170110 – 컴퓨터에 대하여

오늘 배울 것
컴퓨터 / 컴퓨터 프로그램 / 컴퓨터 프로그래밍
컴퓨터의 역사 / 데이터의 표현방식
컴퓨터의 구성 / 구조/ 동작
컴퓨터의 연산

프로그램?
- 컴퓨터에 의해 실행될 수 있는 일련의 기능 모음

컴퓨터?
- 전자계산기 징그러운 0 과 1, 컴퓨터 신호가 나왔다 꺼졌다.

컴퓨터의 역사

- 암호해독 , 복잡하고 긴계산을 빨리하기 위해 생긴게 컴퓨터의 시초
- eniac(에니악) - 최초의 다용도 디지털 컴퓨터
- 진공관 - 제 1 세대 컴퓨터(계속 쓰다보면 터진다. 즉 내구성 안 좋음, 또 벌레도 생김 그래서 버그 bug 라는 말의 시초가 이것이라는 말이 있음)
- 트랜지스터 - 제 2 세대 컴퓨터(이제야 집 반채정도의 크기로 줄어들었음)
- 집적회로 - 제 3 세대 컴퓨터 ~ 현재

컴퓨터의 구성

하드웨어와 소프트웨어

하드웨어

- 컴퓨터와 그 주변을 구성하는 물리적 장치 등
- 입력장치, 출력장치, 기억장치, 제어장치, 연산장치 등

이중 중요한 것은 기억장치

- 주기억장치 - 램 - 속도가 빠름, 전원이 꺼지면 데이터가 지워짐
- 보조기억장치 - 하드디스크 - 매우 느림, 전원이 꺼져도 데이터가 지워지지 않음

제어 / 연산장치 - 중앙처리장치(CPU)

소프트웨어 - 프로그램(명령어의 집합을 수행하는 것)

시스템 소프트웨어 - 운영체제, 로더, 장치 드라이버, 컴파일러, 어셈블러, 링커, 유틸리티

시스템 소프트웨어는 애플리케이션 소프트웨어가 시스템 소프트웨어를 거쳐서
하드웨어로 가게됨, 즉 중간 매개자임

응용 소프트웨어(애플리케이션) - 워드프로세서, 웹브라우저, 스프레드시트, 게임
ot 상에서 실행되는 모든 프로그램

컴퓨터의 구조

하버드 구조
cpu(컴퓨터의 모든 일처리를 하는 녀석)
메모리는 컴퓨터가 처리해야할 명령어와 데이터를 모두 가지고 있다.
명령어 메모리(마우스 오른쪽을 클릭했을 때 유닛을 여기까지 움직여야하는데, 좌표, 대상, 명령어가 뭔지 알아야 한다 그러면 어디로, 어떤 유닛을 어디까지 움직여야 되나라고 할 때 움직여라!하는 것은 명령어 메모리, 어디서 부터 어디까지 어떤 유닛은 데이터 메모리)

ex 유닛을 움직여라
명령어 메모리(움직여라) -> 제어장치(어떤걸 움직여야 하냐) -> 데이터메모리(이거다!) -> 산술논리장치(이것을 움직이려면 어디어디까지해야한다 계산함) -> 데이터메모리(이거다!)

하버드 구조의 장점

프로그램 메모리와 데이터 메모리가 물리적으로 분리되어 있다.
속도가 빠름
구성에 비용이 많이 들며 복잡

폰 노이만 구조(우리는 보통 폰 노이만 구조를 사용하는데, 그 이유는 캐시 메모리에 명령어를 미리 복사를 해놓는다. 그래서 캐시메모리가 이러한 문제를 해결 할 수 있기 때문에 폰 노이만 구조를 쓰는 것 엘원캐쉬와 엘투캐쉬, 하나는 메모리 하나는 데이터, 즉, 내부적으로는 하버드구조를 따르면서 외부적으로는 폰 노이만을 써서 병목과 메모리 속박을 좀 더 빠르게 할 수 있다.)
메모리 memory 한개

모든 길이 다 하나의 길로만 간다.(통로, 버스)
버스를 통해서 움직이는 데이터가 한 방향이다. 그래서 막히는 현상이 심해서 속도가 느림

폰 노이만 구조의 장점
단순, 싸다.
하지만 메모리 속박 문제 및 버스 병목 문제를 가진다.

