HW#2 2017069598 박상지

(\* 본 레포트에 과제 1번 2번 해당하는 내용이 담겨 있습니다. \*)

1.Write a summary report in Topics 0-1 and 0-2

1. 컴퓨터의 탄생(Invention of Computers)

새로운 지식의 창조로 과학적 성취를 이루었으면 그 중에 가장 돋보이면 성취는 19세기의 Boole이다. 또는 자동장치라는 실용적 도구를 개발하였고 마지막으로 트랜지스터(Transistor) 발명으로 구현 기술이 발전되었다.

-명제논리학(Propositional Logic): 참과 거짓으로 선언할 수 있는 명제로 x + 3 = 5와 같은 명제는 미지수의 개념이 들어가 있으므로 명제논리학에 명제가 아니다. 명제는 p, q로 이용하여 반복적으로 불(Boole)의 명제들을 이용하여 반복적으로 적용할 수 있다. 더 확장된 버전인 first-order 논리에서는 미지수(변수)의 개념이 다루어 진다.

-불(Boole): 인간의 논리적 생각은 명제 그리고 명제들을 AND, OR, NOT, IF로 결합함으로써 표현할 수 있다.

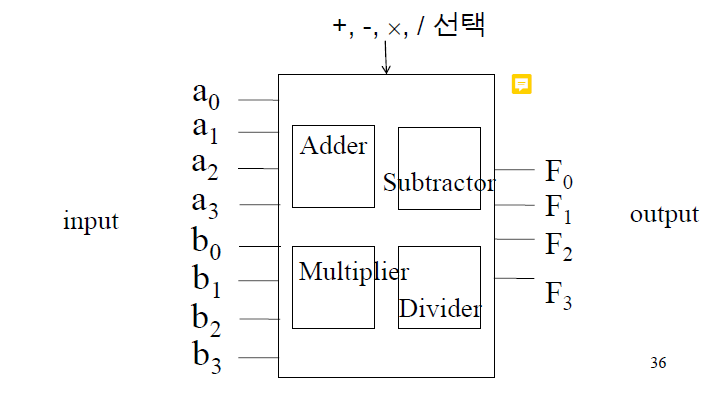
-불(Boole)은 컴퓨터 과학(computer science)에 어떤 영향을 끼쳤나?

* AI: 지식(명제)의 저장 및 논리적 추론할 수 있는 기계를 개발. 컴퓨터가 지능이 있는 기계로 개발할 수 있지 않을까? 라는 궁금증에서 메모리(memory)에 사람의 지식과 논리적인 추론 능력을 담으면 지능이 기반이 된 기계를 만들려 했다.
* 불 대수학(Boolean Algebra): 사칙연산을 사용하지 않고 AND, OR, NOT을 사용하여 0,1을 참과 거짓으로 사용하여 이진 값들로 표현하였다. 사칙 연산(Arithmetic) 과 컴퓨터에서 핵심 역할을 하는 CPU가 계산과 논리적인 처리할 수 있는 ALU를 할 수 있다.
* 자동장치 설계(Automata Design): 댐의 수문에 대한 간단한 자동장치를 설계를 할 때에도 해당 작용을 명제로 다루며, 불(Boole)의 명제들을 사용하면 논리적인 다이어그램(logic diagram)을 쉽게 구현할 수 있다.

1. 디지털 논리 설계(Digital Logic Design)

입력에 의해서 정확하게 결과 값이 결정 되어있는 조합 논리(combinational logic)이며 조합 논리는 AND, OR, NOT 게이트로 이루어진다.

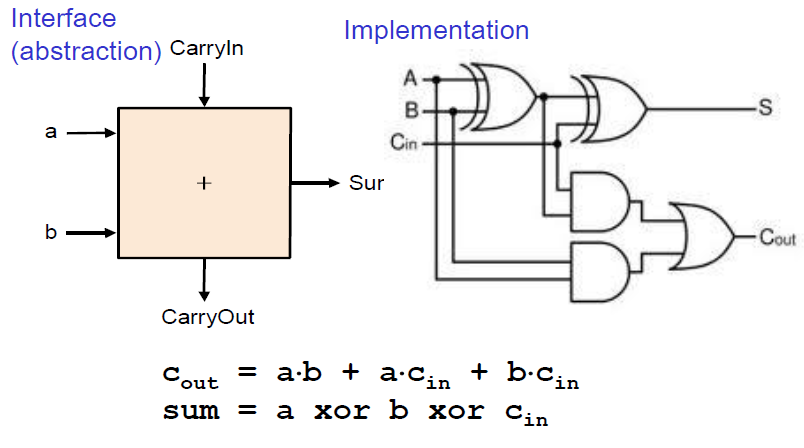
* 4 – bit adder: 4비트 두 개의 합이 입력되었을 때 결과 값으로 4비트가 나온다. 하지만 여기서 중요한 것은 4-bit adder일 경우에는 256 열이 truth table에 주어지기 때문에 간단한 함수로 표현하기 복잡하며, 계산도 까다롭다. 그러므로 복잡한 것을 간단하게 표현하는 게 중요하다. (imagine ALU design without abstraction)



