10주차 예비보고서

전공: 신문방송학과 학년: 4학년 학번: 20191150 이름: 전현길

1. 4-Bit Adder 및 Substractor 이진 병렬 연산 기능에 대하여 조사하시오.

병렬 가감산기는 6주차 예비보고서에서 살펴보았듯이 **전가산기, 전감산기 여러 개를 병렬로 배치하여 4bit 이진수를 계산**하는 회로이다. 4bit뿐만이 아니라, 전가산기와 전감산기를 n개 배치함으로써 n-bit 이진수를 계산할 수 있다.

주의해야 할 점은, 첫 번째 연산(LSB 연산)의 경우 자리올림수나 빌림수가 없기 때문에, **C_{in}, B_{in}에 0을 입력**해야 한다는 점이다.

병렬 가산기/감산기는 자리올림수, 빌림수의 값이 파도가 물결치듯이 한 단위 가/감산기에서 다음 단위 가/감산기로 전송된다고 해서 ripple carry adder/subtractor라고도 부른다. 이와 같은 구현 특성 때문에, 회로의 전파 지연은 단위 회로가 늘어날 때마다 선형적으로 증가한다. 이 때문에 뒤에서 설명한 CLA adder에 비해 훨씬 지연이 크다는 특징을 갖고 있다.

n-bit ripple carry adder

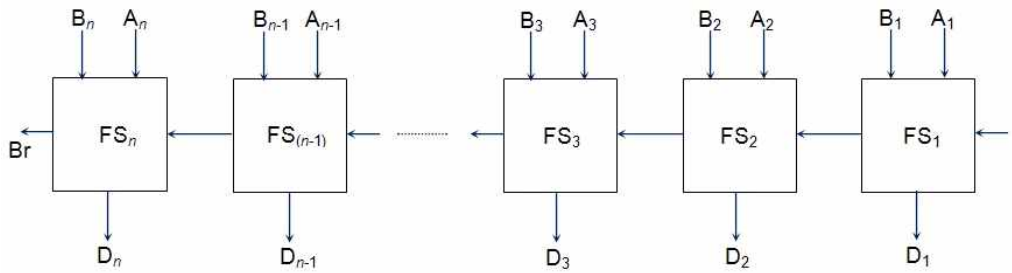


Figure 2 *n*-bit Parallel Subtractor Using *n* full Subtractors

n-bit ripple carry subtractor

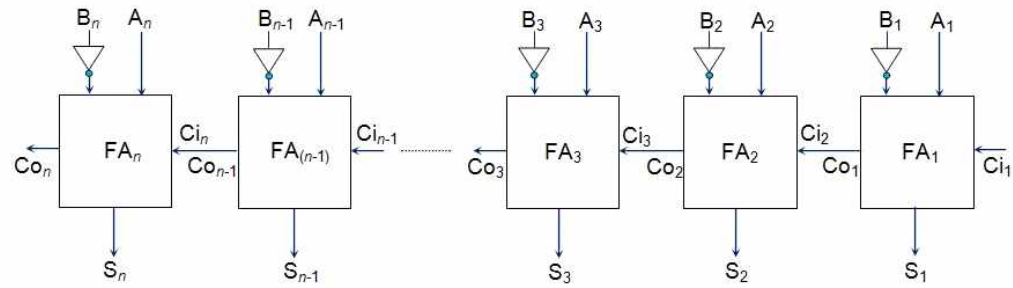
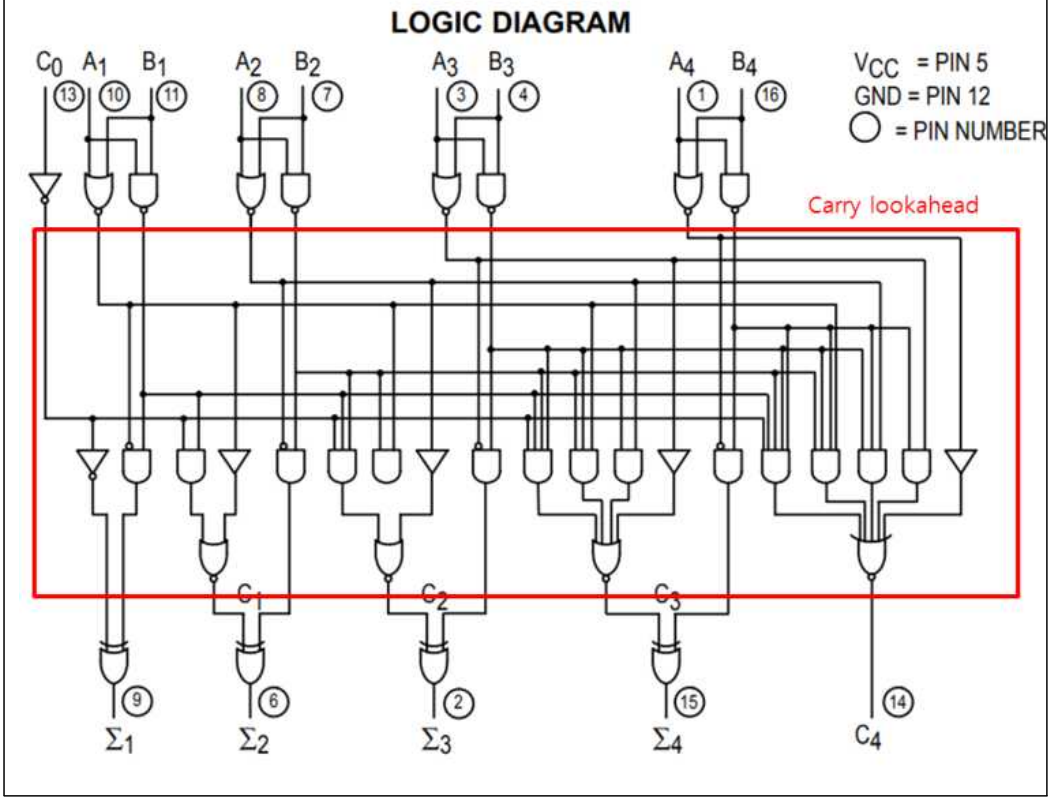


