



C programming - 2

임베디드스쿨2기

Lv1과정

2021. 03. 24

박태인

1. 변수(1)

- 변수란 무엇 일까?

1. 알려지지 않은 값
2. 변하는 값
3. x, y, z ... 등등

정답 : 특정한 데이터 타입을 지정 할 수 있는 메모리 공간.

- 위의 정의가 중요한 이유가 무엇일까?

1. 포인터 변수 => 특정한 타입의 메모리 주소를 저장 할 수 있는 메모리 공간.
2. 배열 변수 => 특정한 타입의 연속된 메모리 주소를 저장 할 수 있는 메모리 공간.
3. 변수 => 특정한 타입을 저장 할 수 있는 메모리 공간.
4. 함수 포인터 변수 => 특정한 프로토타입의 함수 주소를 저장 할 수 있는 메모리 공간.

- 그렇다면 메모리는 무엇 일까?

메모리 계층 구조에서 속도 순은 아래와 같다.

↳ Register(레지스터) > Cache(캐시) > Memory(DRAM) > Disk

우리가 프로그래밍을 하며 메모리라 부르는 영역은 바로 Memory(DRAM)에 해당한다.

↳ 여기에서 DRAM에 대해 더 조사해 보자.(다음 페이지)

1. 변수(2) - DRAM

- DRAM이란 무엇인가

- ↳ DRAM이란 **D**ynamic **R**andom **A**ccess **M**emory로서 '동적 임의 접근 기억장치' 휘발성 메모리 소자를 나타낸다. 전원을 공급해 주어야 데이터들이 유지되는 메모리입니다. 전원 공급이 중단된다면 데이터는 날아가는 것입니다.

- DRAM의 구성

- ↳ DRAM은 **Transistor 1개와 Capacitor 1개로** 구성된다.

이는 하나의 TR(NMOS)만으로 비트요소를 구축하는 것이 불가능하기 때문입니다.

대신 DRAM의 메모리 셀은 MOS TR를 통해 접근하는 작은 C에 정보를 저장하도록 구성되어 있습니다.

여기서 셀은 가장 작은 단위라고 생각하면 됩니다. 이 Cap에 저장되어 있는 전하는 Cap의 특성상 시간이 지나면 저절로 전하가 빠져(방전) 저장한 내용이 없어지게 됩니다. 그래서 이를 위해 Cap의 전하를 Refresh 해주어야 합니다.

Refresh란 다시 전하를 채워 주는 것을 뜻한다. 이러한 이유 때문에 동적(Dynamic)이란 말이 붙게 되었습니다.

그리고 Random Access란 임의 접근을 의미하는 말로, 원하는 정보에 내 마음대로(Random하게) 접근을 할 수 있다는 뜻입니다. 예를 들어 1~50번지가 있다면 원하는 30번지에 순차적 접근이 아닌 바로 접근 가능.

- DRAM 동작 원리

- 쓰기 동작

- ↳ 목표는 Capacitor에 전하를 충전하는 것입니다.

그러기 위해서는 NMOS의 스위치를 ON 시켜야 겠죠? OFF 상태라면 Drain과 Source 사이에 Channel이 형성되어 있지 않으니 Capacitor에 전하를 보낼 수 없습니다.

NMOS의 스위치를 ON 시키기 위해서는 Word Line이라고 불리는 배선에 전압을 인가하여 주어야 합니다. 그러면 NMOS의 Gate 밑으로 Channel이 형성이 됩니다.

자, 이제 길은 만들어 졌고 전하를 Cap쪽으로 보내야 합니다.

그러기 위해서는 Source(혹은 Drain)쪽에 전압을 인가해 주어야 전하들이 움직입니다.

이러한 전압은 Bit Line이라는 배선을 통해 인가해 주면 됩니다.

