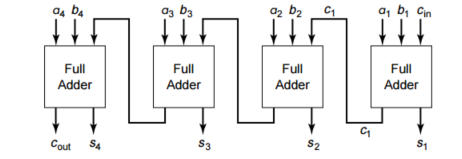
10주차 예비보고서

전공: 물리학과 학년: 3학년 학번: 20191286 이름: 김나현

1. 4-bit adder 및 subtractor 이진 병렬 연산 기능에 대하여 조사하시오.

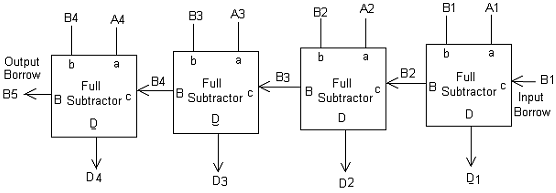
n-bit adder나 subtractor란 1 bit 전가산기나 전감산기, n개를 이용하여 n-bit 이진수의 덧셈 또는 뺄셈을 하기 위한 논리 회로를 의미합니다. 따라서 4-bit adder의 경우, 1 bit 전가산기(full adder) 4개를 병렬로 연결하여 4-bit 이진수 두 개를 더하는 논리 회로이고 4-bit subtractor의 경우, 1 bit 전감산기(full subtractor) 4개를 병렬로 연결하여 4-bit 이진수 두 개를 빼는 논리 회로입니다.

다음 <Figure 1>은 4-bit adder로, A(=a4a3a2a1)이라는 4-bit 이진수와 B(=b4b3b2b1)이라는 4-bit 이진수를 더하여 s4s3s2s1이라는 결과와 cout이라는 출력을 얻는 논리 회로입니다.



<Figure 1> 4-bit adder

4-bit adder은 입력으로 들어오는 두 이진수의 하위 자릿수부터 더하여 s1, s2, s3, s4 순서대로 생성하고 출력하게 되는데 각 전가산기의 출력 중 s는 ai, bi, cin 중 1인 입력의 개수가 홀수일 때 1을, 짝수일 때 0을 출력하게 되고 cout은 ai, bi, cin 중 1인 입력의 개수가 2개 이상일 때 1을 출력하고, 그렇지 않을 때 0을 출력하게 됩니다. 따라서, 4-bit 두 이진수를 더해서 최종적으로 얻은 cout이라는 값이 1일 때는 두 이진수를 더한 값이 4 bit보다 크다는 의미이고, cout이 0일 때는 두 이진수를 더한 값이 4 bit를 넘지 않는다는 의미입니다.

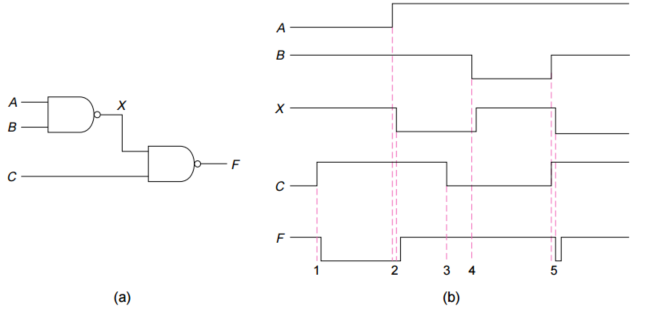
다음 <Figure 2>는 4-bit subtractor로, A(=a4a3a2a1)에서 4-bit 이진수와 B(=b4b3b2b1)이라는 4-bit 이진수를 빼서 D4D3D2D1라는 결과와 Bout라는 출력을 얻는 논리 회로입니다.

<Figure 2> 4-bit subtractor

4-bit subtractor은 입력으로 들어오는 두 이진수의 하위 자릿수부터 빼서 D1, D2, D3, D4 순서대로 생성하고 출력하게 되는데 각 전가산기의 출력 중 D는 ai, bi, Bin 중 1인 입력의 개수가 홀수일 때 1을, 짝수일 때 0을 출력하게 되고 Bout은 ai’bi’Bin+ai’biBin’+ai’biBin+aibiBin이라는 논리 함수로 표현이 가능합니다.

