컴퓨터공학실험2 10주차 예비 보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**0. 목차**

1. 4-Bit adder 및 Subtractor 이진 병렬 연산 기능에 대해 조사한다.

2. Look ahead carry를 조사한다.

3. XOR을 활용한 2’s complement 가감산을 조사한다.

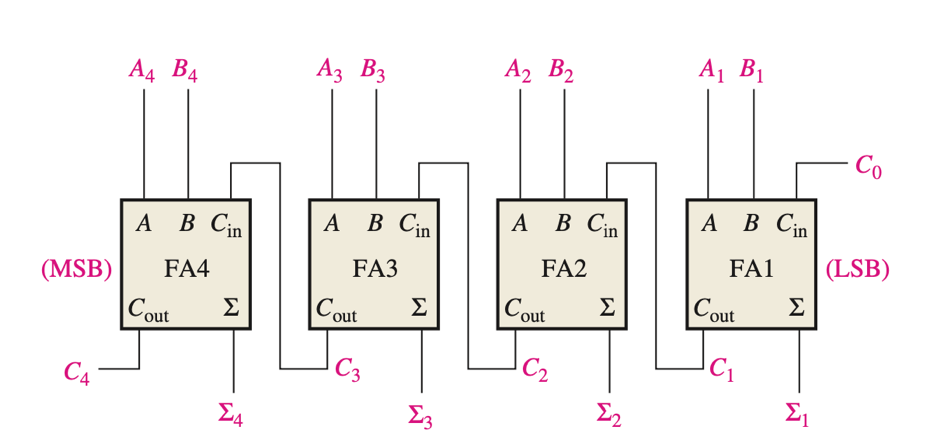
4. BCD 연산을 조사한다.

5. ALU의 기능에 대해 조사한다.

6. 기타 이론

**1. 4-Bit adder 및 subtractor 이진 병렬 연산 기능**

6주차 실험에서 알아본 것처럼, adder/subtractor에는 half/full 두 가지 종류가 존재한다. 여기에서 full adder/subtractor을 병렬적으로 연결하면 1bit 단위의 연산 뿐만이 아닌, 2,3, …, n bit에서의 덧셈, 뺄셈 연산을 수행할 수 있다는 것을 설명한 적이 있다. 이때 4 bit adder의 형태는 다음과 같다..



<4-Bit parallel adder의 형태>1

또한 과정은 아래와 같다.

(1) A1, B1, C0를 input으로 받는 full Adder을 수행한다.

(2) (1)의 결과 C1, A2, B2를 input으로 받는 full Adder을 수행한다.

(3) (2)의 결과 C2, A3, B3를 input으로 받는 full Adder을 수행한다.

(4) (3)의 결과 C3, A4, B4를 input으로 받는 full Adder을 수행한다.

위에서 설명한 것처럼 full Adder을 병렬적으로 연결해서 여러 bit의 덧셈을 수행할 수 있는 회로를 구현할 수 있다. 이와 동일하게 full Subtractor을 병렬적으로 연결해서 여러 bit에서의 뺄셈을 수행할 수도 있다. LSB부터 MSB까지 각각의 full subtractor이 차례대로 수행되며 n bit의 연산을 수행할 수 있다. 또한 4-Bit parallel subtractor의 형태도 위의 4-Bit parallel adder에서 Carry 대신에 Br을 사용한 형태임을 파악할 수 있다. 과정 또한 위와 동일하므로 이는 생략한다.

**2. Look ahead carry**

위에서 본 형태처럼, full Adder/Subtractor을 병렬적으로 연결해서 여러 bit에서 연산을 수행하는 회로를 구현해낼 수 있다. 하지만 이렇게 구성한 회로에서는 단점이 존재하는데, 위에서 설명한 것처럼 LSB부터 MSB까지 연산이 ‘차례대로’ 수행된다는 점이다. 예를 들어 1101 + 1011 이라는 덧셈이 수행되기 위해서 1 + 1 은 바로 수행이 되지만, 그 다음 연산인 0 + 1 + 1(carry에 의한 1) 이 수행되기 위해서는 이전 full Adder의 결과인 C1를 알고 있어야 하기 때문이다. 여기에서 이 delay(이전 결과들을 모두 기다려야 하는 delay)가 연산 능력보다 크다면 덧셈의 성능이 매우 비효율적임을 알 수 있다. 따라서 각각의 연산 결과를 기다리는 것보다, 한 번에 계산을 수행할 수 있는 방법이 존재하는데, 이 방법이 Look ahead carry 이다.

Look ahead carry란, 이전 연산 결과인 c를 알 필요 없이 output C를 계산하는 것이다. 우선 1bit full Adder에 대해서 생각해보면 carry가 생기는 경우는 두 가지 형태로 아래와 같다.

(1) AB = 1 인 경우 (A = 1, B = 1인 경우)

(2) (A^B)C in = 1 인 경우 (A = 1, C in = 1, B = 1, C in = 1인 경우)

따라서 C out = AB + (A^B)C in 과 같이 표현된다.

이번에는 4bit parallel Adder로 확장해서 마지막 output C4를 이처럼 표현해보자. 그 과정은 다음과 같다.

(1) C4 = A4B4 + (A4^B4)C3 형태이다.

