CHUNGNAM NATIONAL UNIVERSITY

# 시스템 프로그래밍

## 강의 1. 과목 소개

Sep. 1, 2010

http://eslab.cnu.ac.kr

# 이 과목의 주제

- **컴퓨터 프로그래밍은 추상화(Abstraction)이 대부분**
  - 그러나, 본질을 놓치면 안된다
- **컴퓨터 전공의 추상화**
  - 자료구조
  - 가상메모리 ?
- **추상화는 한계가 있다**
  - 버그가 있을 때
  - 시스템 내부의 동작에 대한 이해가 필요할 때
- **과목의 목표**
  - 효율적인 프로그래머를 만들자
    - 기괴한 버그를 효과적으로 퇴치
    - 성능을 고려한 프로그래밍
  - 향후 컴퓨터공학 전공 과목들을 위한 준비

# 위대한 현실 #1

- ***정수가 정수가 아니고, 실수가 실수가 아니다?***
- **Examples**
  - **$x^2 \geq 0$?**
    - Float's: Yes!
    - Int's:
      - 40000 * 40000 --> 1600000000
      - 50000 * 50000 --> ??
  - **$(x + y) + z = x + (y + z)$?**
    - Unsigned & Signed Int's: Yes!
    - Float's:
      - (1e20 + -1e20) + 3.14 --> 3.14
      - 1e20 + (-1e20 + 3.14) --> ??

# 컴퓨터 내에서의 연산

- **의미없는 랜덤 값을 만들지는 않는다**
  - 수식연산은 중요한 수학법칙을 갖는다
- **그러나 일반적인 법칙을 가정할 수는 없다**
  - 데이터 표현의 유한성 때문에
  - 정수 연산은 **"ring"** 특성을 갖는다
    - Commutativity, associativity, distributivity
  - 실수 연산은 **"ordering"** 특성을 갖는다
    - Monotonicity, values of signs
- **관찰**
  - 어떤 추상화가 어떤 상황에 적용된 것인지 이해하는 것이 필요
  - 컴파일러 개발자나 고급 프로그래머들에게는 중요한 주제

# 위대한 현실 #2

- ***어셈블리어를 알아야 한다!!!***
- **대개의 경우, 어셈블리 프로그램을 작성할 가능성은 ..... zero !**
  - 컴파일러가 훨씬 더 우수하고 참을성이 있다
- **어셈블리어를 이해하는 것은 하드웨어 수준의 실행모델을 이해하는데 매우 중요**
  - 버그가 있을 때 프로그램의 동작
    - 고차원 언어 모델로는 이해할 수 없다
  - 프로그램의 성능을 튜닝할 때
    - 프로그램의 효율성을 이해하기 위해
  - 시스템 소프트웨어 구현시
    - 컴파일러는 기계어 코드가 목적코드이다
    - 운영체제는 프로세스의 상태를 관리해야 한다
  - **malware**를 만들거나 **malware**와 싸우기 위해
    - x86 어셈블리를 배운다

# 어셈블리 코드 예제

- 타임 스탬프 카운터
  - x86 계열의 프로세서에서 제공하는 64비트 레지스터
  - 매 클럭 사이클 마다 1씩 증가
  - rdtsc 명령으로 접근
- C 프로그램 내에 타임스탬프 접근 어셈블리 작성

```
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

/* Set *hi and *lo to the high and low order bits
   of the cycle counter.
*/
void access_counter(unsigned *hi, unsigned *lo)
{
    asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
        : "=r" (*hi), "=r" (*lo)
        :
        : "%edx", "%eax");
}
```

# 위대한 현실 #3

- ***메모리가 중요하다: RAM 은 추상화일 뿐***

- **메모리크기는 무한하지 않다**
  - 메모리는 할당해주고 관리해야 한다
  - 많은 응용프로그램들은 메모리에 영향받는다
- **메모리 참조 버그는 치명적이다**
  - 버그의 결과가 시공간적으로 무연관성을 보인다
- **메모리 성능은 일정하지 않다**
  - 캐시와 가상메모리 효과는 성능에 비선형적 영향을 미친다
  - 프로그램들을 메모리 시스템의 특성에 최적화 시키면 성능이 대폭 개선된다

# 메모리 참조 버그

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

```
fun(0)  ->    3.14
fun(1)  ->    3.14
fun(2)  ->    3.1399998664856
fun(3)  ->    2.0000061035156
fun(4)  ->    3.14, then segmentation fault
```

# 메모리 시스템 성능

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

59,393,288 clock cycles

1,277,877,876 clock cycles

(Measured on 2GHz Intel Pentium 4)

21.5 **배 더 느리다!**

- **계층적 메모리 구조**
- **성능은 메모리 접근 방식에 영향을 받는다**
  - 다중 배열에서 어떻게 인덱싱 하는가에 따라 달라진다

# 위대한 현실 #4

- ***컴퓨터는 프로그램 실행 이외의 여러가지 일을 한다***

- **데이터의 입출력이 필요하다**
  - I/O 시스템은 프로그램의 신뢰성과 성능에 매우 중요하다
- **프로그램이 순차적으로 실행되는 것은 아니다**
  - 예외적인 순서가 있어야 컴퓨터다

# 과목의 목표

- 프로그램이 어떻게 하드웨어에서 동작하는지 이해한다.
- 리눅스의 프로그래밍 환경에 익숙해 지도록 한다
- 시스템 프로그래밍 기법을 학습한다
  - Process control
  - Signal
  - System level I/O
  - Memory management
  - Concurrent programming

  - And.... Bonus !

