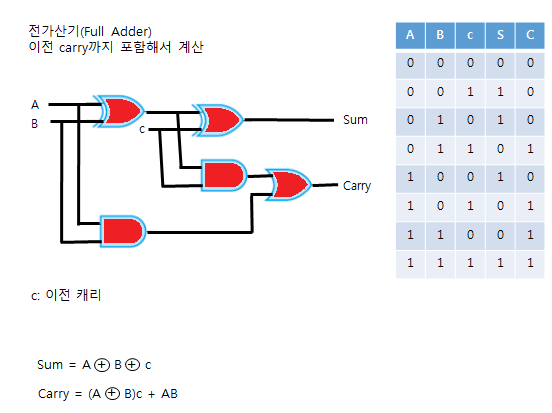
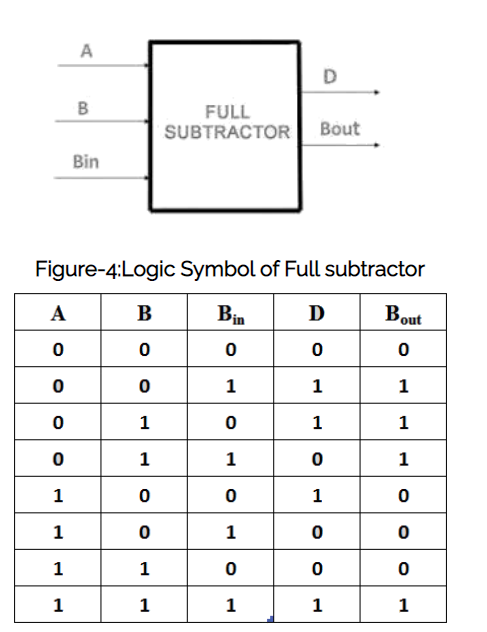
10주차 예비보고서

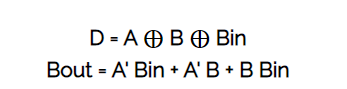
전공: 컴퓨터공학과 학년: 3학년 학번: 20191612 이름: 윤기웅

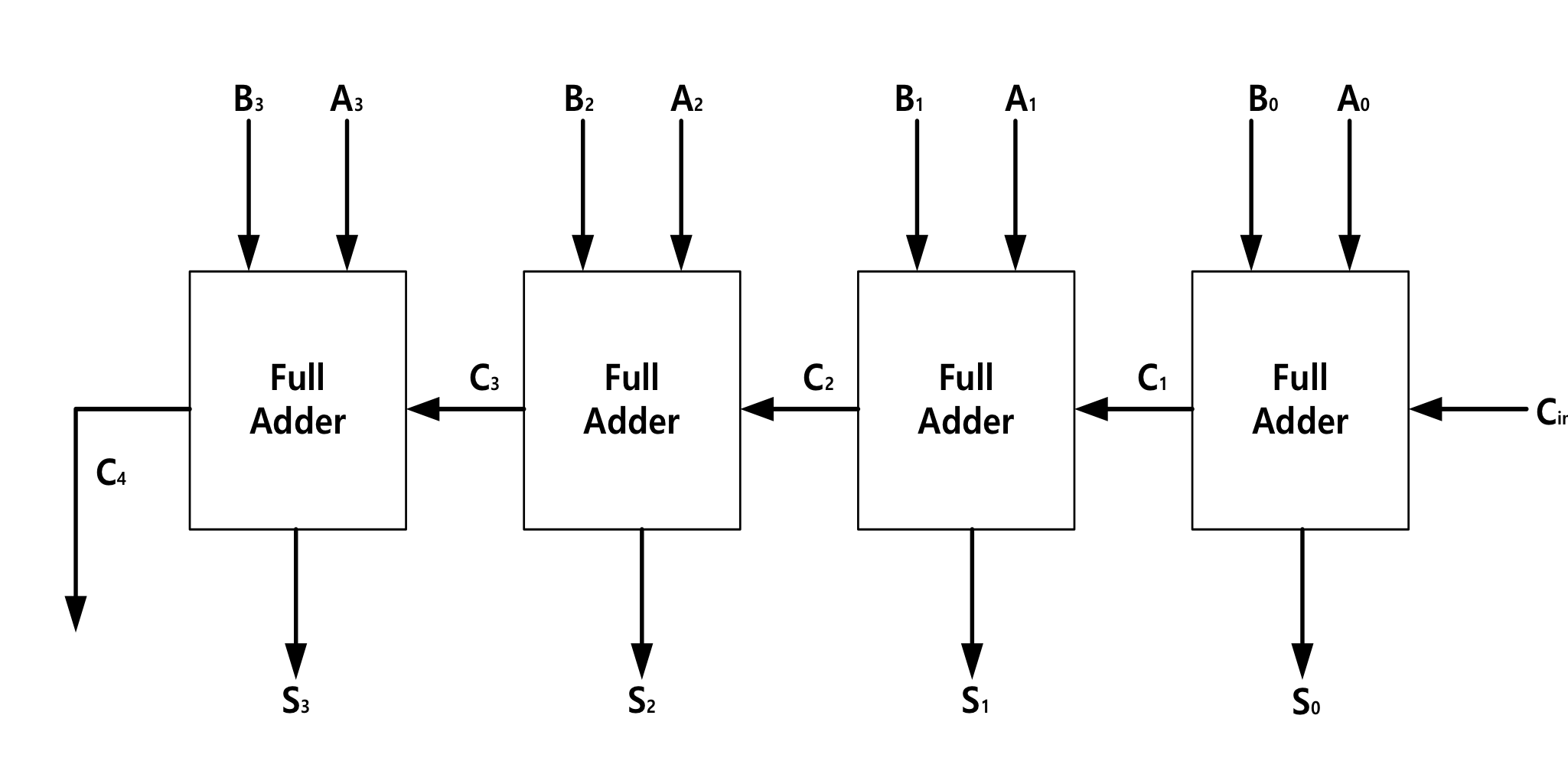
1.

