10주차 예비보고서

전공 : 컴퓨터공학과 학년 : 3학년 학번 : 20211558 이름 : 윤준서

**1. 4-bit Adder / Subtractor 이진 병렬 연산 기능**

Half / Full Adder와 Half / Full Subtractor를 활용해서 병렬 연산기를 만들 수 있다. 즉 Full Adder와 Full Subtractor를 각각 n개씩 병렬로 연결하면 n-bit Adder / Subtractor 이진 병렬 연산 회로를 만들 수 있다. 4-bit Adder / Subtractor 이진 병렬 연산 회로는 각각 두 개의 4-bit 2진수의 덧셈과 뺄셈 연산을 수행하며, Full Adder와 Full Subtractor를 각각 4개씩 병렬로 연결한 형태다.

**2. Look Ahead Carry**

n-bit Full Adder의 연산을 수행할 때 Ripple-Carry Chain 회로 방식은 순차적으로 bit를 더한 후 carry bit를 다음 bit에 전달해서 계산을 진행한다. 이는 bit의 수가 많아질 수록 연산 시간도 증가하는 단점이 있다. 이를 보안하기 위한 방식이 Look Ahead Carry 회로다.

Look Ahead Carry 회로는 모든 bit 간의 계산에서 도출되는 carry bit를 동시에 계산하여 넘기는 방식으로 연산 시간을 줄일 수 있다. 이때 carry-generation function과 carry-propagate function이 사용된다. 두 n-bit 2진수의 각 자리 수가 일 때 두 function의 연산은 다음과 같다.

Full Adder에서 의 공식을 로 나타낼 수 있다.

모든 는 로 표현이 가능하다는 특징이 있다. 이를 통해 carry bit를 동시에 계산하여 시간을 단축시킨다.

**3. XOR을 활용한 2's complement 가감산**

특정 양수에서 음수를 빼는 경우, 이를 양수와 양수를 더하는 방식으로 변형할 수 있다. 반대로 특정 양수에서 양수를 빼는 경우, 이를 양수와 음수를 더하는 방식으로 변형할 수 있다. 이처럼 부호 변환을 통해 두 수 간의 연산을 덧셈으로 표현할 수 있는데, n-bit 2진수 간의 연산에서 2's complement을 사용해 부호 변환이 가능하다. 2's complement를 통해 특정 수의 부호를 변환하는 과정은 다음과 같다.

1. 모든 bit의 수를 반대로 바꾼다. (0 -> 1, 1 -> 0)

2. 1을 더한다.

예시로 숫자 6을 다뤄보면, 6의 2진수 형태는 0110이다. 위의 과정을 따라 모든 bit를 변환하면 1001이다. 여기에 1을 더하면 1010이다. 이는 signed 형태로 -6을 뜻한다.

이때 XOR gate를 활용한 부호 변환이 가능하다. 첫번째 과정은 특정 2진수에 n-bit의 자리수가 모두 1인 2진수를 XOR을 취하면 특정 2진수의 모든 bit의 수가 바뀐다. 그리고 마지막에 1을 더하면 부호 변환이 완료된다. 즉 0110 ^ 1111 = 1001이고, 여기에 1을 더하면 이전과 같은 1010이 도출된다. 이처럼 논리 회로에서 XOR gate를 통해 2's complement 가감산 연산을 할 수 있다.

**4. BCD 연산**

BCD code는 0을 뜻하는 0000부터 9를 뜻하는 1001까지만 사용한다. 즉 나머지 1010, 1011, 1100, 1101, 1110, 1111는 don't care 값으로 사용하지 않는다. 따라서 연산을 진행할 때 추가적인 작업이 필요하다.

예시로 숫자 5와 8을 더할 때, 이는 BCD code로 0101 + 1000으로 나타낼 수 있다. 하지만 그 결과는 1101로, don't care 값에 포함된다. 범위를 벗어나는 문제를 해결하기 위해, don't care 값의 개수인 6만큼 추가로 더하는 작업을 수행한다. 그럼 1101 + 0110 = 0001 0011 이 도출되는데, BCD code는 하나의 4-bit 2진수를 각 한 자리의 숫자로 취급하기 때문에 0001 0011은 13을 뜻한다.

즉 각 한자리의 숫자는 0부터 9까지만 필요하므로 10부터 15의 값을 가지는 1010부터 1111은 don't care 값으로, 사용하지 않는 것이다.

**5. ALU**

ALU는 컴퓨터가 실행할 산술 연산과 논리 연산을 수행하는 디지털 회로다. 이는 프로그램의 명령어에 따라 실행되며 그 결과는 레지스터 또는 메모리에 저장된다.

ALU는 연산기, 논리 연산기, 시프트 유닛, 상태 레지스터 등으로 이루어져 있다. 연산기는 산술 연산을, 논리 연산기는 논리 연산을 수행한다. 시프트 유닛(shifter)은 데이터의 비트를 왼쪽이나 오른쪽으로 shift 연산을 수행한다. 상태 레지스터(flag register)는 연산 결과에 대한 정보를 저장한다.

ALU의 동작 과정은 다음과 같다.

1. 제어 장치에서 연산 명령을 전달받는다.

2. 레지스터나 메모리에서 입력 데이터를 전달받는다.

3. 제어 신호에 따라 연산을 수행한다.

4. 연산 결과를 레지스터나 메모리에 저장한다.

ALU는 CPU에서 연산의 중심이 되어 컴퓨터의 처리 속도에 핵심적인 영향을 주기 때문에, CPU의 성능은 ALU의 효율성에 따라 달라진다고 할 수 있다.

**6. 기타 이론**

ALU에서 flag register의 종류는 크게 두 가지로 나뉜다. 하나는 상태 플래그(statuc flag), 하나는 컨트롤 플래그(control flag)다. 상태 플래그는 연산 결과에 대한 정보를 저장하고, 컨트롤 플래그는 CPU의 동작상태를 제어한다.

상태 플래그의 종류는 다음과 같다. 기본값을 0으로 가진다.

Carry Flag(CF) : unsigned overflow가 발생하면 1의 값을 가진다.

Zero Flag(ZF) : 연산결과가 0이면 1의 값을 가진다.

Sign Flag(SF) : 연산결과가 음수이면 1의 값을 가진다.

Overflow Flag(OF) : signed overflow가 발생하면 1의 값을 가진다.

Parity Flag(OF) : 연산결과 값이 1인 bit의 수가 짝수개면 1의 값을 가진다.

Auxiliary Flag(AF) : 하위 4-bit에 unsigned overflow가 발생하면 1의 값을 가진다.

컨트롤 플래그의 종류는 다음과 같다.

Trap Flag(TF) : 값을 1로 설정하면 하나의 명령어를 수행하고 , 0x01 interrupt (the debug interrupt)를 발생시킨다. 0x01 interrupt를 발생시키면 debuger가 해당 프로세스를 attach 할 수 있는 상태가 된다.

Interrupt enable Flag(IF) : 값을 1로 설정하면 외부 interrupt는 무시되고 , 0으로 설정하면 외부 interrupt를 받아들인다.

Direction Flag(DF) : 데이터를 다른 메모리에 복사할 때 방향을 결정한다. 값이 0으로 설정하면 데이터 주소를 가리키는 register가 증가하고, 1로 설정하면 데이터 주소를 가리키는 register가 감소한다.