**컴퓨터 시스템에 관한 깊은 이해를 제공한다**

# 학습 내용

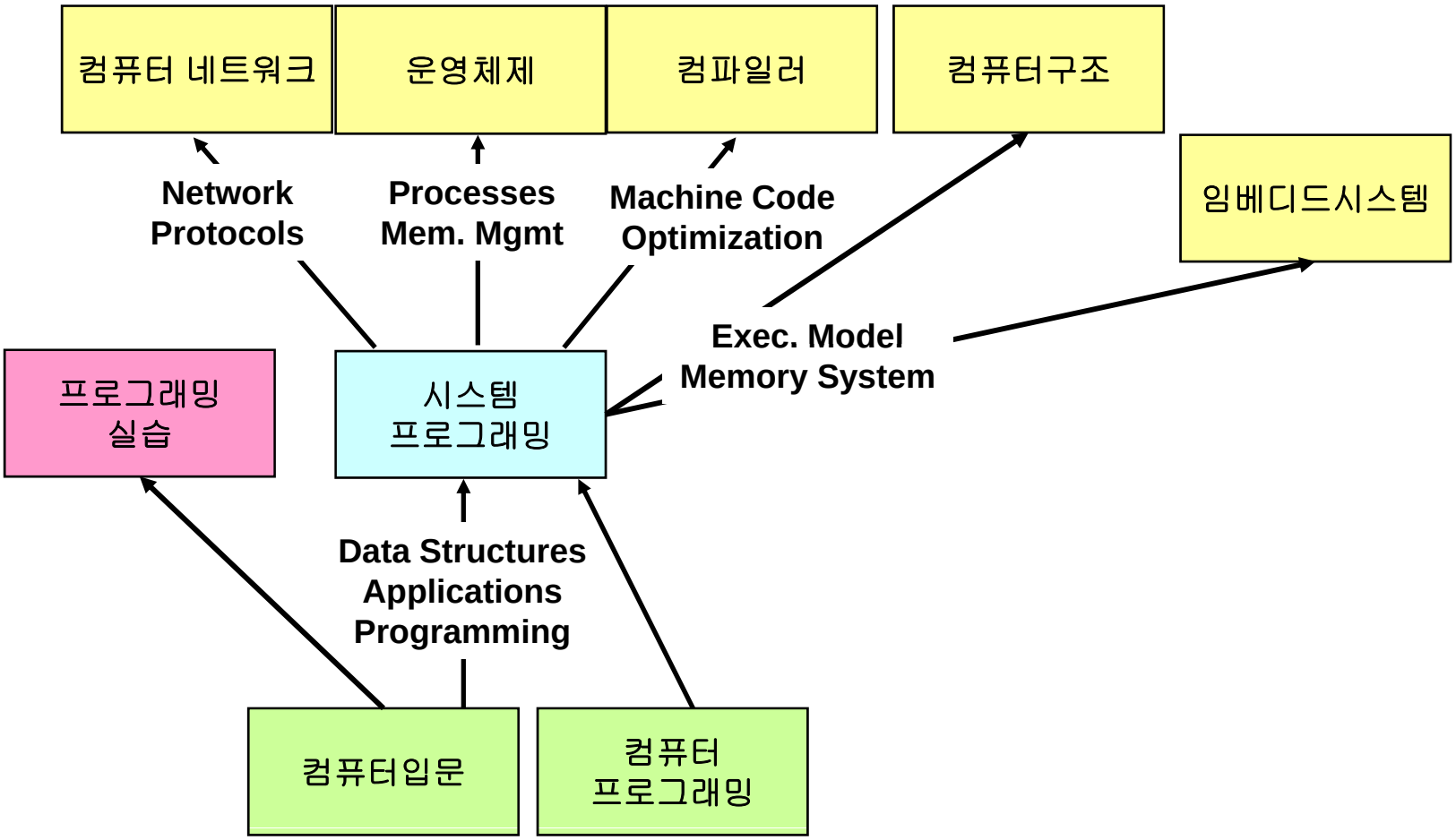
- 컴퓨터에서의 데이터 표현
  - 정수
  - 실수
- 어셈블리어와 프로세서 구조
- 예외적 인 제어흐름
  - 프로세스
  - 시그널
- 입출력 시스템
- 메모리 관리
  - 가상메모리
  - 동적 메모리 할당
- 동시성 프로그래밍 (Concurrent programming)