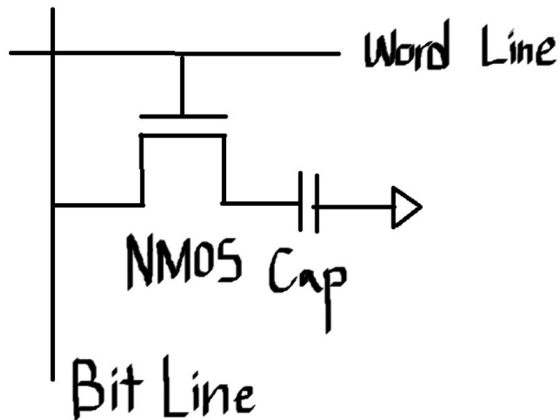
- 읽기 동작

- ↳ 데이터를 읽는다는 것은 Cap에 전하가 저장되어 있는지 없는지를 확인하는 것입니다.

전하가 있다면 1을 의미하고, 없다면 0을 의미하는 것이겠죠. 쓰기의 방법과 아주 유사합니다.

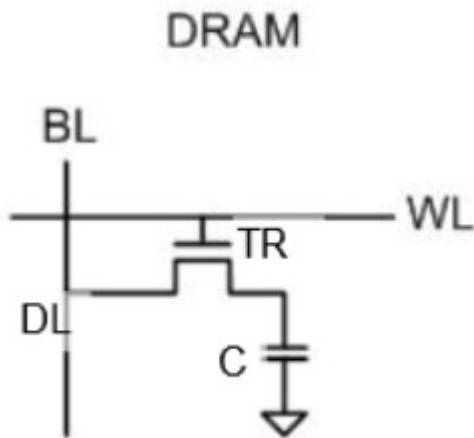
먼저 Word Line을 통해 NMOS의 스위치를 열어 주어야 합니다.

그 다음에는 Bit Line에는 0V(GND)를 인가해 줌으로써 전하가 빠져 나온다면 1이 써져 있었던 것이고, 빠져 나오지 않는다면 0이 써져 있었던 것입니다.



1. 변수(2) - DRAM

- DRAM이란 무엇인가
 - ↳ DRAM이란 **D**ynamic **R**andom **A**ccess **M**emory로서 휘발성 메모리 소자를 나타낸다.
- DRAM의 구성
 - ↳ DRAM은 **Transistor 1개와 Capacitor 1개로** 구성된다.
 - 이는 하나의 TR만으로 비트요소를 구축하는 것이 불가능하기 때문이다.
 - 대신 DRAM의 메모리 셀은 MOS TR를 통해 접근하는 작은 C에 정보를 저장하도록 구성되어 있다.
- DRAM 동작 원리



DRAM 한 비트에 대한 셀을 보면, WL(워드 선택선)에 HIGH 전압을 인가하면 접근 할 수 있다.
1을 저장하려면 BL(비트선)에 HIGH 전압을 인가하여 'on' TR을 통해 C를 충전한다.
0을 저장하려면 BL에 LOW 전압을 인가하여 C를 방전 시킨다.

반대로 상태 값을 읽기 위해서는 TR에 전류를 넣어 준다. C에 전하가 저장되어 있는 경우(1인 경우) 전하가 DL로 빠져 나올 것이고, 전하가 없는 경우(0인 경우) DL로 빠져 나오는 값이 없을 것이다.

DRAM 셀을 읽는 과정을 보다 자세히 살펴보면, BL은 먼저 HIGH와 LOW 사이의 중간 전압으로 예비 충전되고, WL(워드 선택선)이 HIGH로 된다.
C의 전압이 HIGH이냐 LOW이냐에 따라, 예비 충전된 BL은 약간 높아지거나 낮아진다.
감지 증폭기가 이 작은 변화를 검출하고, 그에 따라 1이나 0으로 복구 한다.

셀을 읽으면 축전기에 저장된 원래 전압이 파괴되므로 읽은 다음 복구된 데이터를 셀에 다시 써야한다.
DRAM 셀 내의 축전기는 C가 매우 작지만, 이에 접근하는 MOS TR의 임피던스가 매우 크다. 따라서 HIGH 전압이 LOW로 보이는 점까지 방전하는 시간이 상대적으로 매우 길다.
이러한 방법으로, C는 비트의 정보를 저장 할 수 있다.