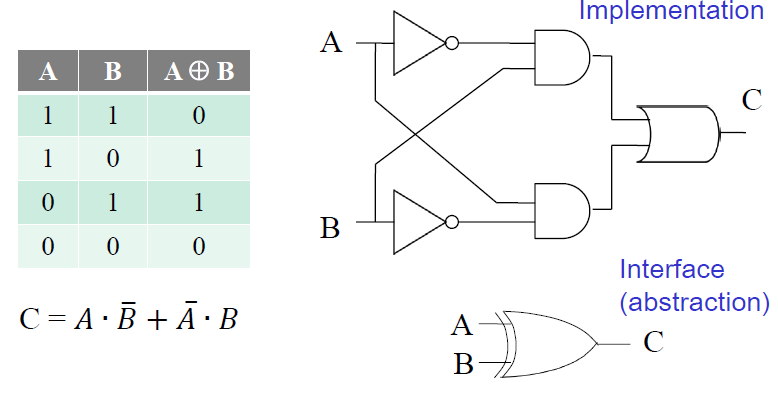
* 추상화(Abstraction): 추상화에서는 인간이 사용하는 자연어, 강조하고 싶은 것을 강조하고 필요한 것만 언급하는 Selective ignorance의 개념, 마지막으로 컴퓨터 과학과 엔지니어링안에 사용된다.

추상화라는 개념은 근본적으로 사용법(Interface)와 구현(Implementation) 개념이 담겨있다. 컴퓨터 과학에서 두가지 중요한 인터페이스를 보자면 컴파일러(Compilers) 와 프로그래밍 언어(ex>c, c++, java)가 있는데 여기서 컴파일러는 구현에 해당하고 c와 같은 프로그래밍 언어는 고급 언어 사용법 즉, interface에 해당한다.

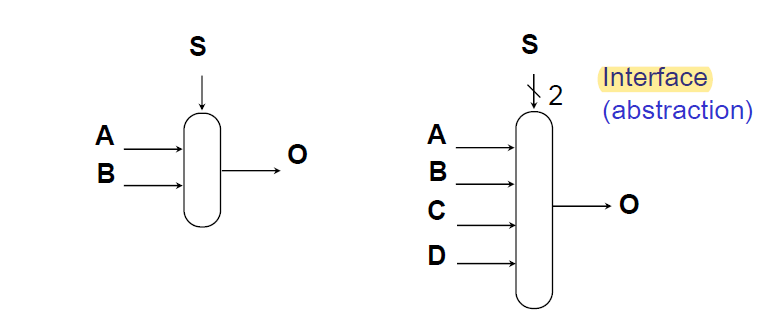
* 1-bit Full Adder Design: a, b, Carry In은 입력 값에 해당하며 각각 1bit를 갖고 있으며, 결과값으로는 sum, Carryout이 있다. sum에서는 입력 값 중에서 하나의 입력 값만 true일 경우에 true라는 결과 값이 나오고, 또는 전체 입력 값이 모두 true일 경우에 true라는 결과 값이 나온다. Carryout은 입력 값이 a, b, c에서 두 개의 입력 값이 true이면 결과 값으로 true가 나온다.



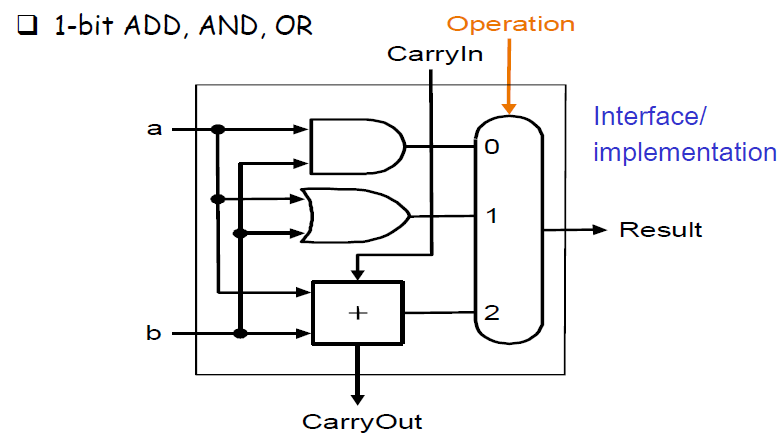
* XOR(Exclusive-OR) Gate: 두 개의 입력 값이 존재하며, 두 입력 값이 서로 다를 경우에 결과 값이 true가 된다.