* 메모리 속박 문제 - cpu 는 빠른데 메모리는 데이터를 불러 오는 과정이 cpu 가 일처리 하는 과정보다 오래걸린다. 그래서 cpu 가 대기하는 시간이 훨씬 길어진다.

질문
폰노이만 구조는 왜 쓰는 것인가 그런데??
답 - 가격이 절대적, 하버드는 너무 복잡하고 단가가 비싸다. 대량 생산하기에는 폰노이만 구조를 쓰는 것이 맞고, 그 다음에 cpu 안에서 캐쉬메모리를 이용해서 단점을 보완하는 것이다.(신기..)

데이터의 표현방식 - 이진법

프로그래밍할때 주로 보는 수는 2 진법, 8 진법, 16 진법

16 진법을 많이 보는 이유는 ? 좀 더 짧게, 편하게 보고 싶기 때문

16 진수를 표현 할 때는 0x 라고 표현한다.

ex) 0xA78F (1~9 다음에 A 부터 감 대소문자 구분 x, 고로 f 는 15)
근데 이걸 10 진수로 환산하는건 복잡한데,
16 진수로 환산하는 것은 굉장히 편하다. 4 칸으로 나누면 1111 이 15 를 표현할 수 있기 때문이다.

rgb(255,255,255)를 ffffffff 로 표현할 수 있는것(맨 뒤는 알파 값 - 투명도)

256 의 세제곱 값에 알파값 256 단계해서 256 의 네제곱가지의 수를 만들 수 있는 것

그래서 16 비트 칼라는 4 의 네제곱인데, 지금은 64 비트를 보통 쓰니까 진짜 많은 것

예를 들어 사진을 뽑았을 때 1024x768 해상도면, 이 구분을 픽셀 단위로 사진이 구성되어 있어서 그런 것이다. 만약 16 비트면 1024x768x16 비트이다.

방금 여기에서는 정수 중 자연수로 밖에 표현 할 수 없었다.
맨 앞에는 양수인지 음수인지 표현을 하자라는 약속을 하기도 함,
그러면 0 을 표현할 수 있는 방법이 없다.

정수 외에 다른 수를 표현 할 때도
2 진법으로 표현하는 것중에서 규칙을 우리가 설정하면 되는 것이다.

* 중요한 것은 0 과 1 을 많이 쓸 수록 더욱 더 많은 숫자를 표현 할 수 있다는 것,

문자

ascii(8 비트 표현 첫 표준 문자 set) / unicode / utf-8 / euc-kr(완성형) / cp 949

다른 코드표에는 다른 방법을 쓰기 때문에 문자를 표현할 때 같은 것이 아니면 통하지 않을 수가 있다.

인코딩(encoding) : 데이터를 코드화하는 것
디코딩(decoding) : 코드를 다시 데이터로 바꾸는 것

방식들마다 조합형이 있고 완성형이 있다. 이런식으로 코드 체계가 모두 다 다르기 때문에, 다들 새로 문자 셋을 바꾼 것이다. 다들 세종대왕이 되려고 했나 보다.

지금 가장 표정으로 쓰는 것은 UTF-8 이라고 한다.(현존하는 거의 모든 글자를 표현하는 방식)

* 인코딩을 볼 때는 0 과 1 이 나열되어 있을 때, 이것을 어떻게 해석해내느냐의 문제

예전에 왜 정수형을 32 비트로 표현을 했을까

32 비트와 64 비트의 차이가 무엇일까?