1. 변수(3)

- 실제로 C 프로그램에서 보는 메모리는 가짜다!
 - ↳ 가짜라고 말한 이유는 진짜 DRAM에 올라가는 물리 메모리와 다르게 가상의 메모리로 데이터를 사용하고 있기 때문이다.
- 그렇다면 메모리에는 어떤 식으로 프로그램이 배치가 될까?
 - ↳ 아래와 같은 프로그램의 메모리가 어떻게 배치되는지 살펴 보자!

```
# include <stdio.h>

int main(void)
{
    int data = 3;

    printf("data = %d\n", data);

    return 0;
}
```

- ↳ 위 프로그램의 경우엔 변수에 3을 저장하고 있다.
이 데이터는 메모리 상에 아래와 같이 배치된다.(현재 여기서 메모리라 부르는 것은 가상 메모리라는 것을 상기하자!)
물리 메모리에 대한 개념은 리눅스 디바이스 드라이버를 만들면서 학습하게 될 것이다.

```
-----
|    3    | 가상 메모리 주소(0xbf238293944488f0)
-----
```

이와 같은 형식으로 저장이 된다.
만약 포인터라면? (→ 다음 페이지)

1. 변수(3)

```
# include <stdio.h>

int main(void)
{
    int data = 7;
    int *p = &data;

    printf("*p = %d\n", *p);

    return 0;
}
```

└ 위 케이스는 일반 변수와 포인터 변수가 함께 동작하는 상황이다.
이에 대한 메모리 구조는 아래와 같다.

```
-----
|      7      | 가상 메모리 주소(0xbf238293944488f0)
-----
| 0xbf238293944488f0 | 가상 메모리 주소(0xbf238293944488e8)
-----
```

이와 같은 형식으로 메모리상에 배치가 됩니다.

2. 가상 메모리(1)

- 먼저 파악해야 할 것들
 1. 비트 수 기반으로 가상 메모리 결정하는 방법
 2. 포인터의 크기는 누가 결정하는가 ?
 3. 가상 메모리의 크기는 왜 포인터의 크기를 따라 가는가?
 4. 물리 메모리의 최소 단위와 단편화 문제
 5. 그냥 다이렉트 맵핑 하면 되지 가상 메모리가 왜 필요한가?

2. 가상 메모리(2)

- 비트 수 기반으로 가상 메모리 표기 하기 !!

먼저 1비트의 변수가 가상 메모리를 표현한다 가정하면 아래와 같이 표시 할 수 있을 것이다.

1 bit -> 2 byte : 2^1

2 bit -> 4 byte : 2^2

이것을 보면 ?? 왜?? 라는 생각이 먼저 들 것이다.

정해진 것이 아닌 **(가상) 메모리 이기 때문에 규칙을 그냥 저렇게 맵핑 한 것이다.**

↳ 가상 메모리를 표기하기 위한 새로운 규칙을 만든 것 뿐

그렇다면 10bit 를 표기한다 가정해 보자!

위의 규칙을 따르면 10 bit -> 2^{10} byte가 된다.

실제로는 아래와 같은 단위 규칙이 있다.

2^{10} byte = 1 KB

2^{10} KB = 1 MB

2^{10} MB = 1 GB

위의 실제 규칙에 우리가 가상 메모리를 도입하기 위해 적용한 규칙을 함께 적용하면 아래와 같이 표현 할 수 있다.

10 bit → 2^{10} byte == 1 KB

20 bit → 2^{20} byte == 2^{10} KB == 1 MB

30 bit → 2^{30} byte == 2^{20} KB == 2^{10} MB == 1 GB

32 bit → 2^{32} byte == 1GB x 2^2 == 4 GB

2. 가상 메모리(3)

- 포인터의 크기는 누가 결정 하는가?

포인터의 크기는 실제 하드웨어 레지스터를 포함하고 있는 ALU가 결정 합니다.