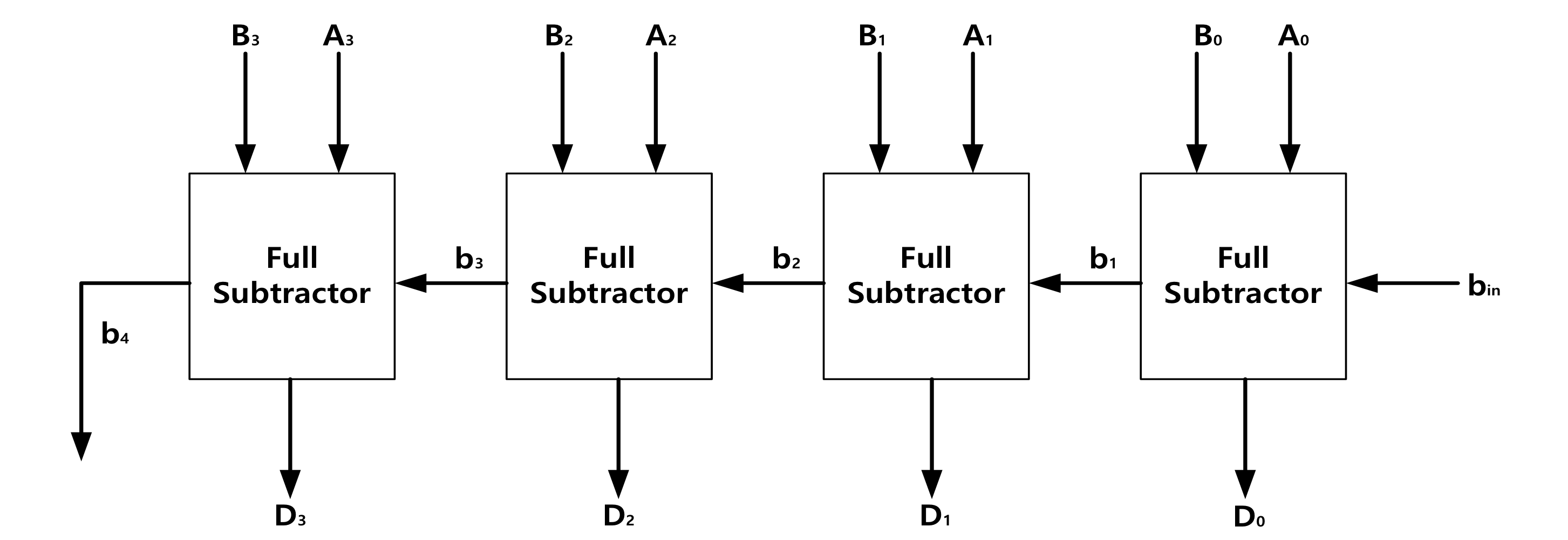
  4- Bit Adder와 substractor를 병렬 연산으로 수행하는 경우에 주어지는 N개의 adder 혹은 substractor를 연결하면 N 비트의 덧셈 혹은 뺄셈의 연산이 가능해진다. 아래의 그림은 adder, substractor의 모습을 나타낸 것이다. carry가 발생하면 그 다음 비트로 넘겨주는데 이 때 낮은 자리의 캐리를 받아야 계산을 할 수 있어서 시간의 지연이 다른 계산에 비해 더 많이 발생한다. 1개의 full adder에서는 S와 C가 결과로 나온다. 아래의 전가산기 그림은 진리표와 수식을 보여주고 있다. 그 아래의 그림은 전감산기의 진리표와 감산기의 결과값인 B와 D의 수식을 보여준다.





