10주차 예비보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20212022 이름: 이예준

**1.**

- 4-bit Adder -

Adder는 두 비트를 받아 덧셈을 하는 장치로 Half Adder의 경우 두 입력값을 넣으면

그 합과 Carry가 나오며, Full Adder의 경우 두 입력 값과 이전 자릿수의 Carry를 받아

그 합과 Carry를 산출한다. 4-bit Adder의 경우 Full Adder 4개를 병렬로 이어 붙여서

만드는데, 4bit의 두 데이터를 입력 받으면 각 Adder에 1-bit씩 두 입력값을 넣어주고,

이전 자릿수의 Carry Output을 현재 자릿수의 Carry Input으로 입력 받아서 계산한다.

그림은 아래와 같으며 Z0는 이전 자릿수의 계산이 없기 때문에 0을 입력 받는다.

또는 Half Adder를 첫번째 Adder로 사용할 수 있다.

스크린샷, 텍스트, 폰트, 상징이(가) 표시된 사진

자동 생성된 설명

\*4-bit Adder Block Diagram

예를 들어 X = 0101(5), Y = 0011(3)을 입력 받으면, 첫번째 Adder에서 1과 1을 받고

Z0에서 0을 받아서 더하면 합이 0이 나오고, Carry가 1이 나온다. 2번째 Adder에서 0과 1

그리고 이전 자릿수 Carry에서 1을 받아서 더하면 합이 0이 나오고 Carry가 1이 나온다.

역시 세번째 Adder에서 1,0,1을 받아서 더하면 합이 0이 나오고 Carry는 1이 나온다.

마지막에 0,0,1을 받아서 더하면 합이 1이 나오고 Carry가 0이 나오면서 최종적으로

결과값이 1000(8)이 나온다.

- 4-bit Subtractor -

Subtractor는 두 비트를 받아 뺄셈을 하는 장치로 Half Subtractor의 경우 두 입력값을

넣으면 그 차와 Borrow가 나오며, Full Subtractor의 경우 두 입력 값과 이전 자릿수의

Borrow를 받아 그 차와 Borrow를 산출한다. 4-bit Subtractor의 경우 Full Subtractor 3개와

Half Subtractor 1개를 병렬로 이어 붙여서 만드는데, 4bit의 두 데이터를 입력 받으면

각 Subtractor에 1-bit씩 두 입력값을 넣어주고, 이전 자릿수의 Borrow Output을 현재

자릿수의 Borrow Input으로 입력 받아서 계산한다. 그림은 아래와 같다.

도표, 라인, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

\*4-bit Subtractor Block Diagram

예를 들어 X = 0101(5), Y = 0011(3)을 입력 받으면, 첫번째 Subtractor에서 1과 1을 받아서

빼 주면 차가 0이 나오고, Borrow가 0이 나온다. 두번째 Subtractor에서 0과 1을 받아서

빼 주면 차가 1이 나오고 Borrow가 1이 나온다. 세번째 Subtractor에서 1과0 그리고

이전 자릿수의 Borrow 1을 받아서 빼 주면 차가 0이 나오고 Borrow도 0이 나온다.

마지막에 0,0,0을 받아서 빼 주면 차가 0이 나오고 Borrow도 0이 나오면서 최종적으로

결과값이 0010(2)가 나온다.

**2.**

위의 Adder를 보게 되면 이전 자릿수의 Carry가 계산되지 않으면 현재 자릿수의 합과

Carry를 계산할 수가 없다. 따라서 입력 받는 bit수가 커지면 커질수록 Delay는 한없이

늘어나게 되는 단점이 있다. 이런 방식을 Ripple Carry Adder라고 한다. 이러한 단점을

보완하여 모든 자릿수의 Carry를 예측하여 한번에 계산하는 방식을 Look Ahead Carry 또는

Carry Look Ahead 라고 한다. 그림은 아래와 같다.

텍스트, 도표, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

\*4-bit Look Ahead Carry Adder Block Diagram

위의 그림에서 Carry를 예측하는 방식은 다음과 같다.

Carry-Generate 공식은 이며, Carry-Propagate 공식은 이다.

위의 두 공식을 이용하면 두 입력 값의 합과 Carry를 구할 수 있다.

합(Sum) 공식은 이며, Carry 공식은 이다.

이 공식들을 이용해 각 자릿수의 Carry를 예측할 수 있는데, 1번째 bit의 Carry Output을

이라고 할 때 위의 그림의 각 bit의 Carry는

이 된다.

이때 공식에 을 대입하면 이 되고,

공식을 정리하면

이 된다.

또 이 공식을 공식에 대입하면 이 되고,

공식을 정리하면

이 된다.

마지막 공식도 공식을 대입 후 정리하면

이 된다.

이 공식들을 일반화하게 되면 최종적으로

이 된다.

따라서 이전 자릿수의 Carry를 계산하지 않고 첫번째 자리의 Carry Input만 알아도

모든 자릿수의 Carry를 계산할 수 있다.

언뜻 보면 Look Ahead Carry 방식이 Ripple Carry 방식에 비해 모든 면에서 좋아 보이지만

Look Ahead Carry 방식도 단점이 있다. 바로 Ripple Carry 방식에 비해 더 복잡한 구조를

가지고 있어 bit수가 커질수록 Carry를 계산하는 과정이 복잡해져서 Delay가

발생할 수 있다. 하지만 이러한 단점은 4-bit CLA를 Module화해 여러 개를 병렬 연결하여

사용하게 되면 줄일 수 있다.