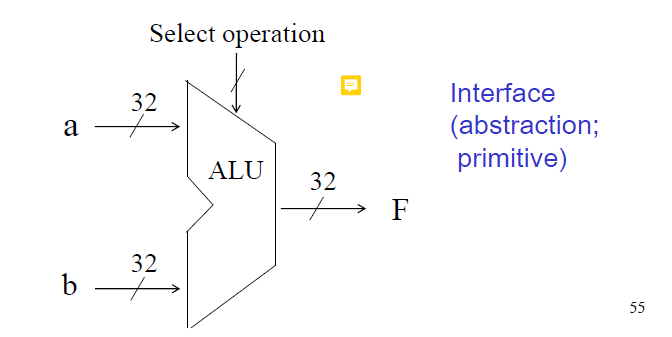
* Multiplexors(data selectors): A, B 입력 값과 s(Signal) 입력 값이 있으며, S가 0이면 A가 결과 값으로 나오고, 1이면 B가 결과 값으로 온다. MUX는 구현(implementation)이 매우 복잡하지만 사용자는 사용법(I/f)만 알면 되기 때문에 문제가 없다.



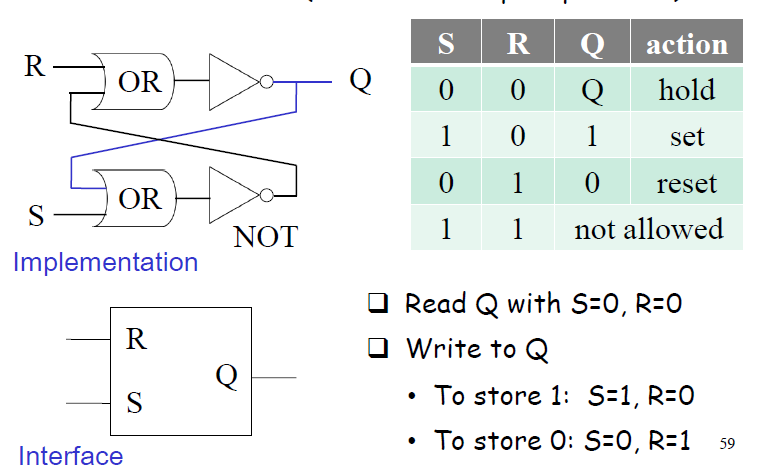
* 1-bit ALU Design: a, b 두 개의 입력 값에 따라서 AND, OR NOT으로 1-bit ALU를 설 계를 할 수 있다.



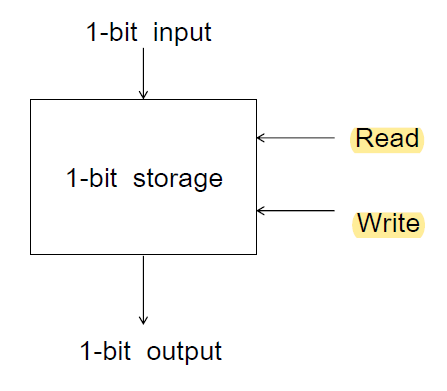
* 32-bit ALU Design: 32-bit을 입력을 받고 32-bit 결과 값을 내보낸다. 어떤 연산을 사용할 것인지는 해당 operation에서 설정해 준다. operation에서는 사칙 연산이 가능하며 AND, OR, NOT으로 논리적 계산을 한다.



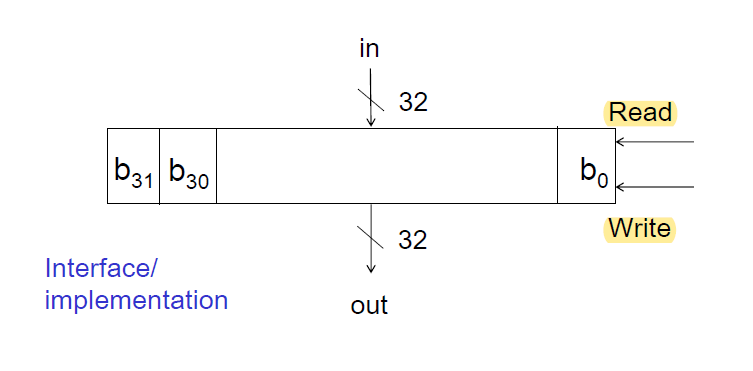
* SR Flip-Flop: 1-bit storage이라 불리며 두 개의 안정된 상태를 갖고 있다. S, R이 둘 다 false 값을 가지면 과거의 상태를 유지하며 S가 true일 경우에 Q에 true를 반대로 R이 true이면 Q에 false을 저장한다.