Figure 1 *n*-bit Parallel Subtractor Circuit Realized Using *n* full Adders

2. Look-ahead carry에 대하여 조사하시오.

4-bit carry look-ahead adder



6주차에서 살펴보았듯이, **CLA(carry look-ahead adder)에서는 carry에 대한 식을 k-bit 단위(주로 4bit)로 쪼개서 계산**한다. 이에 따라서 k-bit 계산에 쓰이는 carry를 한 번에 계산할 수 있고, 계산한 carry를 병렬로 한꺼번에 입력함으로써 **n-bit 덧셈에 따른 선형적인 연산 지연(=O(n))을 이론상 O(n/k)까지 줄일 수 있다.**

CLA 계산은 생성(generation)과 전파(propagation)라는 두 원리를 이용한다. 각 이진함수 식은 다음과 같다.

$$G(\text{generation}) \rightarrow G(A, B) = A \cdot B$$

$$P(\text{propagation}) \rightarrow P(A, B) = A \oplus B$$

G는 **자리올림수 생성 함수**로, 기존의 전가산기 연산이 어떻게 되느냐와 무관하게 반드시 carry가 생성되는 경우(A, B가 모두 1인 경우)를 검사한다. P는 **자리올림수 전파 함수**로, 이전 bit의 carry와 A, B 입력이 조합되었을 때 새 carry가 생기는 경우를 검사한다. 논리함수식으로는 AND 연산, XOR 연산

에 해당한다.

G와 P를 이용하면 기존 전가산기의 Sum과 Carry를 다음처럼 치환할 수 있다. S_i와 C_i의 논리식을 함께 확인하면 어떻게 치환되었는지 확인할 수 있다.

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_iC_i$$

$$\text{Sum} = A \oplus B \oplus C_{in} \mid C_{out} = C_{in} \cdot (A \oplus B) + A \cdot B$$

위의 치환 방식을 i = 0에서 i = 3까지 수행하면, C_4까지의 모든 이진함수식을 구할 수 있다. 이 때 C_0은 처음 입력되는 carry 값으로, 반드시 0을 갖는다. 따라서 $C_1 = G_0$이 항상 성립한다. 이를 바탕으로 아래와 같이 CLA 식을 구할 수 있다.

$$C_1 = G_0$$

$$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2 \cdot (G_1 + P_1C_1) = G_2 + P_2 \cdot (G_1 + P_1G_0)$$

$$\begin{aligned} C_4 &= G_3 + P_3C_3 \\ &= G_3 + P_3(G_2 + P_2C_2) \\ &= G_3 + P_3(G_2 + P_2(G_1 + P_1C_1)) \\ &= G_3 + P_3(G_2 + P_2(G_1 + P_1G_0)) \end{aligned}$$

6주차에서 보았고 위의 C_{out} 진리식에서도 볼 수 있듯이, 전가산기는 1bit carry를 계산하기 위해 logical level에서 XOR gate, AND gate, OR gate의 총 3개의 gate를 거친다. 따라서 4번째 carry의 계산을 위해 12개의 gate를 거쳐야 한다. 이에 비해 위의 diagram에서 볼 수 있듯이 CLA adder는 1개의 NAND/NOR gate, 1개의 AND gate, 1개의 NOR gate만을 거치므로 3개의 gate를 거친다.