32 비트와 64 비트의 차이는, 길이 32 개냐 64 개의 차이다.
32 비트 운영체제에서는 메모리를 4gb 까지밖에 쓰지 못한다.

cpu 랑 메모리 사이에서 데이터와 명령어가 왔다갔다 하는 것,
메모리에 0 과 1 들이 무수히 들어가 있을 것이다. 이 안에다가 주소를 만들어 놓는다.

ex) 철수가 영희네 집에 가야한다. 하나하나마다 숫자를 매겨 놓은 것, 데이터를 만들어 놓은것, 근데 표현할 수 있는 숫자는 몇십기가까지 표현을 할 수 있는데, 한번의 표현이 버스가 한번에 지나갈 수 있는 게 32 비트, 즉 4gb 라는 것이다.

* 간단히 말하면 통로가 32 비트(4gb)까지 밖에 없어서 램 용량이 아무리 커봐야 통로는 4gb 까지 밖에 못 쓰는 것

8bit == 1byte
32 bit == 4byte
64 bit == 8byte

64bit 일 때는 192gb 까지 쓸 수 있다는 것

근데 버스통로 때문에 생기는 문제

예 : ip 주소는 지금 32 비트로 표현하고 있는데, 이게 수가 적다.
이제 ip 주소가 고갈이 되었다. 2012 년도에 이미,

지금 이제 ipv6 가 우리가 요즘 쓰고 있는 ip 주소이다.
지금은 128 비트까지 올렸다. 이것은 지구상의 개미새끼 하나씩만 줘도 된다고 한다.

컴퓨터의 연산

cpu 는 더하기밖에 할 줄 모르는 애다. 계산기가 아니라 가산기이다.

논리연산

and or not xor nor nand

and 와 or 와 not!

* NOT - 하나하나씩 반전을 시켜주는 것, 그것이 NOT 이다. 비트연산에서 비트를 다 거꾸로해주고 +1 을 해주는 것

* AND = 논리곱, 얘네들을 서로 곱하는 것이 AND, 즉 둘다 1 이어야 1 이고, 아니면 0 이다.

* OR = 둘 중의 하나만 1 이면 1 이 나오고, 둘다 0 이면 0 이 나오는 것

* XOR = 둘다 더하는 것

반가산기는 SUM : A XOR B, CARRY = A AND B 이다.

* 빼기는 반가산기, 더하기는 가산기, 곱하기는 가산 계속, 나누기는 빼기 계속

그런데 좀 전에 0110 - 0001 을 했을 때, 자릿수가 넘어가서 10101 이 된다. 근데 앞의 1 이 탈락하는데 이것을 오버플로우라고 한다.

만약 4 비트로 표현하고 16 까지 표현을 할 것이다.

0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1111

0~7 까지 하고 다시 -8 -7 -6 ~ -1 까지 온다

0111 에서 1 을 더해줘

1000 이라는 값을 가질 것, 결과값이 1000 이라고 나왔는데 값은 8 이 아니라 -8 이 나와버린다. 이것이 오버플로우이다. (자릿수가 초과해버려서 원치 않는 값을 얻게 되는 것)

비트논리연산 vs 부울논리 연산

& vs &&, | vs ||

비트논리 연산 = 각 비트 별 0 과 1 연산
부울논리 연산 = 참, 거짓 판별

& | ~(비트 낫) !(부울 낫)

왜 이런 연산을 이해해야 하나요?
- 부울논리는 우리가 프로그래밍할 때 많이 쓴다.

그럼 비트논리연산은 ?
- 원하는 데이터를 중간중간 뽑아낼 수 있기 때문에, 비트마스킹이라고 함
* 중복되지 않는 옵션값들을 표현 할 때 사용하게 된다.

과제

RAM 은 Random Access Memory 의 약자이다.
임의접근 방식을 가지고 있는 읽기/쓰기 메모리로 휘발성 메모리이다. 휘발성이기 때문에 ROM 보다는 속도가 많이 빨라도 빠르다고 한다.

RAM 은 DRAM 과 SRAM 으로 나뉜다.