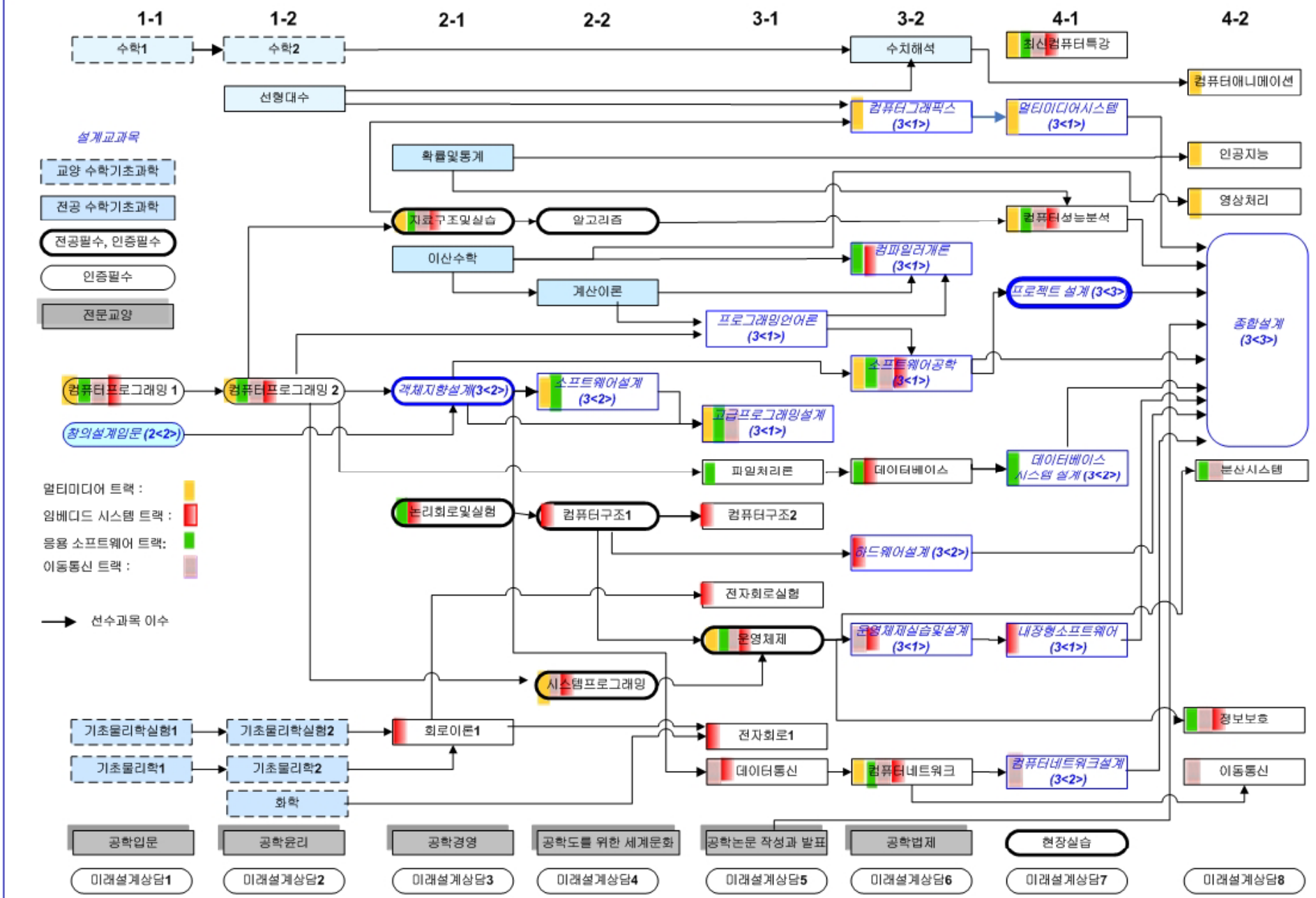
# 이 수업이 끝날 때 여러분은,

- IA-32 어셈블리어를 사용할 수 있게 된다.
- 다른 프로세서의 어셈블리어를 쉽게 습득할 수 있게 된다.
- 저 차원의 디버깅 기술을 사용할 수 있게 된다.
- 컴퓨터 내부에서의 숫자의 표현방법을 이해하게 된다
- 운영체제의 예외처리 과정을 이해한다
- 프로세스를 제어하는 프로그램을 작성할 수 있다
- 운영체제의 메모리 관리 원리를 이해하게 된다
- 동시성 프로그램을 작성할 수 있다.
- 성능을 고려한 시스템 프로그램을 작성할 수 있다

**막강한 준-프로 시스템 프로그래머가 된다!**

# 컴퓨터 전공과목과의 결정적 관계






2009년도 교과목 이수체계도

**2010, 가을학기 – 시스템 프로그래밍**

# 과목 개요 – 1 / 2

- 조교 : 조으뜸, 정현아, 백종철
- 교재 :
  - Computer systems : A programmer's perspective, 2판, by Randal E. Bryant and David O'Hallaron, Prentice Hall
- 참고문헌
  - M. Mitchell, J. Oldham, and A. Samuel, Advanced Linux Programming, New Riders Publishing, 2001. (Available online)
  - "UNIX for Programmers and Users", 3rd Ed. Graham Glass & King Abbles
- 교과목 홈페이지
  - http://eslab.cnu.ac.kr, lecture menu, 2010 Undergraduate
- 선수지식
  - 능수 능란한 C 언어 프로그래밍