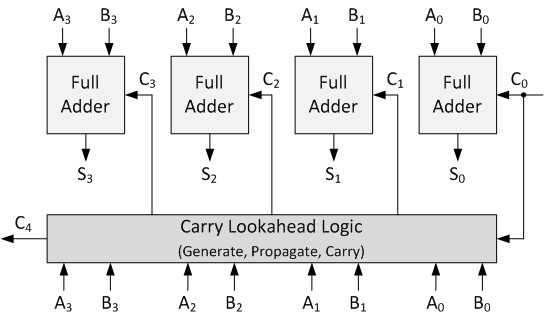


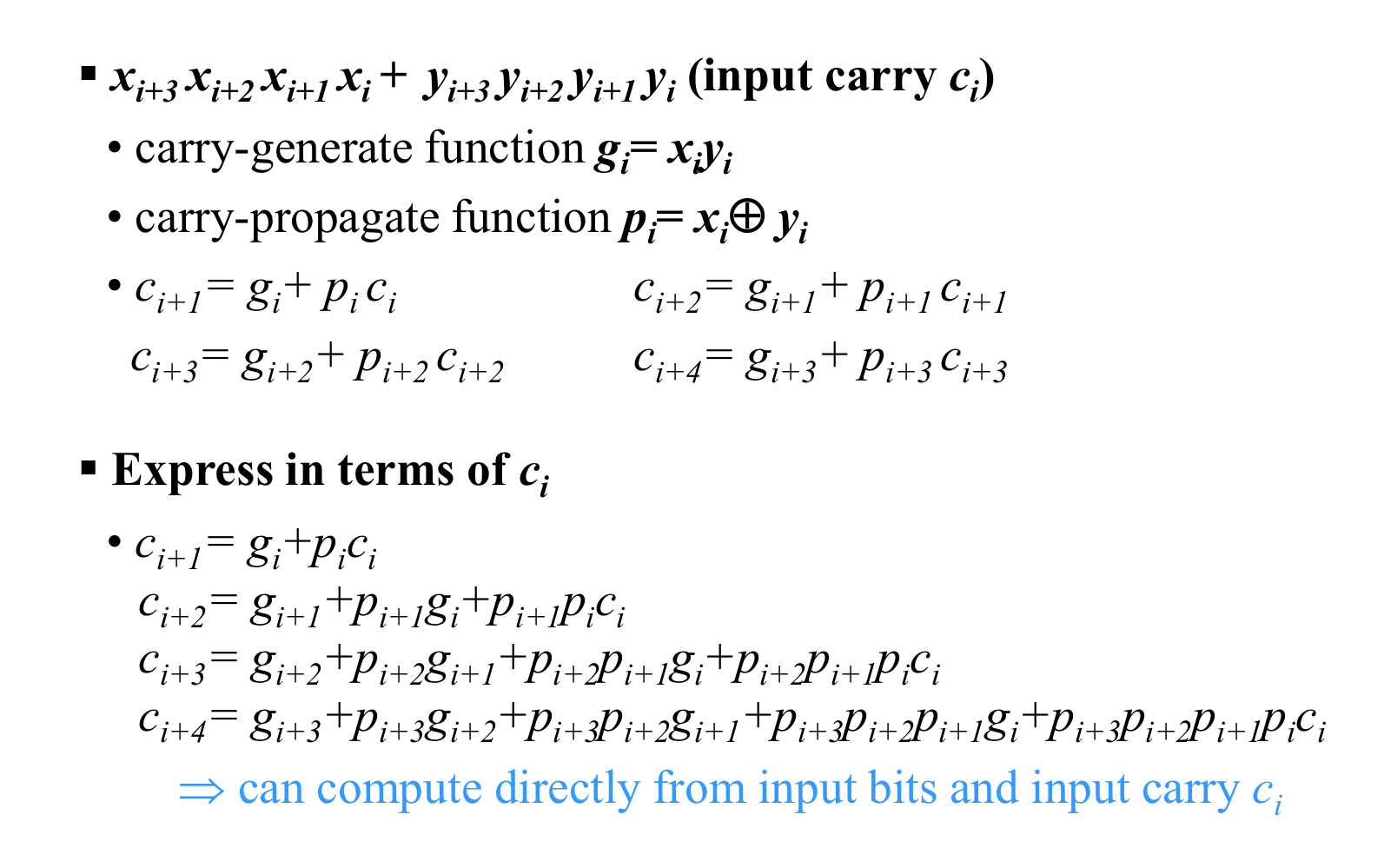




2.

Ripple carry adder를 더 효과적으로 시행하기 위한 회로이고 연산의 속도가 비교적 빠르다. Rca의 경우는 full adder를 계속 이어 붙여서 만든 가산기라서 그 full adder의 개수가 증가할수록 연산의 지연정도가 높아지고 계산의 속도가 매우 느려진다. ripple이라는 단어의 의미 그대로 올림 수가 가장 낮은 자리의 수에서 시작해서 물결처럼 올라와서 마지막의 Cin에 와야 연산이 끝난다. 하지만 Look ahead carry는 동작방식이 다르다. Generate, propagate 두 신호를 이용하고 각각은 Gi, Pi로 표현하며 Gi = Ai \* Bi , Pi = Ai + Bi 로 정의가 가능하다. 그리고 Ci= Gi+Pi\*Cin 으로 나타낼 수 있다. 만약 4비트의 look ahead carry가 있는 경우를 가정해보면 각 단계의 연산에서 C1, C2, C3 , Cout이 다음과 같이 연산이 