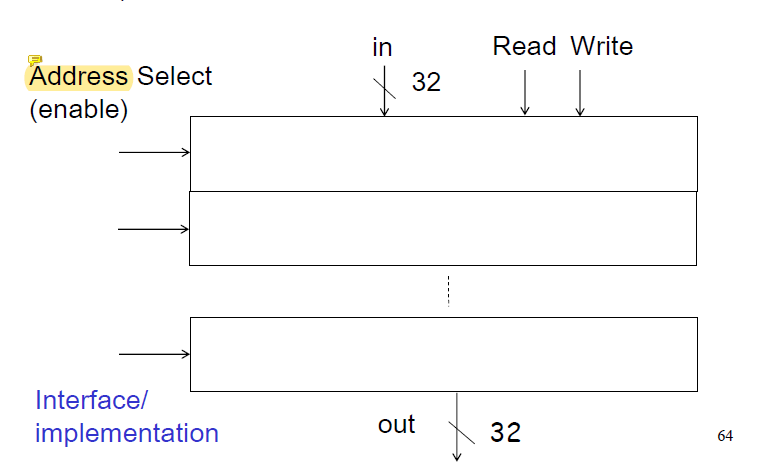
* 1-bit Storage: 입력 값과 결과 값이 1-bit로 들어오고 나오며, 읽고 쓰는 연산이 가능하다.



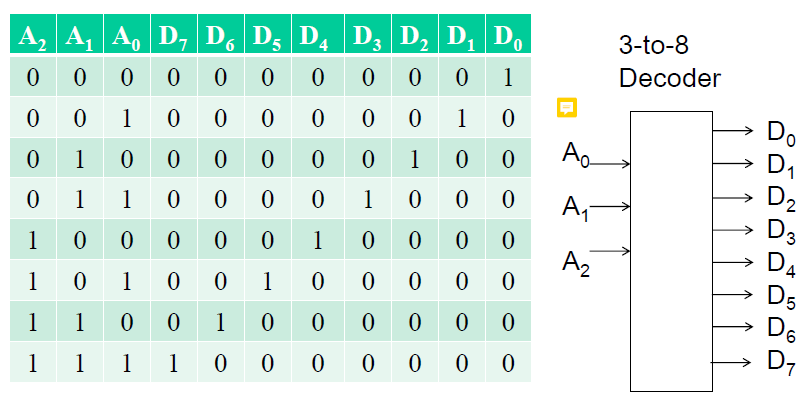
* 32-bit Storage: 1-bit storage를 32개의 병렬로 정렬되어 있다. 마찬가지로 읽고 쓰는 연산이 가능하다.



* Main Memory in 32-bit Computer: 32-bit storage를 포개 놓았으며, 해당 storage는 번지 값이 존재하며 메인 메모리에서는 Address Select로 몇 번지에 있는 데이터 값을 읽고 쓰는 연산이 가능하다.



* 3-to-8 Decoder: 입력 값으로 address의 3-bit을 받으며 결과 값은 입력 값의 000 ~ 111에 따라서 순서데로 true로 신호를 받는다.



1. 더 깊이 다루는 추상화라는 개념(more on Abstraction) – 복잡함을 어떻게 다룰까?

* C programming

C 언어는 빠르게 배울 수 있는 장점을 갖고 있으며, C 언어를 배우면 다른 언어를 이해하고 사용할 수 있다. 그리고 명확하게 primitive-composition-abstraction이라는 개념을 한 눈에 볼 수 있다.

1. c언어에서 primitive란 무엇인가?

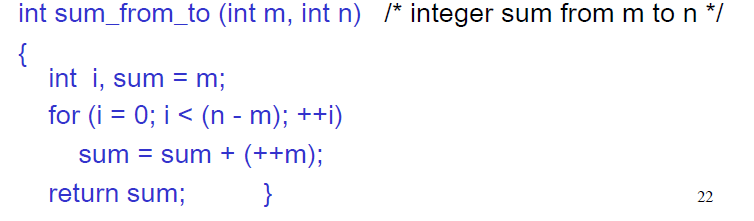
c언어에서 primitive는 statement이다. 여기에서 statement는 의미를 담는 최소의 단위이다.