# 과목 개요 – 2 / 2

- **성적평가**
  - 중간고사 : 20%
  - 기말고사 : 30%
  - 실습 : 30%
  - 출석 : 10%
  - 숙제 : 10%
  - 주의 ! 2학년이 아닌 고학년의 수강 시 고학년의 수강수준을 감안하여 높은 기준으로 별도로 평점 부여
- **출석**
  - 2회까지는 눈 감아줌. 3회 결석부터 2점씩 감점
- **숙제**
  - 예습 교재읽기, 문제풀이, 프로그래밍 등 다양한 유형
  - 마감일을 넘기는 경우 50% 감점
- **숙제/실습 복사에 대해**
  - 부도덕한 행위에 대해서 F 부과
  - 학업에 이름을 걸고 하자

# 이 과목의 역사

- 2005 가을학기 Unix programming 2-1-2 : 4.1/5.0
- 2006 가을학기 Unix system programming, 3-2-2 : 3.9/5.0
  - 2005 Unix programming + System programming
  - 교재 변경=> Programmer's perspective ...
  - REB 책 후반부로 구성 (Ch.8 – Ch.13)
  - 영어 강의 1개반, 일반강의 1개 반
- 2007 가을학기 – Unix 프로그래밍 : 4.1/5.0
- 2009 가을학기 – Unix 프로그래밍 4.2/5.0

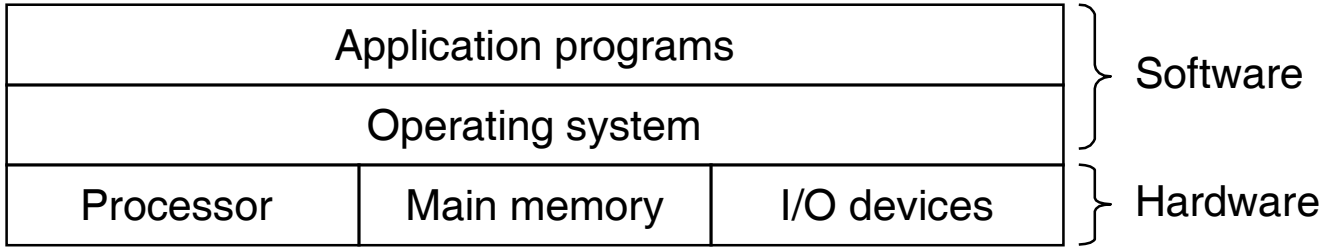
# 2010 시스템 프로그래밍 강의 구성

- **2009 어셈블리어 프로그래밍 + 유닉스 프로그래밍**
  - 두 과목의 최대 인기 실습으로 최강 실습 구성
- **중요한 개념들 추가**
  - **Floating Point**의 표시
  - 가상 메모리
- **덜 중요한 것들 제외**
  - 비트 연산자
  - 네트워크 프로그래밍
  - 링커
- **강의 주안점**
  - 도전적인 실습과제의 구성
  - 2학년 수준에 적합한 강의내용
  - 시스템 프로그래밍 본질의 이해
  - 영어 원서 강의 교재 적응
  - 어리버리한 2학년들의 전공으로의 연착륙

# 제 1장. 컴퓨터 시스템 개관

- 컴퓨터 시스템의 본질을 찾아서...

# 컴퓨터 시스템의 계층도


Application programs
Operating system
Processor
Main memory
I/O devices
Software
Hardware

# 컴퓨터 하드웨어의 구조

- **하드웨어 구성요소**
  - 버스 **buses**
  - 입출력 장치, **I/O devices**
  - 주기억장치, **main memory**
  - 캐쉬, **cache memory**
  - 중앙처리장치, **CPU (Central Processing Unit)**
- **버스**
  - 구성요소간의 정보교환 통로
  - 종류 : 입출력버스, 시스템 버스

# 프로세서

- **Central Processing Unit : CPU**
  - 산술논리연산장치 : **arithmetic and logic unit, ALU**
  - 제어장치 : **Control unit**
  - 레지스터
    - 프로그램 카운터 PC
    - 상태 레지스터
    - 메모리 주소 레지스터
    - 메모리 데이터 레지스터
    - 명령 레지스터, instruction pointer, IP
    - 범용 레지스터, general-purpose register

# CPU

## 주요기능

- 기억장치로부터 명령/데이터 호출 및 저장
- 명령의 해석 : 제어장치
- 명령의 실행 : **ALU**
- 입출력 연산 : **I/O**

## 명령 주기 Instruction cycle

- 인출 **fetch** : **PC**에 의거하여 명령어를 가져옴
- 해독 **decode** : 명령어를 해석하여 제어 신호 생성
- 실행 **execution** : 가져온 명령어를 실행
- 결과저장 **store** : 실행 결과를 저장