실질적인 지연은 회로의 특성과 NAND, NOR gate로의 치환 등의 과정에서 달라지겠지만 이론상 k-bit만큼 전송되는 gate가 줄었음을 확인할 수 있다.

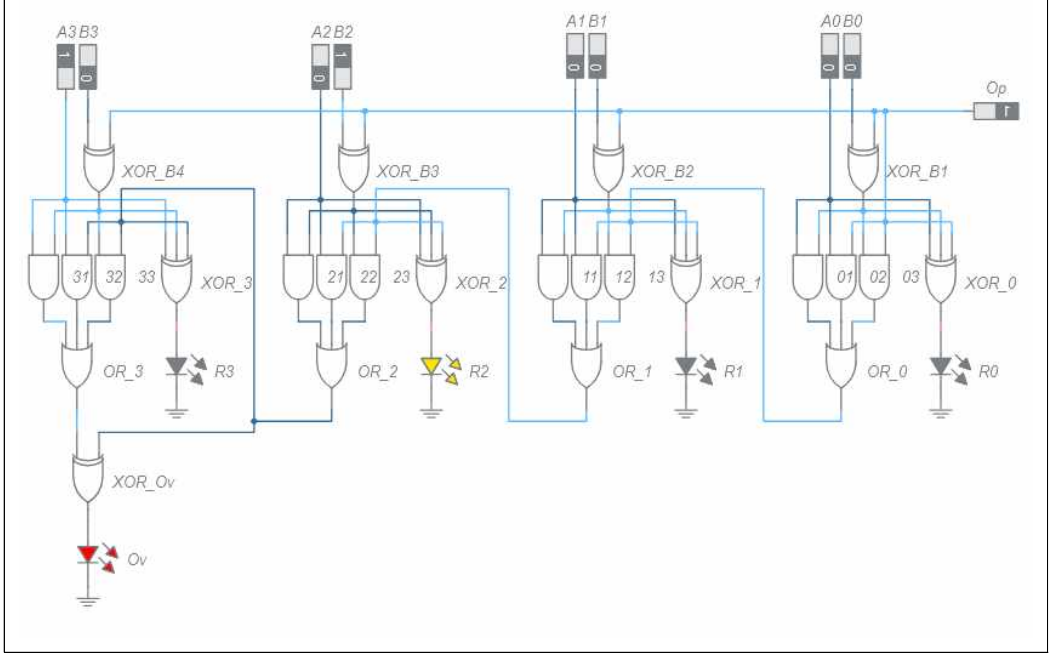
3. XOR을 활용한 2's complement 가감산에 대하여 조사하시오.

4-bit ripple carry adder를 CLA adder로 바꿨듯이, 4-bit ripple carry subtractor를 BLA subtractor로 바꿀 수 있을까? 실질적으로 이러한 회로는 사용하지 않는다. 대신에 2's complement를 이용해 뺄셈을 덧셈으로 바꿈으로써 계산을 모두 덧셈으로 바꾼 뒤에, CLA adder를 통해 계산할 수 있다.

2's complement로 뺄셈을 덧셈으로 치환하는 과정은 다음과 같다. 4-bit 정수 A에서 B를 빼는 상황을 가정한다. 덧셈으로 치환하기 위해 B의 부호를 반대로 바꾸어야 하는데, 이는 **XOR gate를 이용해 각 bit를 1과 XOR 연산하고 LSB에 1을 더하는 것으로 구현**할 수 있다. 반대로 각 bit를 0과 XOR 연산하면 원래 입력이 유지된다.

아래 2's complement 4-bit adder는 **1bit 입력 Op를 통해 덧셈과 뺄셈을 분리해서 구현**한다. Op = 1일 때는 입력 B의 부호가 반대로 바뀌면서 뺄셈으로 동작하며, Op = 0일 때는 값이 유지되며 덧셈으로 동작한다. 이 때 LSB에 덧셈을 하는 부분이 중요한데, LSB의 AND gate에 Op를 추가로 입력하는 부분이 이 부분이다. 이 Op 연산은 4-bit ripple carry adder의 LSB가 갖는 C_{in} 입력(C_0 입력)과 구조가 동일하다.

