느리고 오차율이 높기 때문에 Number theoric transform 을 사용한다고 한다.

이 알고리즘의 경우 행렬연산을 비롯해서 Forwared FFT, Inverse FFT 를 해야하는 등 다양한 High-Level 연산들을 필요로 한다. 게다가 대형 숫자의 연산에 주로 사용되는 기법이기 때문에 의사코드를 적지 않겠다.

1. Hardware Multiplication Method: Combinational Muliplier

Combinational Multiplier는 레지스터와 같은 메모리 소자를 사용하지 않는 곱셈 회로이다. 이는 위에서 Standard Algorithm이라고 설명한 방식을 하드웨어적으로 구현한 것이다.

Boolean Algebra에서 곱셈은 AND 연산으로 표현되기 때문에, Standard Algorithm에서의 곱셈은 쉽게 표현될 수 있다. 이렇게 각 자릿수를 곱한 결과를 더하면 아래와 같은 회로의 형태가 된다.

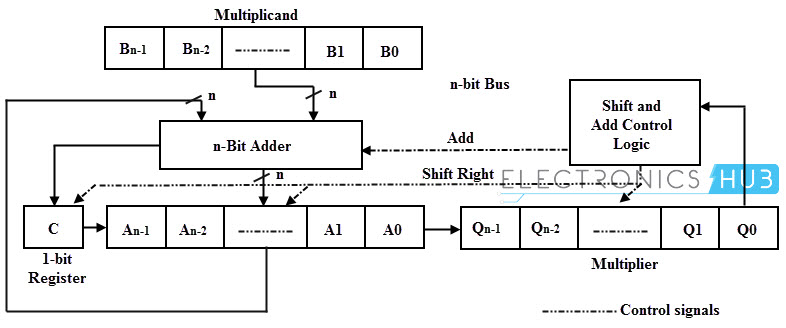
이를 확장할 경우, 사용하는 Binary adder들이 한번에 B개의 비트를 더할 수 있다고 할 때,

만큼의 딜레이가 발생한다. 여기에 Carry Lookahead Adder 유닛을 적용할 경우 이러한 딜레이를 줄일 수 있다. 2014년도 논문에서는 Carry Select Adder를 적용한 곱셈 회로도 연구가 됐다.

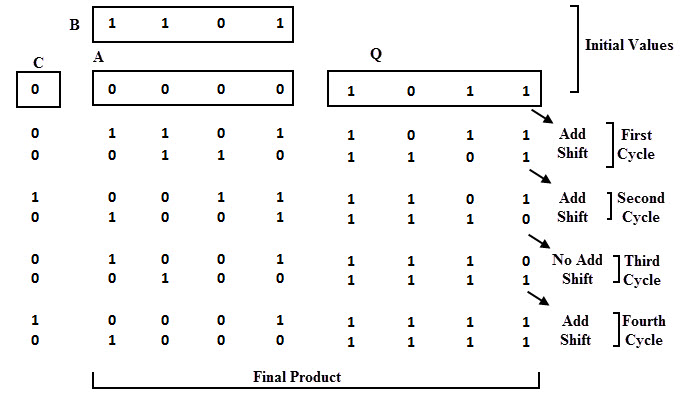
2의 보수를 적용한 이진수 데이터를 곱할 경우 일부 Partial Product의 비트를 역연선해야 하는 복잡함이 있다.

1. Hardware Multiplication Method: Sequential Circuit

Register를 사용한 Sequential Circuit 곱셈 연산 회로는 훨씬 더 적은 공간을 사용하여 구현할 수 있다. n Bit Adder 하나와 4개의 레지스터 A, B, C, Q 를 이용한다.



먼저 B의 LSB인 가 set 돼있는지 확인하는 걸로 시작된다. 곱해질 숫자들은 각각 B와 Q에 저장돼 있으며 A는 초기 상태가 0이다.



가 0일 경우 C 와 Q 레지스터는 하나의 레지스터로 취급되는 동시에 한 비트씩 쉬프트된다.

가 1일 경우 B의 비트들이 A의 LSB들과 더해지며 C 레지스터의 비트와 더해진다. A와 Q레지스터의 값들은 한 비트씩 쉬프트된다.

이러한 과정은 곱해지는 숫자의 사이즈 n번 반복된다. 즉 32비트, 32비트 연산에서는 총 32번 연산이 이루어진다는 것이다.

Shift and Add Control Logic은 일반적으로 FSM으로 구현된다.