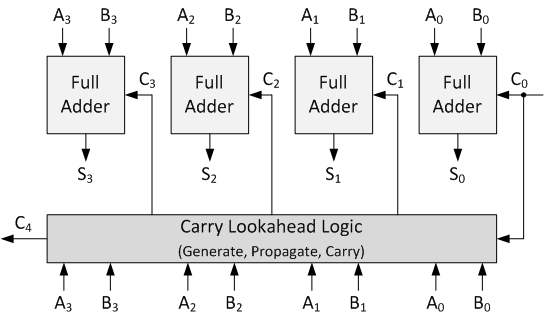
1. Look ahead carry에 대하여 조사하시오.

<Figure 1>, <Figure 2>와 같은 adder나 subtractor는 하위 자릿수부터 연산하여 상위 자릿수로 물결 치듯 넘어가므로 이러한 adder나 subtractor를 ripple carry adder나 ripple carry subtractor라고 부릅니다. 하지만 논리 게이트에 입력이 들어와서 출력을 생성할 때까지는 약간의 시간 지연(delay, Δ)이 있고 하나의 논리 게이트의 출력이 다른 게이트의 입력으로 들어가면 이러한 시간 지연이 점차 늘어납니다. 다음 <Figure 3>는 그러한 시간 지연을 알아보기 위해 단순한 회로를 이용하여 시간 지연을 timing diagram을 이용하여 나타낸 것입니다.



<Figure 3> Illustration of gate delay

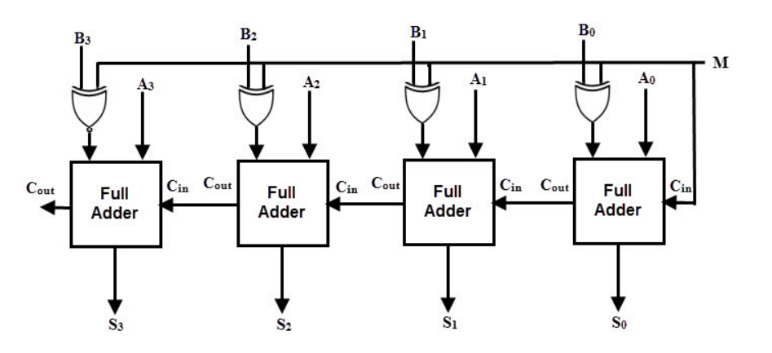
만약 n-bit adder가 있다고 가정하면 첫 번째, 즉 가장 하위 자릿수를 연산하는 전가산기에서의 시간 지연은 5Δ가 되고, 중간에 있는 자릿수들을 연산하는 전가산기에서의 시간 지연은 2(n-2)Δ가 되고, 가장 마지막, 즉 가장 상위 자릿수를 연산하는 전가산기에서의 시간 지연은 3Δ가 되기 때문에 n-bit adder의 전체 시간 지연은 2(n+2)Δ라고 표현할 수 있습니다. Ripple carry adder의 이러한 시간 지연을 줄이기 위해 Carry-look-ahead adder가 사용됩니다. Look-ahead라는 말처럼 carry-look-ahead adder는 앞을 미리 내다보는 것처럼 동작을 하게 됩니다. 이 말의 의미는 n-bit adder에 있는 각 전가산기에서 adder에 사용되는 carry를 하위 자릿수에서부터 계산해오는 것이 아니라, 각 단계마다 carry-generate signal과 carry-propagate signal이라는 두 개의 출력을 생성하여 하위 자릿수에서부터 연산을 해오는 것을 기다리지 않고 각 전가산기의 carry를 바로바로 계산한다는 것입니다. 이때, carry-generate signal gi는 gi=xiyi라는 함수로 만들 수 있고, carry-propagate signal pi는 pi=xi⊕yi라는 함수로 만들 수 있습니다. gi와 pi를 이용하여 4-bit adder의 각 전가산기의 carry bit를 c2=g1+p1c1, c3=g2+p2g1+p2p1c1, c4=g3+p3g2+p3p2g1+p3p2p1c1, c5=g4+p4g3+p4p3g2+p4p3p2g1+p4p3 p2p1c1라는 논리 함수로 구할 수 있습니다. Carry-look-ahead adder는 4-bit carry chain 대신 CLA(carry-look-ahead) generator를 이용해 ripple carry adder의 시간 지연을 줄일 수 있습니다.



