C - HW3

임베디드스쿨1기

Lv1과정

2020. 08. 07

박하늘

1. 메모리 관리 체계

컴퓨터 운영체제는 cpu 레지스터 처리값에 의해 32bit와 64bit가 있으, 아직 개발 현장에서 32bit를 더 많이 사용하고 있다.

보통 32bit 운영체제의 RAM은 4GB까지 사용되는데, 레지스터는 한번에 처리 할 수 있는 데이터가 2^32 = 4,294,967,296 = 2^2 * 2^30(4GB)이다. .

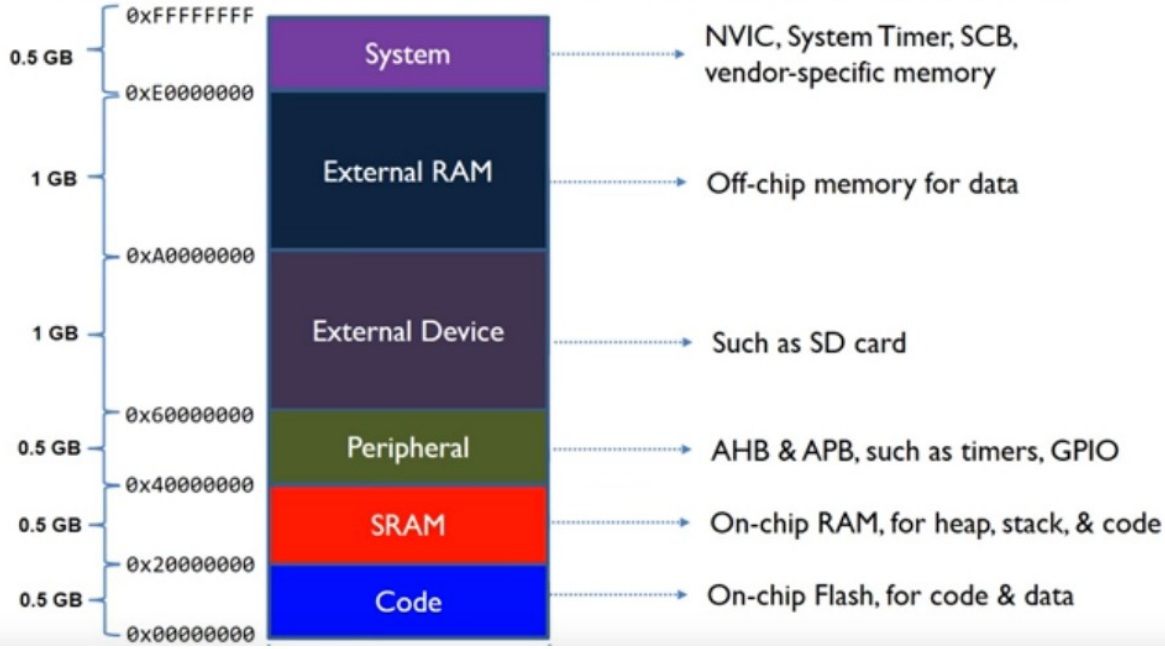
우리가 일상에서 자주 사용하는 숫자는 10진수이지만, 컴퓨터 내부는 2진수로 정보를 처리한다. C언어에서 프로그래밍할때는 2진수에 가까운 16진수를 자주 이용한다. 16진수는 2^4이므로, 한 자릿수를 4bit로 표시할 수 있다.
0x(16진수)f(4bit)f(4bit)f(4bit)f(4bit) -> 즉, 16진수의 숫자를 두자리씩 자르면 1byte(8bit)가 됨.

1) 숫자(데이터)를 어떻게 처리할까?
1234 (10진수) 이라면, 0000,0100,1101,0010 (2진수)임,
8bit씩 나눠서 0x04, 0xd2(16진수)가 된다.
최종적으로 byte 단위로 저장하면, 0x04d2가 됨 (여기서 부터 byte 단위, 한자리수는 16개 문자 0~F 사용)

C언어가 프로그래밍할때 바이트 단위로 작업을 하면, 메모리 공간을 훨씬 효율적으로 사용할 수 있다

2. 메모리맵

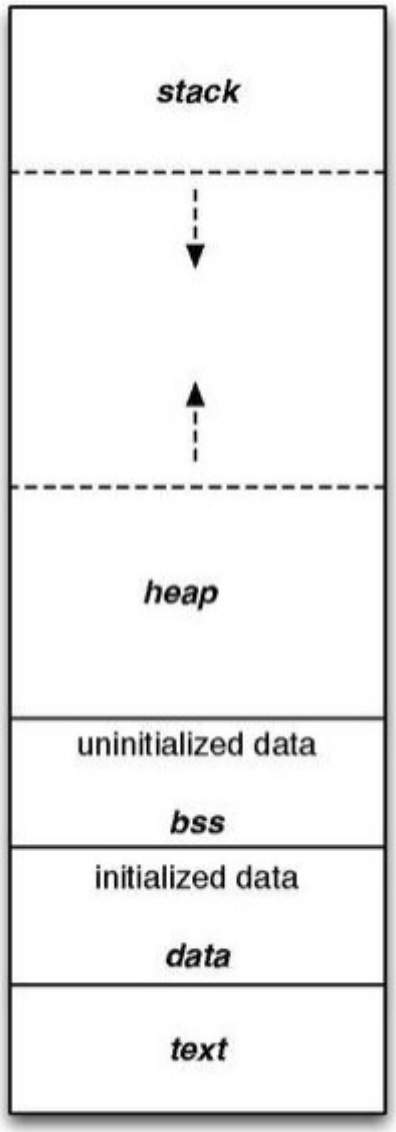
2) 메모리를 관리하는 공간은 어떻게 생겼는가?
메모리 맵은 메모리 각 영역의 역할을 나타낸 것이다.
<STM32L4 메모리 맵>



우리 사용자가 실제로 사용하는 메모리 공간은 SRAM(DATA)영역, CODE영역이 된다. SRAM은 Stack, Heap으로, Code영역은 bss, data, text가 된다.

데이터 영역은 읽고 쓰기가 가능한 영역으로 RAM공간을 가리킨다. 코드 영역은 읽기전용 영역으로 ROM 공간을 가리킨다. 즉, 메모리맵은 소프트웨어 측면에서 봤을때 ROM/RAM 등의 하드웨어의 사용영역에 대한 segment를 나타낸다.

2. 메모리맵 STACK



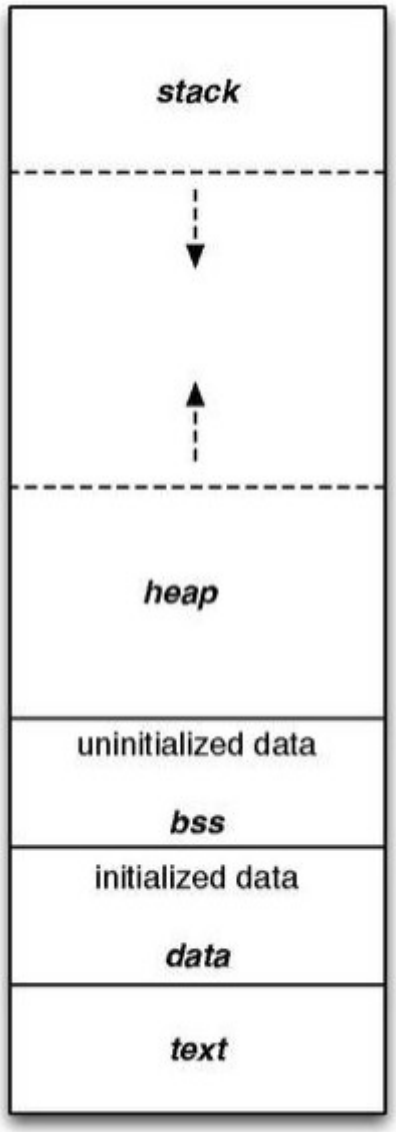
1. stack

함수가 실행될 때마다 할당되고, 끝날 때 할당이 해제되는 세그먼트 영역이다.
32bit 레지스터에서 스택은 주소 0으로 증가하며, 힙에 수렴하게 된다. (높은 주소에서 낮은 주소로 감)
Last in first out(LIFO): 맨 마지막에 들어간 값이 맨 첫번째로 나온다.

스택은 프로그램이 실행되면서 임시로 저장될 데이터들을 저장하기 위한 메모리로 일반적으로 아래와 같은 값들을 저장.

스택에 저장되는 값
ㄱ. 함수에서 생성되는 지역변수와 인수
ㄴ. 함수의 반환 값
ㄷ. 인터럽트가 발생되었을 때의 CPU Register 값
ㄹ. 함수 실행 후 되돌아갈 주소

2. 메모리맵 HEAP



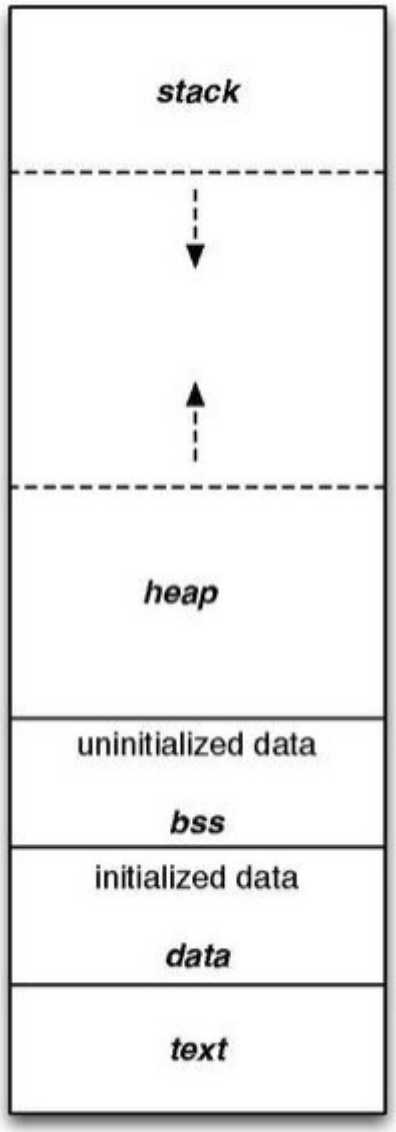
2. heap

힙은 항상 전역변수나 지역변수처럼 항상 지정된 크기의 메모리를 사용하지 않고, 프로그램 동작시 동적으로 할당되는 메모리를 위한 지정된 영역이다. 원하는 크기 선언 후 프로그램 동작시 여기서 메모리 할당해준다.

리이브러리 내장함수로 제공되는 할당(calloc, malloc)과 해제(realloc, free) 함수를 이용하여 프로그래머가 주소에 의한 접근으로 기억장소를 관리하는 방식이다. 동적 메모리 할당은 사용된 메모리의 포인터와 사용된 크기 등이 포함되기 때문에 overhead가 발생될 수 있으며 할당과 해제가 반복되면서 단편화(fragment)가 발생될 수 있다.

-> 포인터는 스택에 저장되며 동적할당되는 영역이 힙이다. 스택에 있는 포인터가 힙에 있는 할당영역을 가리키는 것이다.

2. 메모리 BSS,DATA,TEXT



3. bss
초기화되지 않은 전역 또는 정적 변수이다.

4. data
초기화된 전역 또는 정적 변수이다.
변수가 처음엔 읽기 전용 메모리에 저장되고, 프로그램 시작되면 data segment로 복사된다. 만약, 함수내에서 선언되면 지역 스택에 저장된다.

5. text
작성된 코드가 기계어를 보관하는 영역이다. 변수가 아닌 순수 코드만 있다.

3. Big,Little endian

Q3. 시스템의 메모리 정렬 방식은?

Big endian 과 Little endian이 있다. 비트 단위의 메모리 정렬은 RISC냐 CISC냐 방식 같아 상관없는데, 여러개의 바이트가 표현되는 메모리를 정렬할때는 두 방식에서 차이가 발생한다. 즉, char은 1byte라 상관없지만, int(4byte)라면 얘기가 달라짐. 0x12345678을 가지고 설명해보자.

1. Big endian
4바이트 메모리에 저장할 때 큰 자릿수의 값부터 저장한다.
(낮은주소) 0x12 0x34 0x56 0x78 (높은주소) 순으로 저장한다.
RISC기반의 하드웨어에서 동작하도록 만든 유닉스와 리눅스가 이 방식으로 바이트를 정렬한다. (JAVA 포함)

2. Little endian
4바이트 메모리에 저장할때 작은 수의 값부터 저장한다.
(낮은주소) 0x78 0x56 0x34 0x12 (높은주소) 순으로 저장한다.
CISC기반의 하드웨어에서 동작하도록 만든 운영체제가 이 방식으로 바이트를 정렬한다.
2바이트 이상의 자료형을 사용할때는 Big 이냐 Little이냐에 따라 값 자체가 달라지므로 유의해야 한다.