더 자세히 다루면 C statement로는 변수를 선언하는 것(variable declaration statements), 값을 할당하는 것(Assignment statements), 사칙 연산과 할당을 하는 것(Arithmetic and assignment statements), 조건을 다루는 것(Conditional statement), 반복하는 것(Loop statements), 복합체(compound statements), 함수라고 불리는 것(Function call statements), 이렇게 많은 statement가 존재한다.

C 언어에서 statement를 갖고 있다는 것은 대수 방정식(algebraic equation –사칙 연산 statement, 조건부 statement)을 사용할 수 있으며, 영어로 된 구조(IF, FOR, …)를 사용할 수 있다.

그렇다면 메인(main)에 1000개 정도의 statements를 쓴다면 어떻게 될까?

설계도 어렵고 디버깅(debugging)도 어렵고, 다른 사용자가 코드(code)를 이해하는 것 마저도 어려워진다. 그래서 우리는 복잡성을 줄이기 위해 구성과 추상을 function으로 표현하여 복잡성을 감소시켜야 한다.



그림과 같이 하나의 작은 C 코드를 보자면 sum\_from\_to라는 함수가 만들면 function body를 구현(implementation)에 해당하고 body는 statement로 이루어 져 있으며, 해당 function은 한번 제대로 구현을 하면 더 이상 구현(implementation)에 대해 알 필요가 없고 사용법(I/F)만 알면 된다.

즉 함수는 한번 만들어 놓으면 여러 다른 장소에서 언제든지 사용할 수 있게 된다. 사용자 입장에서는 함수는 단일의 statement에 해당하게 되는 것이다.

1. Hiearachical Function Abstractions

함수를 만들어 놓으면 그것은 곧 단일의 statement가 되어 단일의 statement가 된 여러 함수를 새로운 함수를 구현할 때 사용한고를 반복하여, 함수를 statement로 계속 사용하여 아래서부터 위로(bottom-up) 상속되어 복잡한 함수를 추상화를 한다.

하지만 설계에 있어서는 하나의 함수 자체를 설계하는 것이 아니라 여러 개로 나누어서 설계를 하는 Top-down을 기반으로 설계가 되어야한다.

1. what is library or API(Application Programming Interface)?

라이브러리는 연관된 함수의 모임으로 C언어에서는 헤더 파일이며 사용법을 뜻한다.

소프트웨어를 설계를 할 때에는 라이브러리를 통해서 점차 큰 라이브러리를 만든다.

* C 프로그래밍에서는 가장 중요한 개념

1. statements – 단일 함수가 될 수 있는 statement는 필요한 statement를 정교하게 완성되는 것이 중요하다.
2. functions – 단일 C언어의 파일이 될 수 있으며, 우아하게 분해가 가능해야 한다.
3. Libraries – 다수의 C언어 파일이 될 수 있으며, 성능을 고려한 논리적인 API 설계를 해야 한다.

* 컴퓨터 구조(Computer Architecture)

1. ISA(Instruction Set Architecture): 컴퓨터에서 매우 중요한 I/F이며, 하드웨어와 소프트웨어 사이의 I/F를 뜻한다.

* Object-Oriented Programming

한가지 메커니즘(mechanism)이 추가되는데, object가 추가되며 다른 함수(function)보다는 커다란 데이터와 프로세싱(processing)로 결합되어 진다.

예를 들면 data structure 1, 2 function 1, 2의 statement가 하나의 object로 사용되며 사용자는 오로지 object의 I/F를 사용한다.

object는 function과 동일하게 hierarchical abstraction이 일어나는데 function보다 훨씬 큰 단위이다.

object는 실제 존재하는 것(real-life objects)를 모델(model)할 수 있다. 예를 들면 강의실에 있는 강사(teacher-object), 학생(student-object), 칠판(blackboard-object), 등등 실제 존재하는 물체를 컴퓨터 object로 만들어서 설계를 한다. 하지만 실제 존재하는 물체마다의 상호작용을 컴퓨터에서 I/F로 만드는 것이 중요하다.

그렇다면 real-life object를 어떻게 컴퓨터 object로 만들 것인가?

여기서도 hierarchy와 inheritance가 중요하다.

* Programming Paradigms

c 언어에서는 function, statement가 하나 하나 연속적으로 실행되고, OOP는 필요한 object를 많이 만들어서 object간에 message를 주고받는 분산 시스템 개념을 갖는다.

2.Discuss a real engineering example of the (hierarchical) primitive-composition-abstraction paradigm