DRAM(Dynamic RAM)
저장된 내용을 유지하기 우해 일정 간격으로 반드시 Refresh(재충전)이 필요하다.
집적도가 높아 전력소모가 적고, 구조가 간단해서 용량이 크기 때문에 일반적인 pc 메모리용 메모리가 이것에 속한다.

SRAM(Static RAM)
Refresh 가 필요없고, DRAM 보다 빠르다. 왜냐하면 재충전 없이 계속 전력을 공급시킬 수 있기 때문이다.그래서 Static(정적)인 것이다.
집적도가 낮아 전력소모량이 크고 구조가 복잡해서 용량은 작지만 속도가 굉장히 빠른데, 그래서 CPU 의 캐쉬 메모리로 주로 사용된다.

RAM 이라고 부르는 이유

RAM 은 한글로 번역하면 '임의 접근 메모리'이다. 메모리의 어느 위치로든 바로 접근할 수 있기 때문에 이러한 이름이 붙게 되었는데, 이에 반해 순차적으로만 접근이 가능한 장치는 SAM(Sequential Access Memory)이다. 주기억 장치는 아니지만 하드디스크도 일종의 RAM 이라고 한다.

사실 RAM 을 두가지로 나누면 읽기 전용 메모리 ROM(Read Only Memory)과 읽기 쓰기 메모리(Read Write Memory)인 RWM 으로 나눌 수 있는데, 우리가 흔히 부르는 RAM 은 엄밀하게는 RWM 이라고 볼 수 있다고 한다.

이런 일이 벌어지게 된 이유는 임의 접근 읽기 쓰기 메모리를 개발한 당시엔 RWM 을 이미 SAM 의 세분화된 메모리 방식을 일컫는 용어로 굳어졌다고 한다. 그래서 혼란을 막고자 RAM 이라고 부른 것이다.

한글인코딩 방식은 크게 UTF-8 과 EUC-KR 로 나눠집니다.

UTF-8

조합형 방식 - 말그대로 조합형, 자음과 모음을 초성,중성,종성으로 구분하여 문자를 작성, 초성 중성 종성을 따로 인식하고 그것을을 하나의 바이트로 인식하기 때문에 총 3 바이트의 문자로 인식

EUC-KR

완성형 방식 - 문자를 하나의 완성되어져 있는 글자로 인식하는 방법, 문자표를 토대로 작성합니다. 그러나 확장성이 떨어지는 단점이 있는데 만약 "꿿"이라는 글자가 문자표에 없다면 "ㅁ"으로 표기가 된다. 따라서 오래된 한글 표현 방식,

이렇게 보면 조합형이 더 좋아보이지만, 윈도우가 운영체제 점유율이 굉장히 높은데 윈도우의 인코딩 방식은 기본적으로 완성형이었다고 합니다. 그래서 완성형이 많았습니다.

그러나 EUC-KR 에서 진화한 CP949 방식은 더 많은 한글 테이블을 제공하기 때문에 조합형과 거의 차이가 나지 않는데도 불구하고, 많은 불편함이 있는데요. 그 이유는 웹서비스 때문입니다.

웹서버나 데이터베이스 또는 php 의 경우에 UTF-8 과 EUC-KR 중에서 인코딩을 서로 똑같이 맞춰줘야 정상적으로 문자표현이 가능하기 때문입니다.

UTF-8 같은 조합형 방식은 다른 국가에서 한글 언어팩이 설치되지 않았다고 하더라도 한글 표현이 가능하고, 다른 나라의 언어팩이 없어도 다른 나라의 언어를 볼 수 있습니다. 즉 다양한 언어로 작성되는 환경이나 웹과 같은 다양한 국가의 사람들이 보기에는 조합형이 더 편한 것입니다.

하지만 보통 사용하는 EUC-KR 방식은 완성형 방식을 따르고 있었고, 한글을 사용하는 곳에서만 문자가 제대로 보이는 단점이 있었습니다. 그래서 한글과 영어만 사용하는 페이지에 적합합니다.

다음 사진은 인코딩 방식에 따라 표기할 수 있는 관계를 표현한 것입니다.