# 운영체제 Operating System

- **운영체제란?**
  - 응용프로그램이 하드웨어를 효율적으로 사용할 수 있도록 도와주는 프로그램
- **목적**
  - 하드웨어 추상화 **Hardware Abstraction**
  - 시스템 인터페이스 추상화
- **운영체제의 관리대상**
  - 프로세스
  - 메모리
  - 입출력장치
  - 소프트웨어 자원
    - 파일시스템, 라이브러리, 유틸리티

# Hello world ?!!

- 위대한 프로그램 hello.c

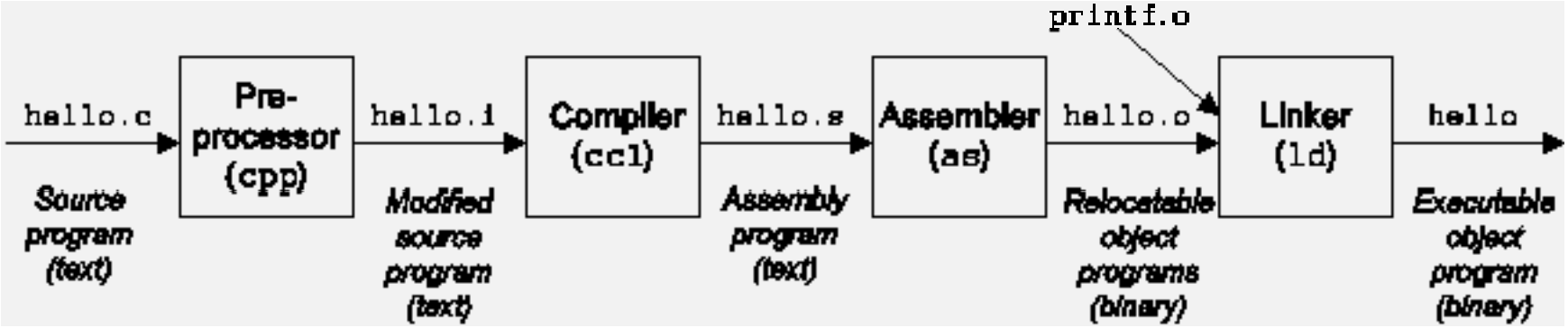
```
#include <stdio.h>

int main()
{
        printf("hello, world₩n");
}
```

- 본질 ! 컴퓨터의 모든 정보는 비트, 바이트로 저장된다

# 프로그램은 변신한다

- hello.c 컴파일 과정
  - unix> gcc –o hello hello.c




- 본질 ! 컴파일러의 뒷면에는 이렇게 복잡한 일이 일어나고 있다

**어셈블리 프로그램 ? 실행 목적 프로그램 ? 기계어 프로그램 ?**

# 컴파일 과정을 이해하는 것은 매우 중요하다

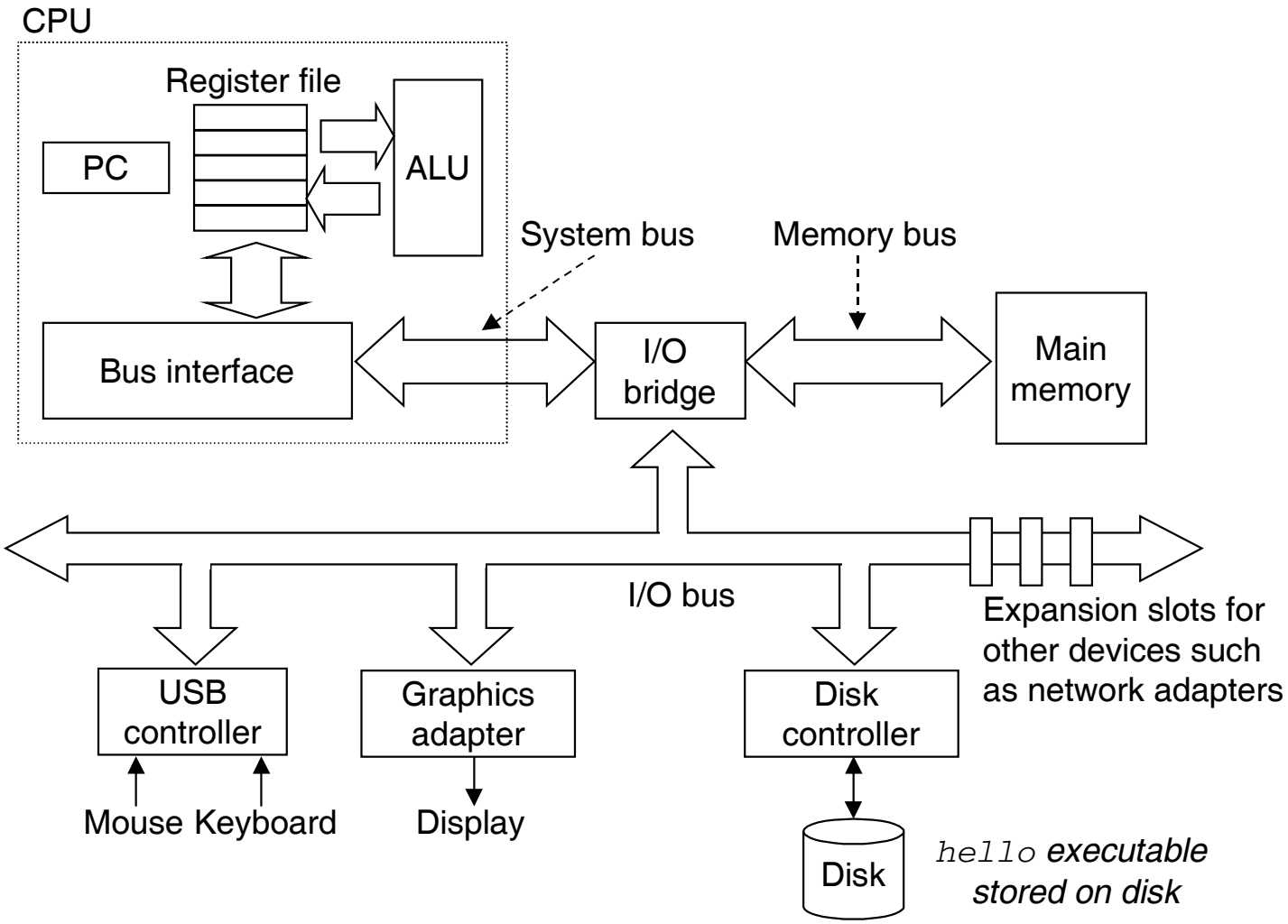
- 프로그램 성능을 개선할 수 있다
- 링크시에 발생하는 에러를 이해할 수 있다
- 보안과 관련된 프로그램의 약점을 해소할 수 있다

# 변환된 프로그램의 실행

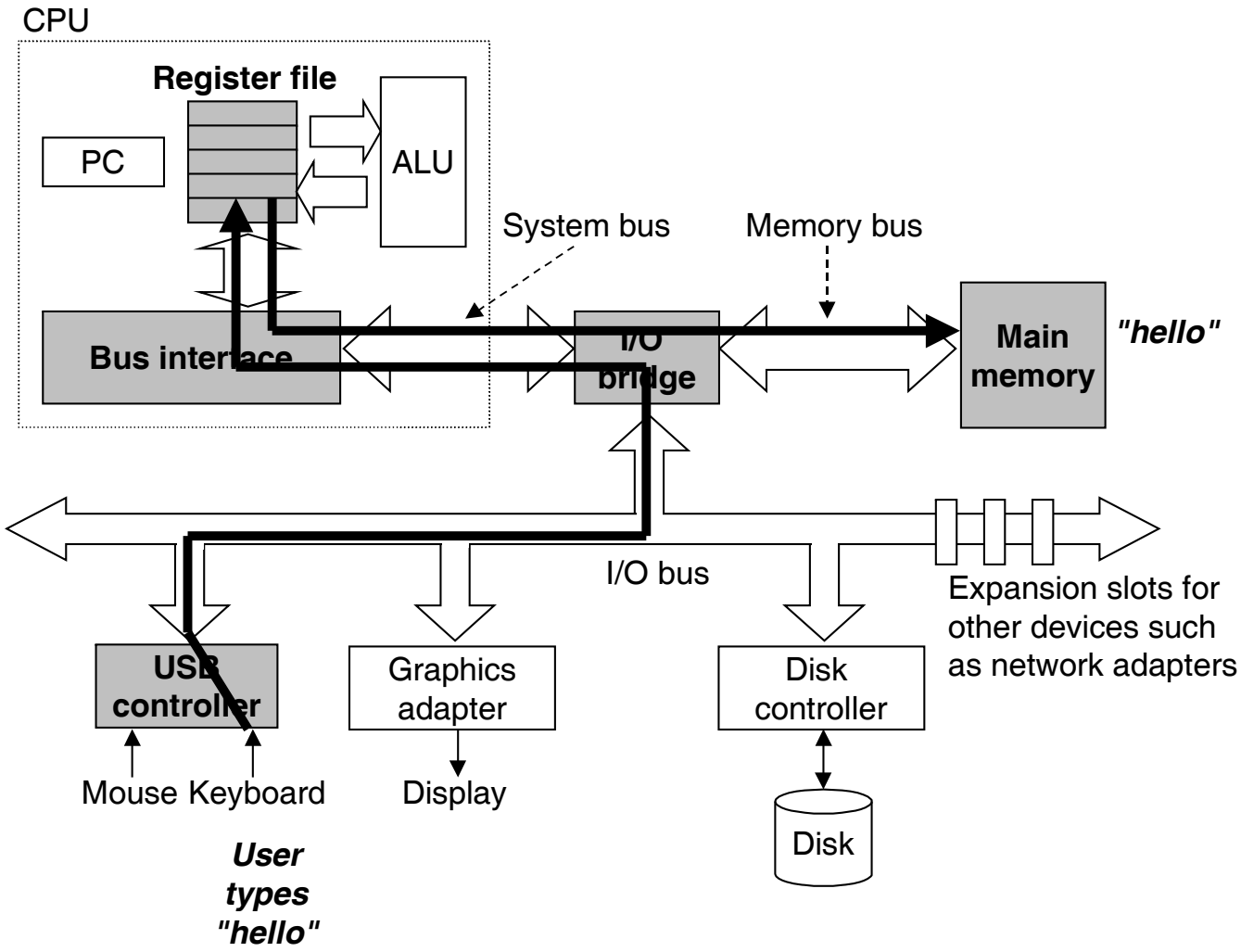
```
unix>  ./hello
hello, world
unix>
```

- **쉘에서 실행파일 이름을 치면 실행됨**
  - 쉘이 여러분이 입력한 파일 이름을 찾아서, 그것이 실행파일이면 프로그램을 로드하여 실행함
  - **hello** 프로그램이 실행되면서 결과가 화면에 출력됨.
  - 쉘이 다음 명령을 기다리고 있음
- **본질?**

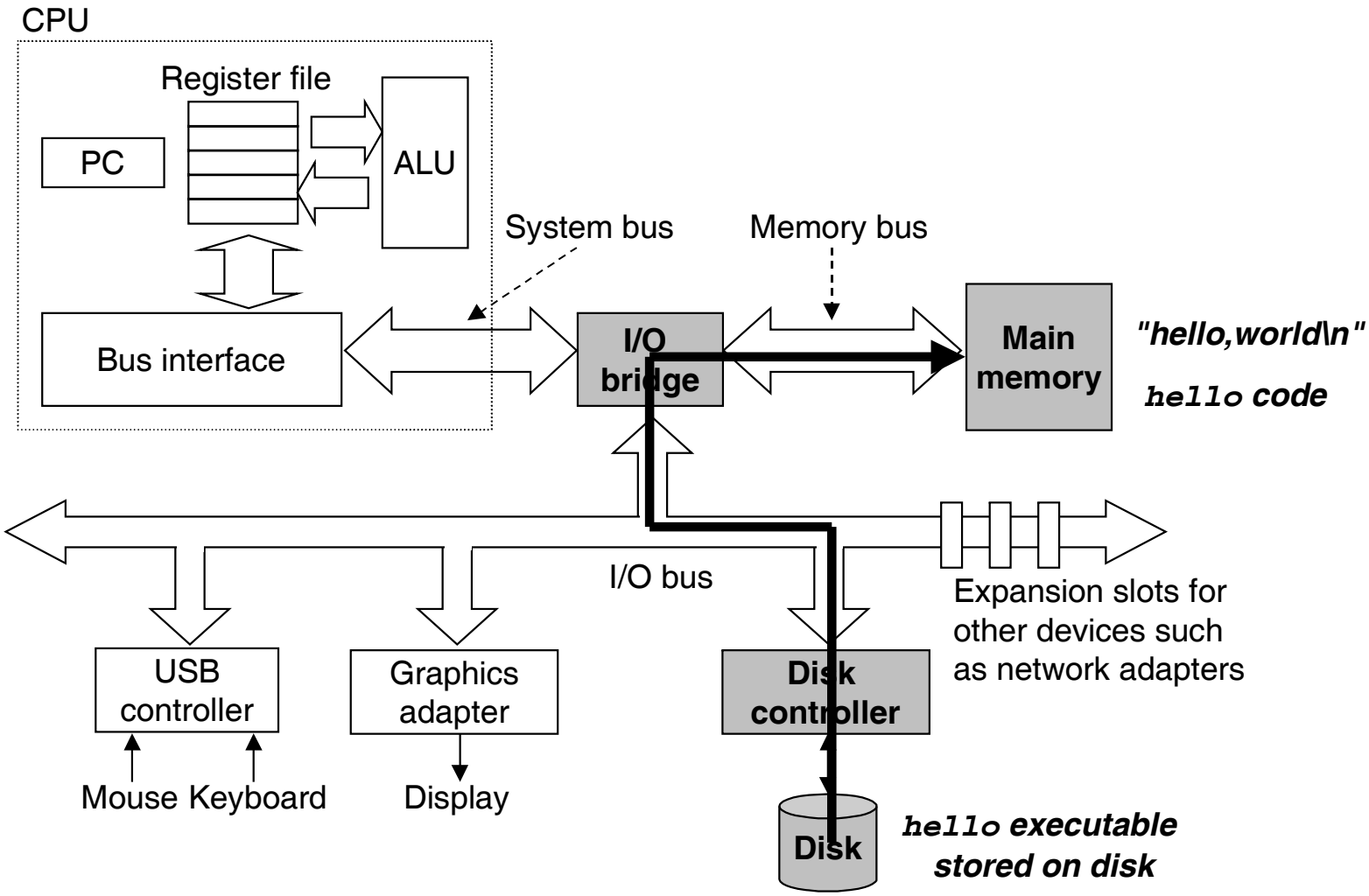
# 컴퓨터의 구조


CPU
Register file
PC
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters
USB controller
Graphics adapter
Disk controller
Mouse Keyboard
Display
Disk
hello executable stored on disk