된다. C1=G0+P0Cin, C2=G1+P1G0+P1P0Cin, C3=G2+P2G1+P2P1G0+P2P1P0Cin, Cout=G3+P3G2+P3P2G1+P3P2P1G0+P3P2P1P0Cin

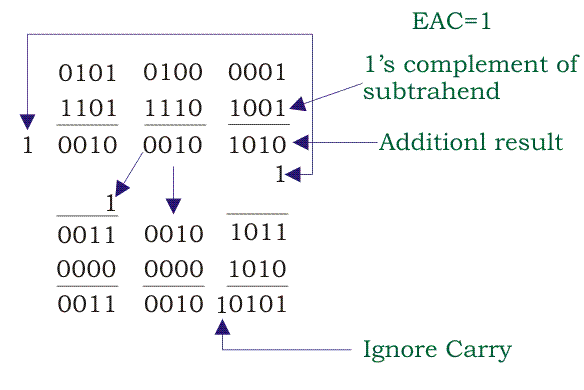




3.

  complement는 보수라는 말로 번역 가능함으로 2's complement는 2의 보수이고 보수는 보충해주는 수라는 의미를 갖고 있다. 10진수를 예로 들었을 때 4의 보수를 구하려고 하면 더해서 10이 되어야 해서 6이 4의 보수이다. 2진수 혹은 2진법의 보수를 구하게 되면 그 원래 수와 보수를 더해서 가장 왼쪽의 비트(가장 높은 자리의 수)가 1이 되어야 하고 나머지 자리는 0이 되어야 한다. 그래서 2의 보수를 구하는 공식이 있다. 예를 들어서 00000111이라는 7을 표현하는 2진수가 있다고 가정해본다. 먼저 비트의 반전을 시켜서 0은 1로 1은 0으로 바꿔준다. 그러면 11111000이 되는데 여기에 1을 더한다. 그러면 11111001이 되기 때문에 -7이 되고 이 두 개의 이진수를 더하면 100000000가 되고 십진수로 0이 된다.