(2) C3 = A3B3 + (A3^B3)C2을 대입하면, C4 = A4B4 + (A4^B4)(A3B3 + (A3^B3)C2) 형태이다.

(3) 동일한 방식으로 계속해서 수행하면, C4를 A, B, C0만으로 표현해낼 수 있다.

이렇게 과정을 거치면 C4를 이전 연산을 기다리는 delay없이 한 번에 구해낼 수 있다. 또한 C3, C2, C1 또한 delay 없이 한 번에 구해낼 수 있어서, 이전 결과를 기다리는 4-bit parallel Adder보다 훨씬 시간 면에서 효율적인 회로를 구현할 수 있다.

하지만 Look ahead carry 형태에는 단점이 존재하기도 한다. 단순히 생각해도 훨씬 많은 gate가 필요하다는 것을 파악할 수 있다.(모든 C를 Look ahead carry 형태로 표현하기 위해서는 n bit 연산에서 O(n2)개의 회로가 필요한데, 이는 일반적인 parallel adder과 O(n)개 필요한 것과 큰 차이가 있다.)

**3. XOR gate를 이용한 2’s complement 가감산**

1’s / 2’s complement는 XOR gate를 사용해서 구현해낼 수 있다. 1’s complement는 XOR gate만으로, 2’s complement는 XOR gate와 함께 0001을 더해주면서 구현하게 된다. 여기에서 더 나아가 XOR gate를 이용해서 가감산을 수행할 수도 있는데, 우선 가산(덧셈)의 경우에는 parallel adder을 통해서 수행할 수 있다.(일반적인 binary adder와 동일한 방식으로 수행할 수 있다) 하지만 단순하게 adder로 사용해서 끝내는 것이 아닌, adder로 구한 결과가 실제로 2’s complement의 범위 내에 들어가는지 파악해야 한다. 이는 여러 병렬 adder에서 가장 왼쪽 MSB의 덧셈을 수행하는 adder에서 Cin과 Cout이 동일한지를 파악해서 알 수 있다. 만약 동일하다면, 오류가 없는 것이다.

2’s complement의 감산(뺄셈)은 가산을 활용해서 구현할 수 있는데, A – B의 경우에서 B에 다시 2’s complement를 취해 A + B’ 형태로 만들어주는 것이다. 이때 XOR gate와 adder을 활용한다면 간단한 방법으로 감산을 수행할 수 있다.

**4. BCD 연산**

BCD는 4bit에 각 10진수의 한 자리를 표현하는 코드이다. 이때 BCD 또한 binary code와 같이 덧셈, 혹은 뺄셈을 수행할 수 있다. 덧셈을 수행하는 방식은 4-Bit parallel adder과 동일하다. 각 bit에서 덧셈을 차례대로 수행하며 결과를 만들어낸다. 하지만 BCD에서 중요한 것은 4bit 단위로 0~9를 표현해야 한다는 것이다. 따라서 만약 BCD의 덧셈 연산의 결과가 10~18이라면(특히 10~15 라면) 구현된 결과에 대해서 10을 빼준 이후 Carry output에 1을 더해주는 과정이 필요하다. 이는 6주차 예비보고서에서 회로와 과정을 설명한 적이 있으므로 생략한다.

BCD 뺄셈은 덧셈과 다르게 1’s complement method로 구현할 수 있다. 뺄샘을 수행하는 각 4bit를 1’s complement로 변환해준 이후, 덧셈을 수행하는데, 이때 EAC(end-around-carry)라는 새로운 변수를 활용하며 뺄셈을 수행하게 된다. EAC가 1이라면 총 결과가 + 이며, EAC = 0이라면 총 결과가 – 임을 토대로 뺄셈을 수행한다. 각 경우에 맞추어 A – B라는 연산에서 A/ B에 1010을 더해주며 뺄셈을 수행할 수 있다.

더 간단한 방법으로는, 계산을 두 번 수행하는 것인데, 4-bit parallel subtractor로 계산한 결과에서 output Br = 1인 경우에, Br = 1을 유지하며, 나온 결과에 대해서 6을 빼주는 형식으로 계산을 다시 수행하며 뺄셈을 구현할 수도 있다.

**5. ALU의 기능**

ALU는 arithmetic logic unit의 약자로, CPU, GPU 등에서 기본적인 연산(산술 연산, 논리 연산)을 수행할 수 있는 한 unit 이다. ALU가 수행하는 산술 연산에는 위에서 계속해서 수행했던 덧셈, 그리고 덧셈과 1’s / 2’s complement를 활용한 뺄셈을 수행한다. 또한 곱셈과 나눗셈을 수행할 수도 있는데, 이 또한 덧셈과 뺄셈을 통해서 구현해낼 수 있다.

논리 연산에서는 AND, OR, XOR 등의 기본적인 논리 연산을 여러 bit 단위로 수행할 수 있다. 또한 산술 연산에서 필요한 complement 연산도 논리 연산 영역에서 수행한다. 또한 어셈블리에서 다양한 형태로 활용하게 되는 Shift 연산 또한 논리 연산의 영역에서 수행할 수 있다.

**6. 출처**

1. Thomas L. Floyd, Person, Digital Fundamentals Eleventh edition(global edition), 326p