<Figure 4> Carry-look-ahead adder

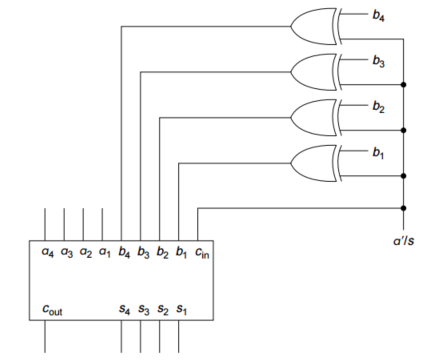
1. XOR을 활용한 2’s complement 가감산에 대하여 조사하시오.

4-bit adder와 subtractor는 1-bit 전가산기만을 이용하여 구현할 수 있습니다. XOR 연산의 특성상1⊕x=x’이며 0⊕x=x이라는 것을 알아야 하고 A라는 것의 2의 보수는 A’+1으로 구할 수 있다는 것을 알아야 합니다. 그럼 다음 <Figure 4>와 같이 4-bit adder와 subtractor로 동시에 활용 가능한 논리 회로를 구현할 수 있게 됩니다. 이때, M이 0이면 전가산기에 들어가는 B의 입력이 원래 Bi이고, 가장 하위 자릿수의 1-bit 전가산기에 들어가는 cin 또한 0이므로 일반적인 4-bit adder 논리 회로로 동작하고 M이 1이면 전가산기에 들어가는 B의 입력이 Bi’으로, B의 1의 보수가 되고 가장 하위 자릿수의 1-bit 전가산기에 들어가는 cin이 1이 되어 각 전가산기가 Ai+Bi’+1과 같은 연산을 수행하기 때문에 전감산기와 같은 역할을 하게 됩니다.



<Figure 5> 4-bit adder/subtractor

<Figure 5>를 하나의 4-bit adder/subtractor로 나타나게 그려보면 다음 <Figure 6>와 같습니다. <Figure 6>에서는 a’/s가 <Figure 5>의 M과 같은 역할을 하는데 a’/s가 0이면 <Figure 6>의 논리 회로는 4-bit adder로, a’/s가 1이면 <Figure 6>의 논리 회로는 4-bit subtractor로 동작하게 됩니다.



<Figure 6> 4-bit adder/subtractor

1. BCD 연산에 대하여 조사하시오.

두 개의 이진수 0110과 0111을 더한다고 가정하면, 1101이 되고 carry는 0이 됩니다. 0110과 0111은 각각 6과 7이므로 두 값을 더하면 13이 되어 4비트로 표현이 가능하기 때문에 이러한 경우는 문제가 없지만 13(1101)과 5(0101)를 더한다고 하면 18, 즉 10010이 되므로 4비트를 넘어 5비트로 표현이 가능한 값이 됩니다. 이처럼 n-bit 양수를 더하였을 때, (n+1)-bit의 값이 나오는 경우, overflow가 발생하였다고 말을 합니다. 예를 들어, 4비트 양수를 더할 때 16과 같거나 큰 결과를 얻으면 overflow가 발생하고 13과 5를 더한 앞선 예시에서도 18은 15보다 크기 때문에 4비트로 나타낼 수 있는 최대 숫자를 넘어가므로 overflow가 발생한 것입니다. 위에서 다룬 4-bit adder의 경우, 두 개의 4-bit 양수를 더하여 s4s3s2s1이라는 합을 얻고, cout을 얻는데 이때 cout이 1이면 overflow가 발생하였다고 말할 수 있습니다.