2's complement 4-bit adder



4. BCD 연산에 대하여 조사하시오.

BCD code는 6주차에서 보았듯이, 8421코드를 기준으로 보았을 때 십진수의 한 자릿수를 이진수 4자리를 통해 표현하는 기법이다. BCD adder는 BCD code에 따라 이진수 bit sequence를 계산한다. 이 때문에 BCD adder에서 0101(5) + 1001(9)의 결과는 1110(14)가 아니라, 0001 0100 (14)로 저장된다.

10진수	2진수	BCD (8421코드)		10진수	2진수	BCD (8421코드)	
0	0000	0000	0000	10	1010	0001	0000
1	0001	0000	0001	11	1011	0001	0001
2	0010	0000	0010	12	1100	0001	0010
3	0011	0000	0011	13	1101	0001	0011
4	0100	0000	0100	14	1110	0001	0100
5	0101	0000	0101	15	1111	0001	0101
6	0110	0000	0110	16	10000	0001	0110
7	0111	0000	0111	17	10001	0001	0111
8	1000	0000	1000	18	10010	0001	1000
9	1001	0000	1001	19	10011	0001	1001

위의 계산을 구현하기 위해, BCD 덧셈은 다음과 같은 계산법을 따른다.

1) BCD 숫자를 4bit 단위로 더한다.

2) 4bit의 값이 9보다 크거나 캐리가 발생하면 계산 값에 0110(6)을 더한다.

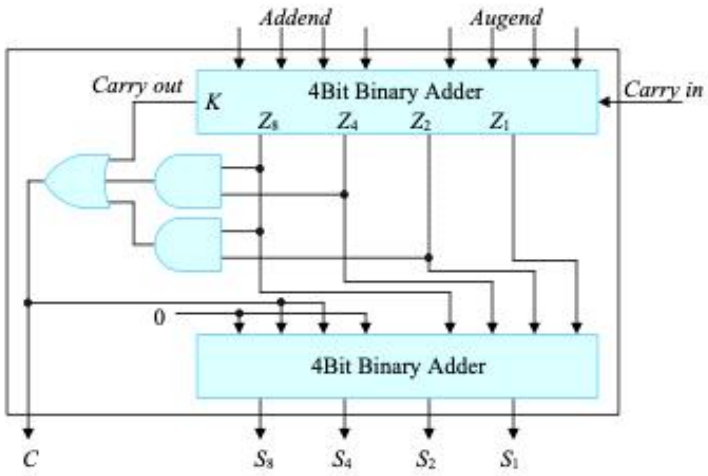
3) 0110을 더했을 때 캐리가 발생하면 자리올림한다.

6을 더하는 이유는 사용하지 않는 6개의 코드의 영역(10~15)을 벗어나서 자리올림하기 위해서이다. BCD 가산기는 위의 계산법을 아래의 진리표를 따라 구현한다.

BCD adder truth table

2진 합					BCD 합					10진값
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD adder circuit diagram

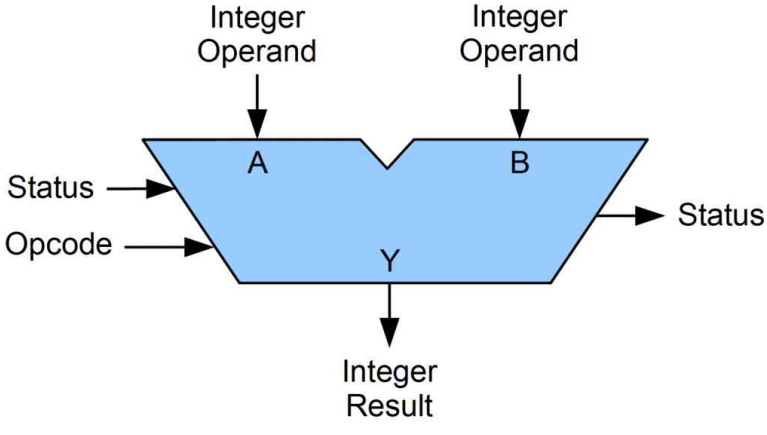


실제 구현은 다음과 같다. 먼저 4-bit 이진수를 4-bit binary adder를 통해 계산해 Z_8, Z_4, Z_2, Z_1을 출력한다. binary adder의 출력 결과를 확인하고, Z_8이 1이고 Z_4가 1이거나(값이 12-15), Z_8이 1이고 Z_2가 1이면(값이 10-11) 보정이 필요한 값이므로 캐리 출력을 활성화하고 값을 변환한다.

