10주차 예비보고서

전공: 경영학과 학년: 4학년 학번: 20190808 이름: 방지혁

1.

4-bit adder와 subtractor은 4 bit 크기의 두 이진수에 대하여 덧셈 연산 및 뺄셈 연산을 하기 위하여 사용하는 회로입니다. 우선, 4-bit adder는 각 자리의 비트마다 full adder가 존재하며, 이 4개의 adder를 병렬로 연결하여 구성합니다. 각 자리에서는 캐리비트가 다음 비트로 전달되는 구조입니다. 이러한 4-bit adder에서 나아가 가산 및 감산 기능이 각기 다른 회로에서가 아니라 한 회로에서 모두 가능할 수 있도록 만들 수 있는데 이것이 4bit adder/subtractor입니다. 기존의 감산기에 input 신호 값을 하나 더 추가하여 가능하게 할 수 있습니다. 방식에 대하여 부가적인 설명을 하자면, a - b = a + (-b)와 같으므로 우리는 b에 2의 보수를 취해주는 것으로 마이너스 연산을 한다고 생각할 수 있습니다. 그렇기에 input 신호 값이 1이 주어지고 이를 b의 각 자리 bit와 xor 연산을 한다면 반전이 되어, 감산 연산을 구현할 수 있게 될 것입니다. 또한, LSB에 이 input 신호인 1이 더해집니다. input 값이 0이라면 보수화되지 않기 때문에 가산 연산을 기존처럼 하게 되는 것입니다. 이것에서 input값이 그냥 1으로 고정되게 만든다면 그것은 오직 감산 연산만을 수행하는 subtractor로서 동작하게 될 것입니다.

2.

1번 항목에서 언급한 병렬 가감산기는 계산 시간이 오래 걸린다는 단점이 존재합니다. Look ahead carry는 한 번에 carry를 계산하는 방식을 통해 이전에 순차적으로 계산하면서 생기는 계산 시간의 단점을 개선합니다.

이에 대하여 부가적인 설명을 하자면, carry generate를 $g_i = x_i y_i$, carry propagate를 $p_i = x_i \oplus y_i$라고 설정을 합니다.

carryout인 $c_{i+1} = x_i y_i + x_i c_i + y_i c_i = x_i y_i + (x_i + y_i) c_i = g_i + p_i c_i$를 이렇게 간단화하여 나타낼 수 있습니다. 예를 들자면,

c1 = g0 + p0 * c0

c2 = g1 + p1 * C0 = g1 + p1*(g0 + p0 * c0) = g1 + g0* p1 + p1 * p0 * c0

이렇게 이어지는 carry out들을 바꾸어 가장 최초에 발생한 c_1으로 재귀적으로 표현할 수 있게 됩니다.

다음 장에 이어서

3.
앞서 언급했듯이 우리가 a - b 연산을 하고 싶다는 상황을 가정할 때, 이는 a + (-b)로 바꿀 수 있기 때문에, -b을 구해야 합니다. 이 때, 목표 b의 모든 비트를 반전시키고 LSB에 1을 더하여, -b = ~b+1라는 방식으로 구할 수 있게 됩니다. 이 과정에서 우리는 b를 반전시켜야 하기 위한 게이트가 필요합니다. 그것이 바로 XOR 게이트입니다. 각자리 비트에 1을 XOR한다면 반전이 되어 1의 보수를 구할 수 있고, 이 값에 1을 더한다면 2의 보수가 나옵니다. 반면, a + b 연산을 하고 싶다면 각 자리 비트에 0을 XOR한다면 반전되지 않기 때문에, 목표하는 값을 구할 수 있습니다.

4.
BCD라 함은 각 자리의 10진수를 4비트짜리 이진수로 표현하여 연산하는 것입니다. BCD코드는 즉 10진법을 사용할 때 한 자리로 표현 가능한 0부터 9까지의 값을 4비트짜리 2진 코드로 표현하는 것입니다. 그렇기 때문에 0인 0000부터 9인 1001까지만 사용하고 그 이상은 사용하지 않습니다. 예를 들어 우리가 9 + 9를 BCD연산을 사용하여 계산할 때 1001 + 1001인데 결과 값은 10010이 나오게 됩니다. 해당 값이 9보다 크기 때문에, 결과 값에 6(0110)을 더합니다. 이를 더한다면 11000이 나오는데 이는 0001 1000이기 때문에 18이 맞습니다.

5.
ALU는 컴퓨터에서 가장 중요한 회로입니다. ALU의 연산에는 크게 4가지 종류가 있다. 레지스터끼리 정보를 옮기는 전송 연산, 수치 데이터에 대한 산술 연산, 비수치 데이터에 대한 논리 연산과 쉬프트 연산입니다. 이중 산술 연산에는 덧셈, 뺄셈, 곱셈, 나눗셈, 보수 등이 있으며 논리 연산에는 and, or, not, xor, asl, asr, rol, ror 등이 있습니다. 하나의 마이크로 연산을 수행하는 시간은 마이크로 사이클 타임이며 이 지표가 CPU의 성능 정도를 알려주는 중요한 지표입니다. 이 ALU를 조작하기 위해 operaton code가 필요한데 예를 들자면, 우리가 b + c를 저장하여 레지스터 a에 저장하고 싶다고 가정하면 MIPS 코드상 add a, b, c가 필요합니다.

6.
MIPS assembly ALU 기본 연산 코드
mips 레지스터 세트에는 여러 종류가 존재하는데, 그 중 $t0 ~ $t7은 임시 변수에 사용되는 레지스터로 이를 이용하여 설명해보겠습니다.
첫 번 째 동작 - 우선 숫자를 load
lw $t0, number1
lw $t1, number2
가정) AND 연산을 하고 싶음 result = number1 and number2
and $t4, $t0, $t1