지금까지는 우리는 양수에 대해서만 고려하였지만 컴퓨터는 음수를 포함한 signed number를 다뤄야 하므로 이진수를 이용하여 양과 음의 부호를 나타내는 법에 대해 알아보겠습니다. 양수와 음수를 구별하는, 인간에게 익숙한 표기법은 +, -의 기호이지만 컴퓨터는 이러한 부호를 사용하지 않고 이진수의 n개의 bit 중 첫 번째 비트, MSB를 sign indicator로 사용합니다. 이때, 컴퓨터는 이진수의 MSB가 0이면 양수로 인지하고, 1이면 음수로 인지합니다. 따라서, 4비트 시스템에서 +5는 0101으로 나타내고 -5는 1101으로 나타낼 수 있습니다. 4비트 시스템에서 MSB를 제외한 3비트만을 magnitude를 나타내기 위해 사용하므로 4-bit signed number를 사용하면 -7부터 +7까지의 숫자를 나타낼 수 있습니다. 하지만 이런 방법으로는 0000과 1000이 모두 0을 표현하는 데에 사용되기 때문에 혼란을 야기할 수 있습니다. 따라서 일반적으로 컴퓨터가 음수를 표현할 때에는 2’s complement, 즉 2의 보수라는 방법을 사용합니다.

2의 보수를 사용하면 양수는 일반적인 이진수 표기법을 이용하여 나타내고, 음수는 절댓값을 이진수로 표현한 다음, 모든 비트의 보수를 취한 후, 1을 더하여 나타냅니다. 이렇게 하면 n개의 비트가 있을 때, -(2n-1)부터 +(2n-1-1)까지의 숫자를 표현할 수 있고 0을 표현하는 방법 또한 0000으로, 한 가지만 존재한다는 장점이 있습니다. 2의 보수를 이용하여 덧셈을 할 때, +4와 +5를 더한다고 가정하면 0100과 0101을 더하여 1001이 되고 이 값은 -7을 의미하므로 우리가 원하는 +9를 얻을 수 없습니다. 따라서 2의 보수를 더할 때, 두 양수를 더해서 음수의 결과를 얻거나 음수를 더해서 양수의 결과를 얻게 되면 overflow가 발생하였다고 할 수 있습니다. 하지만, 양수와 음수를 더할 때에는 절대 overflow가 발생하지 않습니다.

2의 보수의 뺄셈을 하는 방법은 단순히 두 번째 operand의 2의 보수를 얻어 첫 번째 operand와 두 번째 operand의 2의 보수를 더하는 것입니다. Signed number에 대한 2의 보수 뺄셈에서 MSB에서 발생한 carry는 무시되고 첫 번째 operand와 두 번째 operand의 2의 보수의 부호가 같고 그 두 값을 더한 결과의 부호와 첫 번째 operand의 부호와 다르다면 overflow가 발생했다고 합니다. 반면, unsigned number에 대한 2의 보수 뺄셈에서는 MSB에서 발생한 carry가 1일 때 overflow가 발생했다고 합니다.