5. ALU의 기능에 대하여 조사하시오.

ALU(Arithmetic Logic Unit)는 CPU의 주요 부품 중 하나로서, 메모리 주소 계산, 비교 연산, 덧셈 및 뺄셈 연산, AND, OR 연산, 곱셈 및 나눗셈 연산 수행에까지 이용되는 장치다. 다양한 기능을 가지고 있는 만큼 회로의 비용도 값비싼 편이다.

symbol of ALU



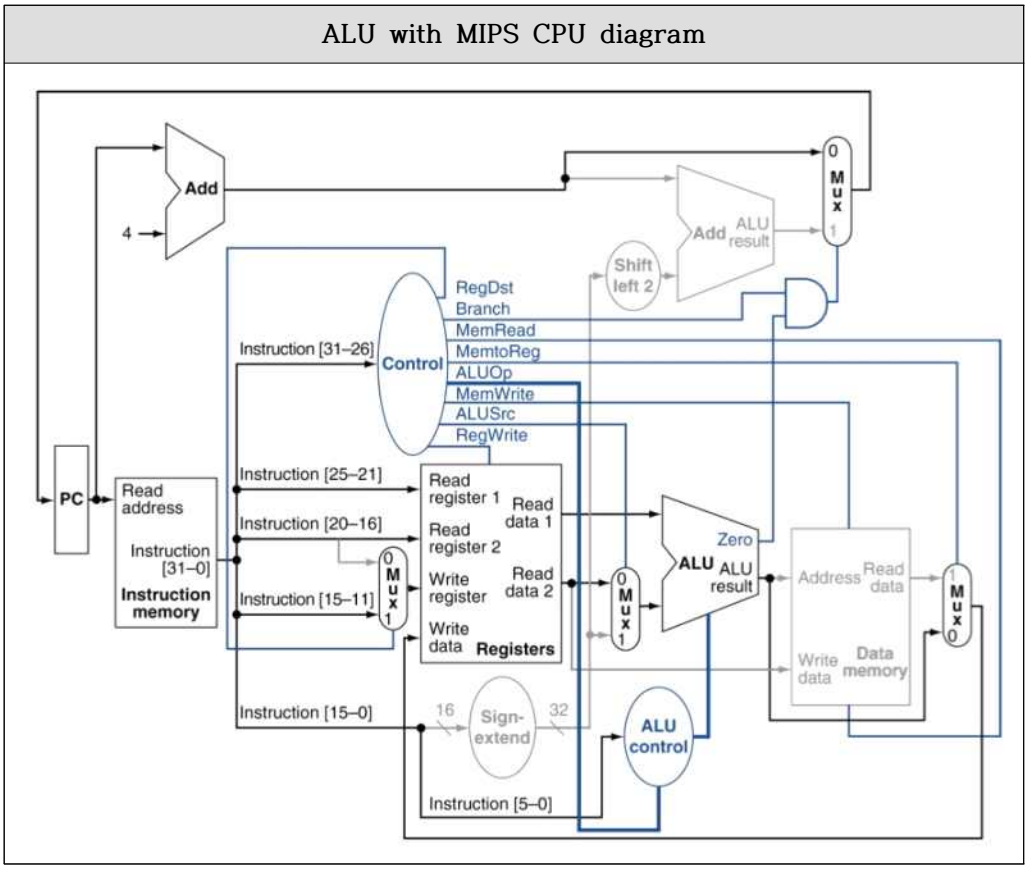
ALU은 위처럼 쪼개진 사다리꼴 형태의 symbol로 표현된다. 위 diagram에는 두 개의 integer operand를 갖는다고 표현되어 있지만, 실수 연산에 사용되기도 한다. opcode는 ALU를 덧셈, 곱셈, 나눗셈, 비교, AND/OR/NOR 연산 등의 다양한 연산 중 어떤 연산에 사용할지를 조작하는 제어 필드이다. ALU는 opcode를 통해 결정된 연산에 따라 두 피연산자를 입력받아 연산 결과를 출력한다. 연산 결과뿐만이 아니라 상태 레지스터에 보낼 값(=status)을 출력하기도 한다.

일반적으로 MIPS CPU에서는 **명령어의 opcode 필드**가 **제어 유닛**에 전달되면 이를 조합하여 **2bit ALUOp를 출력**하고, **ALU 제어 유닛**은 이 **2bit ALUOp 신호와 명령어의 funct 필드**를 조합하여 **4bit ALU 제어신호**를 출력한다. ALU는 이 4bit 신호에 의해 수행할 연산을 결정하게 된다.

6. 기타 이론

MIPS CPU에서 사용되는 ALU의 CPU diagram, ALU 제어 신호의 조합식과 ALU 제어 신호에 따라 결정되는 연산의 종류는 다음과 같다.

ALU with MIPS CPU diagram



ALU control signal

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

operation type

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR