1. computer science란 프로그래밍으로 통해 문제를 해결하는 분야이다. 기본적인 패러다임으로는 무엇을 풀어야 하는지? 문제를 인식할 주 알아야 하고, 어떻게 문제를 풀 건지? 해결방법에 대해서도 창의성을 가져야한다. computer science 문제를 해결하기 위해서는 문제를 해결하는 법과 프로그래밍(구현 기술)을 복잡하게 연결되어 있다. 우리 학교에서는 1/2 학년에서는 프로그래밍, 알고리즘, 수학적 사고/지식에 대해서 공부를 한다. 쉬운 문제에서 시작하여 난이도/복잡성 높여간다. 여기서 중요한 것은 소프트웨어 프로젝트와 전공 동아리 활동이 중요하다. 하지만 여기서 프로그래머는 비전공적이다. 3/4학년에서는 문제 해결에 대해서 초점을 맞추고 있다. 컴퓨터 구조, OS, DB, 네트워크, 등등 컴퓨터시스템에서 필요한 infra 기술에 대해서 공부를 하고 인간의 모든 활동분야, AI, 기계언어, 등 다양한 응용 시스템에 대해 공부를 한다. 예를 들면 OS개발 과정에서 무슨 문제에 부딪쳤고 이를 어떻게 해결하였나 등 문제를 해결하는 과정에서 오류와 해결방안에 대해서 공부를 한다. computer science를 하는 과정에서 성취도(전공지식 및 숙련도)에 따라서 희망하는 회사 또는 원하는 직장에서 대우가 달라진다.

초창기에 computer science는 컴퓨터에게 무엇을 해야 하는지 말하는 것에 대해 어려움을 가졌다. 컴퓨터라는 기계가 low-level languages를 사용하기 때문에 중간의 프로그래머에서 기계언어를 번역하여 명령어를 뽑는 과정에서 생산성이 많이 뒤처진 상태였다. computer science는 이 문제를 1950년 후반에 최초의 high-level language 개발하였고 컴파일러를 통해 컴퓨터와 프로그래머 사이의 자동적으로 상호작용을 쉽게 하였다. computer science는 CPU, I/O, Memory 3요소로 구성되어 있는 architecture(구조), OS, 인터넷, DB, 보안, AI 등 여러가지 응용프로그램을 중요한 역할을 하는 software(SW), 계산과 지능을 담당하는 이론, 즉 과학 이렇게 3가지의 분야 가 computer science 분야를 구성하고 있다. 공학적인 입장에서 computer science에서는 보다 빠른 machines, 보다 스마트한 software을 만들려고 노력하고 있다. 여기서도 마찬가지로 컴퓨터 전문지식 기반의 프로그래밍 문제 해결이 필요하다.

2. register-register에서 register-memory 또는 memory-memory 가면 명령어의 수는 줄어 레지스터 공간은 줄어 들지만 instruction 자체는 복잡해진다. 왜냐하면 레지스터-레지스터에 자체에서 썼던 명령어를 짧게 instruction으로 표현해야하기 때문에 하나의 instruction이 해야하는 기능이 많아졌기 때문이다. 70년대에는 메인메모리의 공간이 작았다. 그래서 디스크에서 응용프로그램을 가져올 때 부분적으로 실행시켜야 했기 때문에 instruction 수를 줄여서 레지스터 공간을 남길 필요가 있었다. 하지만 80년대 이후에는 메모리의 크기가 커졌기 때문에 굳이 복잡한 instruction을 구현할 필요가 없어졌고 컴퓨터 자체의 성능을 높여야했기 때문에 RISC die는 파이프라인 또는 캐시메모리를 구현하여 성능을 높였다. 하지만 CISC die는 복잡한 instruction이기 때문에 80년 이후에는 사라졌다.

3. 2차 대전 후에 많은 반도체 회사 중 하나인 인텔은 칩 주문 설계 및 생산하여 컴퓨터는 미국 동부가 주도하고 있었다. 한 고객이 계산이 특화된 칩을 인텔에게 요구하였지만 인텔에서는 범용

프로세서를 싱글 칩으로 만들어 4004 마이크로프로세서 칩을 만들었다. 범용 프로세서를 칩으로 만들면 계산할 수 있는 전용 칩을 만들 필요 없이 프로그램으로 만들면 되기 때문에 더욱 효율적이었다. 이것은 후에 컴퓨터 회사가 되었다. 1970년의 메인 프레임의 프로세서들은 32비트로 시작하였지만 IC 기술이 발전이 되어있지 않았기 때문에 32-bit 컴퓨터를 구성하기 위해서 많은 칩들을 필요했기 때문에 PCB(printed circuit boards)가 대량으로 필요했다. 그런 시대에서는 인텔에서는 4비트 프로세서를 만들었다. 후에 8비트 프로세서가 상용하게 되면서 마이크로 컴퓨터가 만들어지기 시작하여 소프트웨어와 마이크로 컴퓨터가 대중화되었다. 싱글 칩 프로세서들은 강력한 프로세서로 트랜지스터의 규모와 향상된 설계가 더해졌고 컴퓨터 회사들은 프로세서 공급업체로부터 프로세서를 구입하여 1980년도에 컴퓨터 회사에 거대한 일자리에 대한 변화를 가져다 와줬다. 이후 시스템과 소프트웨어, 서비스에 집중하게 되었고, 작은 마이크로프로세서는 여전히 시스템이 탑재되어 적은 비용으로 남아있다.

4. RISC 설계에서 다이 사이즈를 계속 크게 만드는 것은 가능해졌지만 다이 사이즈를 두배를 높인다고 성능은 두배가 아니라 1.4배 즉 40% 증가한다. 그리고 가장 문제가 된 POWER은 두배가 높아졌다. 초창기의 40%의 성능 증가는 기업 입장에서는 좋은 성적이였지만 계속 다이 사이즈를 증가시켜 성능을 높여봤자 겨우 1.4배 증가되고 POWER는 계속 두배가 늘어나서 COOL AIR을 쓰는데 한계에 도달해졌다. 열을 제거할수 없고 부피를 줄일수 없었기 때문에 다이 사지를 증가시키는 것은 무리가 있다. 그래서 프로세서를 단순한 프로세서를 써서 해당 다이 하나에 단순한 프로세서를 여러 개 넣음으로써, 즉 다이안에 단순한 프로세서들을 넣어서 멀티 코어, 멀티 프로세서를 만들었다. response time은 늘어나지 않고, 프로세서가 여러 개 있으면 프로세서에 대한 별개의 일을 주기 때문에 throughput은 늘어났다.

5.실제 응용프로그램을 대표할 수 있는 프로그램의 집합을 잘 선정하여 그것을 이용해서 response time을 측정한다. single number을 구하는 방법은 일단 specint, specfp 두게를 구할 수 있는데 예를 들면 기존 응용프로그램을 돌리는 ref time과 해당 응용프로그램의 시간을 비교하여 SPECratio를 계산을한다. 응답 시간에 보통 초점을 둔다. 왜냐하면 Throughput 자체의 성능을 높이려면 대규모 설계가 필요하다. 하지만 Response time을 줄이면 자동적으로 Throughput도 동시에 성능이 좋아지기 때문에 Throughput보다는 성능 개선이 편한 Response time에 초점을 둔다.

6. 디지털 논리 설계에서 AND, OR, NOT 게이트로 통해 1-bit full adder을 구현하여 하나의 1-bit full adder 하나를 인터페이스로 사용하기 위하여 추상화를 시킨다. 1-bit adder가 이제 하나의 primitive가 되어서 계속 상속되어 32-bit binary adder을 만든다.

7. 컴퓨터 성능을 위해 무엇을 측정해야하는가?,컴퓨터 성능을 어떻게 측정하는가?, 컴퓨터 성능을 향상시키기 위해 어떻게 해야하는가?,

IC(instruction count)

최상위 개념 IC는 ISA(instruction set architecture)(I/F) 설계에서 결정되며, ISA가 확정이 된다면 컴파일(Compiler) 설계가 가능하다.

CPU 시뮬레이터로 Benchmark을 돌리면 IC를 얻고, 이에 따라 implementation(CPI & cct)가 영향을 받는다.

CPI(cycle per instruction)

차상위 개념, high-level implementation – 한 instruction이 실행되는데 필요한 clock cycles의 수로 instruction 마다 다르다.

최종적인 CPI는 빈도에 따른 weighted average로 계산한다. – 실제 실행되는 Dynamic count를 고려한다.

설계된 ISA 구현시 CPI가 결정된다.

cct(clock cycle time)

하위 개념으로 clock 한 주기의 길이를 의미한다. ISA 및 high level implementation이 결정되면, 한 클럭 내에서 할 일 들일 결정되는데 cct를 줄일수록 성능이 좋아진다.

회로 설계자는 cct를 줄이려 노력을 하는데, clock 내에 할 일들 중에서 가장 오래 걸리는 작업을 더욱 빠르게 만들려고 노력을 한다. 1clock cycle은 이미 결정되었지만 그 안에 signal을 줄이려고 노력을 한다.

8. CSE에서는 공학 분야에서는 CPU, I/O, Memory 3요소의 구조와 소프트웨어, 그리고 이론(과학)에 대해서 배운다. 컴퓨터 전문지식 기반으로 프로그래밍 문제해결 방법으로 보다 빠른 기계와 보다 똑똑한 소프트웨어를 훈련하는데 보다 빠른 기계를 만들기 위해서는 컴퓨터 구조에 대한 학문을 공부를 해야 하고 보다 스마트한 소프트웨어를 만들기 위해서는 계산능력과 창의성을 길들어진 알고리즘을 공부를 해야 한다. 소프트웨어에서는 대표적으로 OS, 인터넷, 데이터 베이스, 보안, AI, 스마트응용 등 여러가지의 응용 프로그램이 있으며 그 만큼 소프트웨어에서 여러 분야를 심도 있게 공부하기 위해서는 창의성과 계산능력을 향상시키는 것을 중요시해야 한다. CSE는 infrastructure 문제들을 해결하곤 했는데 infra 소프트웨어는 OS, 인터넷, 웹, DB 등등 있다. 21세기에서는 소프트웨어를 이용한 자동화가 어떤 영역에도 적용 가능하며 모든 산업과 우리의 일상을 바꾸어 놓고 있다. 현재 4차 산업 혁명에서 주요 인재는 문제 해결 능력을 갖춘 소프트웨어 인재가 될 것이다. CS 교육 목표는 소프트웨어 개발자 양성뿐만 아니라 고급 개발자를 만들기 위한

큰 교육이 무엇보다 중요시되고 있다. 전공 공부를 통한 전문인의 자신감과 도전정신을 갖고 유익한 사회 구성원이 되어야 한다. CS전공자의 진로는 IT 정통한 사업가, 투자자, 교육자, 관리자, 경영자, 산업 디자이너, 법률가, 관료, 등등 폭 넓은 진로가 열려져 있다. 기억해야하는 것은 우리는 컴퓨터 과학자이고 문제 해결사라는 것을 항상 각인하고 살아야 한다.