1. ALU의 기능에 대하여 조사하시오.

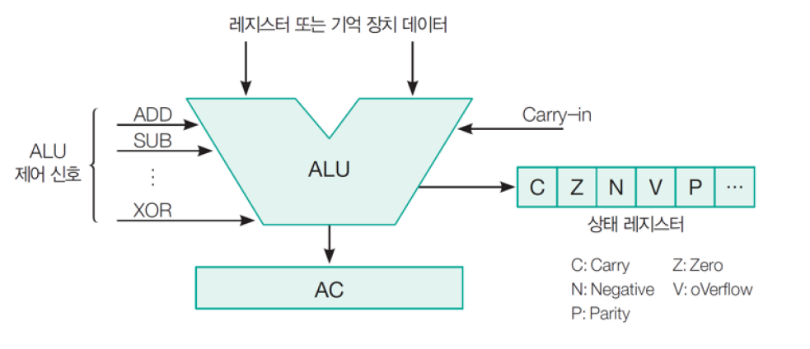
ALU는 arithmetic logic unit의 약자로, 산술 논리 장치를 의미합니다. ALU는 덧셈과 뺄셈, 곱셈과 나눗셈, 증가, 감소 등의 산술연산(arithmetic)과 논리합(OR), 논리곱(AND), 배타적 논리합(XOR), 논리 부정(NOT) 등의 논리연산(logic)을 하는 디지털 회로이며 컴퓨터 CPU의 기본 설계 단위라고 할 수 있습니다. 이때, 덧셈, 뺄셈과 같은 산술연산을 하기 위해서는 앞에서 언급한 가산기와 감산기로 모두 이용이 가능한 n-bit adder/subtractor를 사용합니다. 뿐만 아니라 ALU는 비교 연산, 보수 연산, 시프트 연산 등을 수행할 수 있으며 이 중 시프트 연산은 아래 <Figure 7>과 같이 오른쪽이나 왼쪽으로 비트를 이동하는 것을 의미합니다.

테이블이(가) 표시된 사진

자동 생성된 설명

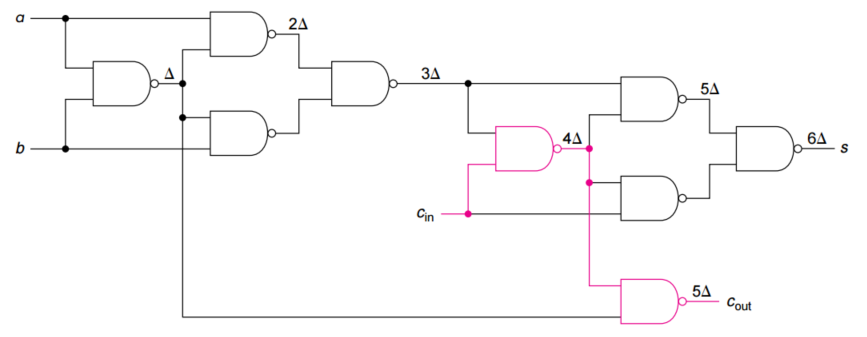
<Figure 7> ALU에서의 시프트 연산의 예시

ALU에서 마이크로 연산이 일어나는 것을 다음 <Figure 8>과 같은 diagram으로 표현할 수 있습니다. ALU는 제어 신호에 따라 레지스터 또는 기억 장치로부터 데이터를 가져와 연산을 하며 연산 결과를 상태 레지스터에 저장하게 됩니다. ALU가 연산을 수행할 때, 연산 장치의 입력 데이터와 출력 결과를 일시적으로 보관해놓는 기억 장소를 accumulator라고 하며 아래 <Figure 8>에서는 AC라는 이름으로 표현되어 있습니다.



<Figure 8> ALU에서 연산이 일어나는 과정을 나타낸 diagram

1. 기타 이론



<Figure 9> 1-bit adder에서의 시간 지연

위의 <Figure 9>와 같은 1-bit adder schematic diagram을 통해 1-bit adder에서의 시간 지연을 알아보도록 하겠습니다. N-bit adder에서 첫 번째 노드, 즉 가장 하위 자릿수를 연산할 때 발생하는 시간 지연을 5Δ라고 하였는데 이는 a와 b, cin을 이용해 cout을 출력할 때까지의 시간 지연으로 <Figure 9>를 보면 cout을 출력하기까지 다섯 단계의 GATE를 거치므로 5Δ임을 확인할 수 있습니다. 또한 n-bit adder에서 마지막 노드에서의 시간 지연은 3Δ라고 하였는데 이는 cin이 입력되고부터 s를 출력하기까지의 시간 지연으로 <Figure 9>에서 볼 수 있듯이 세 단계의 GATE를 거치므로 3Δ임을 확인할 수 있습니다.