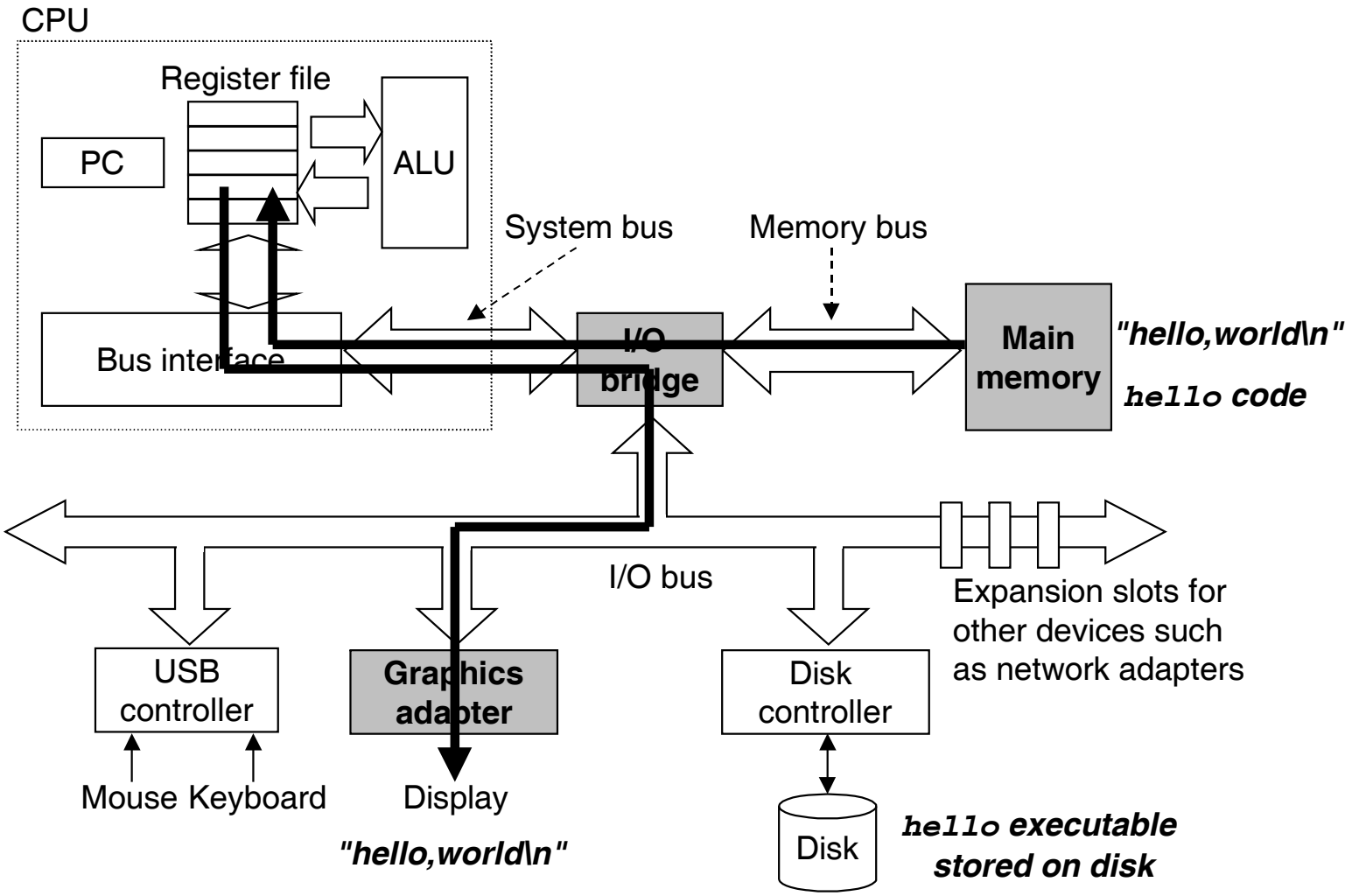
# 실행의 본질 : hello 명령의 인식



# 실행의 본질 : hello 프로그램의 로딩



# 실행의 본질 : hello 프로그램의 실행



# 요약

- 컴퓨터 시스템은 하드웨어, 시스템 소프트웨어, 응용프로그램으로 구성된다.
- 프로그램은 번역되어 변화한다.
- 프로그램은 시스템 프로그램의 도움으로 하드웨어에서 실행된다.

# 다음주를 위한 준비

- **새 학기를 맞는 여러분의 결심**
  - 강의는 열심히, 생활은 즐겁게
  - 열정적인 생활
  - 적절한 수준의 수강신청
- **예습 숙제 1.**
  - 교재 **2.1.4 Addressing and Byte Ordering(pp.73-79)** 읽고 **A4 1**쪽으로 요약
  - 다음주 수업시간에 제출
- **실습 1**
- **홈페이지 가서 강의자료 내려받기.**