타이포그래피이(가) 표시된 사진

낮은 신뢰도로 자동 생성된 설명

\*16-bit CLA Adder using 4-bit CLA Adder Block Diagram

**3.**

위에서는 Adder와 Subtractor를 따로 구현해서 설명했는데, 이 두 회로를 XOR Gate를

이용한 2의 보수를 활용해 하나의 회로로 구현할 수 있다. 기본적으로 가감산기는

Full Adder를 병렬 연결을 한 구조로 덧셈인지 뺄셈인지 알기 위한 신호로 입력값 K를

넣는다. 만약 덧셈이라면 0을 입력하고, 뺄셈이라면 1을 입력하는데 이 입력값을 통해

각 bit의 보수를 취할지 말지 결정한다. K에 1을 입력하게 되면 아래 그림과 같이 B의 bit와 XOR연산을 하게 되어 B의 보수를 취한 뒤 A와 B의 각 bit를 더해준다. 당연히 K에 0을 입력하게 되면 B의 bit는 바뀌지 않는다. 2의 보수를 취할 때 bit를 반전시킨 뒤 1을 더해줘야 하는데 그 1은 K를 통해 더해준다. 따라서 K를 Cin으로 입력해준다.

도표, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명

\*4-bit Adder-Subtractor 회로도

**4.**

BCD란 0~9까지의 십진수를 4-bit의 이진수 형태로 바꾸어서 십진수의 각 자릿수를

이진수로 표현하는 것을 말한다. 예를 들어 42는 0100 0010 이라고 표현할 수 있다.

BCD연산은 그렇게 이진수로 변환한 각각의 BCD 코드들을 덧셈, 뺄셈, 곱셈, 나눗셈과 같은

연산을 하는 것을 말하며 그러한 연산을 마친 후 다시 십진수로 전환하여 나타낸다.

이때 BCD에는 0~9까지만 표현할 수 있기 때문에 이진수의 연산이 10이상이 되면 보정이

필요하다. 이진수 연산이 10 이상이 되면 다음 BCD코드에 1을 더해주고 현재 계산한

자릿수의 BCD코드에서 6을 더해 주어야 한다.

예를 들어 53과 12를 더하는 계산을 하면, 53은 0101 0011, 12는 0001 0010이다.

각 BCD 코드끼리 더해주면 0110 0101, 즉 63이 되면서 정상적인 계산 값이 나온다.

다른 예를 들어 29와 17을 더해주면, 29는 0010 1001, 17은 0001 0111이 된다.

각 BCD코드끼리 더해줄 때, 0111과 1001의 합이 1 0000이 되면서 10을 초과하게 되기에

현재 계산 값에 0110을 더하고 다음 자릿수 계산에 1을 추가한다. 따라서 일의 자리는

0110이 되고 십의 자리는 0010+0001+0001=0100이라는 계산 값이 나온다. 최종적으로

결과는 0100 0110이 되면서 46이라는 정상적인 계산 값을 얻을 수 있다. 뺄셈을 할 때

여러가지 방법이 있지만 이번에 설명할 방법은 9의 보수를 이용하는 방법이다. 피감수의

9의 보수를 취한 뒤 감수와 피감수의 보수를 더해준다. 이때 Carry가 발생하면

최하위 비트 1을 더해주고, Carry가 발생하지 않았다면 최종 계산 값에 9의 보수를 취하고

‘ - ‘부호를 붙인다. 예를 들어 51 – 21을 하게 되면, 51은 0101 0001, 21은 0010 0001이

된다. 이때 21에 9의 보수를 취하면 78, 즉 1110 1000이 되며, 51과 78을 더해주면

129가 된다. 이때 세 자릿수 1이 Carry로 발생했기 때문에 1을 버려주고 최하위 비트에

1을 더해준다. 그러면 최종적으로 결과는 30, 즉 0001 1110이 나온다.

**5.**

ALU란 Arithmetic and Logical Unit의 약자로 두 수의 사칙연산, 논리연산, 비교연산, Shift 등을 계산하는 디지털 회로이다. Accumulator, Adder, Register, Complementary 등으로 구성되어 있다.

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

\*ALU Block Diagram

Accumulator는 산술 및 논리 계산 결과를 임시로 저장하는 장치이며, Adder는 논리합을

겨산하고, Complementary는 보수 값을 출력한다. 또한 Register는 Data Register와

Status Register가 있으며, Data Register는 연산에 필요한 데이터를 임시로 저장하고 있다가

필요할 때 제공하는 역할을 하며, Status Register는 연산 결과가 음수인지, 양수인지 또는

Carry나 Overflow가 발생했는지 등, 연산과 관계가 있는 정보를 저장한다.

ALU는 제어장치로부터 제어신호를 전달받아 두 Data Register의 값을 연산한 뒤 결과값을

Accumulator에 저장한다. 이때 연산 결과에 대한 추가정보는 Status Register에 저장한다.

**6.**

디지털 코드에는 BCD 코드 말고도 다양한 코드가 존재한다. Excess-3 코드는 BCD코드에

3을 각각 더한 코드로 반전시키는 것만으로도 9의 보수를 구할 수 있어 감산에 유용하다.

Gray 코드는 서로 이웃하는 숫자와 1비트만 다른 코드로 오류가 적은 장점이 있어

데이터 전송에 잘 활용된다. Parity 코드 또는 Hamming 코드는 전송 받은 데이터의 오류를

검출하는 용도로 사용된다. 그 외에도 ASCⅡ코드, EBCDIC와 같은 디지털 코드들이 있다.