4. Bcd는 binary coded decimal의 줄임말로 우리가 사용하는 10진수를 2진수로 표현하기 위한 방법 중 하나로 bcd 코드를 사용하면 된다. 4자리의 이진수를 기준으로 볼 때 0부터 9까지 표현하면 되니 1010부터 1111까지의 6개의 숫자는 사용하지 않게 된다. 그래서 두 2진수의 합이 1010 이상인 경우 0110을 더해준다. 4비트의 이진수가 16가지이고 10진수는 10개라서 올림을 표현하려면 6을 추가로 더해가지고 표현해주어야 한다. 만약 58과 67을 bcd 덧셈을 통해서 표현한다고 가정해보면 01011000+01100111=10111111이 되는데 결과값을 4자리씩 나누어 보면 양편 모두 10보다 크기 때문에 양쪽 모두 0110을 더해줘야 한다. 그리 하게 되면 결과는 0001 0010 0101이 된다. 이를 10진수로 바꾸면 125가 되니까 연산이 잘 됨을 볼 수 있다. Bcd 연산 중에서 뺄셈도 만들 수가 있는데 뺄셈은 뺄 수를 먼저 비트의 반전을 시켜주고 두 수를 덧셈해준다. 이후 가장 높은 자리에서의 올림이 발생하면 가장 낮은 자리에 더해준다. 이후에 1010을 더해주면 된다.



bcd연산은 10진수의 형태와 비슷하고 하드웨어의 알고리즘 구현이 비교적 단순하면서도 아스키코드보다 차지하는 용량이 작다는 점이 장점이다.

5.

Cpu의 산술 논리 연산 장치(Arithmetic Logic Unit)을 의미하고 ALU가 어느 정도의 크기인지에 따라서 cpu의 크기가 정해진다. 주어지는 논리 연산과 산술 연산을 수행하게 되고 이외에도 시프트 연산이나 비교 연산 등을 수행하게 된다. 산술연산의 4가지 연산 중에서 뺄셈, 곱셈, 나눗셈은 덧셈을 이용해서 구현이 가능하다. 뺄셈은 덧셈 연산에서 뺄 수의 부호를 변형해서 더하면 된다. 곱하기는 덧셈을 계속 반복함으로써 작동이 가능하다. 나눗셈은 덧셈을 변형한 나눗셈을 반복해서 작동한다. 논리 연산은 and, not, or, xor 을 구현할 수 있고 비트의 이동으로 시프트 연산을 실행 한다. ALU 연산의 과정은 다음과 같다. 먼저 컴퓨터의 메모리나 레지스터에서 데이터를 받아오고 연산을 처리한 이후 그 결과값이 다시 레지스터나 메모리에 저장된다. 즉 메모리나 레지스터 사이의 데이터 전송을 도와주는 역할을 ALU가 한다.

6.

Cpu에 대해서 알아보자 : 중앙처리 장치는 컴퓨터의 핵심 장치 중에 하나로 명령어의 해석과 연산을 처리하는 역할을 맡고 있다. Cpu는 제어 장치, 연산 장치(ALU), 레지스터 그리고 이들을 연결해주는 버스로 구성되어 있다. 제어장치는 control unit이라고도 불리며 컴퓨터의 작동을 지시하고 기억장치에서 명령을 갖고와서 해석하여 명령을 수행하기 위한 신호제어를 다른 장치들로 보내준다. 제어장치는 pc(program counter), 부호기 , 명령 레지스터 등으로 이루어진다. 연산장치는 명령어 실행을 수행하기 위한 연산을 수행하고 누산기, 보수기 , 가산기 등으로 이루어진다. 레지스터는 중앙처리장치 안에 있는 아주 작은 크기의 속도가 빠른 기억 장치이다. 명령어 실행에 필요한 정보 혹은 데이터를 임시로 저장하고 컴퓨터의 메모리 계층에서 제일 빠르기 때문에 최상위 위치에 존재한다고 볼 수 있다.

Cpu의 연산은 크게 4단계로 구성된다고 볼 수 있는데 fetch, decode, execute, writeback의 순서로 진행된다. Fetch는 프로그램 카운터가 지시하는 명령어를 cpu로 갖고 오는 과정이다. Decode는 명령을 해석해서 명령어의 종류와 대상을 정하고 execute 단계에서는 해석에 따라서 연산을 수행한다. Writeback은 명령을 따라서 완성한 데이터를 메모리에 적는다.