왜냐하면 연산 할 수 있는 최대치가 결국 ALU에 있는 범용 레지스터에 의해 결정 되기 때문 입니다.

(ALU는 Arithmetic Logic Unit 이라 해서 컴퓨터의 연산을 담당하는 하드웨어 로직이라고 보면 됩니다.)

실제 하드웨어 레지스터는 인텔의 경우 rax, rbx, rcx, rdx와 같은 형식으로 나타나며
ARM의 경우엔 r0, r1, r2, r3, ... 형식으로 나타난다.

여기서 또 다른 오해가 발생 할 수 있는데 펌웨어 개발을 하다 보면 펌웨어를 제어하기 위한 레지스터가 존재한다.
현재 케이스에서 거론되는 레지스터는 연산용 레지스터 이므로 펌웨어를 제어하기 위한 레지스터와는 별개 입니다.
(실제로 펌웨어는 ARM아키텍처 이므로 범용 레지스터는 r0, r1, r2, ... 형식으로 활용됨)

64비트 운영체제에서는 범용 레지스터가 64비트 이므로 포인터의 크기가 8 바이트가 된다.

범용 레지스터는 실제 프로그램이 구동되는데 필요한 레지스터 이며

제어 레지스터는 특정한 하드웨어를 구동 시키기 위한 옵션이라고 생각하면 된다.

2. 가상 메모리(4)

• 범용 레지스터 vs Peripheral 제어 레지스터

둘을 구별 해야 합니다.

실제 프로그램이 구동 될 때 동작하는 CPU 연산은 범용 레지스터가 담당 합니다.

반면 Peripheral 의 장치를 활성화하고 제어하는 목적으로는 Peripheral 제어 레지스터가 활용 됩니다.

이 름	범용 레지스터 (General-purpose register)	주변장치 제어 레지스터 (Peripheral-purpose register)																		
정 의	명령에 사용되는 데이터를 임시로 저장하는, CPU 내부에서 볼 수 있는 유형의 레지스터	주변 장치 모듈에 대한 설정 값이나 측정 결과를 저장하는 레지스터																		
종 류	ALU (산술연산, 논리연산), Index register(인덱스) 등	타이머, USART, ADC 등																		
예 시	<table><tr><td>EAX</td><td>사칙연산 중 곱셈과 나눗셈에 사용되며, 함수의 반환값을 저장</td></tr><tr><td>EBX</td><td>ESI 또는 EDI 와 결합하여 인덱스에 사용되며, 산수와 변수를 저장</td></tr><tr><td>ECX</td><td>반복 카운터를 저장. ECX에 반복할 횟수를 지정하여 반복레지스터가 반복 수행.</td></tr><tr><td>EDX</td><td>EAX 보조, 부호 확장 명령 등에 사용</td></tr><tr><td>ESI</td><td>출발지 주소 저장</td></tr><tr><td>EDI</td><td>목적지 주소 저장. 주로 ESI 레지스터가 가리키는 주소 데이터 복사됨.</td></tr><tr><td>EBP</td><td>스택 프레임의 시작주소 저장(스택포인터의 기준점 저장)</td></tr><tr><td>ESP</td><td>스택 프레임의 끝 지점 주소 저장</td></tr><tr><td>EIP</td><td>다음 실행할 명령어주소 저장</td></tr></table>	EAX	사칙연산 중 곱셈과 나눗셈에 사용되며, 함수의 반환값을 저장	EBX	ESI 또는 EDI 와 결합하여 인덱스에 사용되며, 산수와 변수를 저장	ECX	반복 카운터를 저장. ECX에 반복할 횟수를 지정하여 반복레지스터가 반복 수행.	EDX	EAX 보조, 부호 확장 명령 등에 사용	ESI	출발지 주소 저장	EDI	목적지 주소 저장. 주로 ESI 레지스터가 가리키는 주소 데이터 복사됨.	EBP	스택 프레임의 시작주소 저장(스택포인터의 기준점 저장)	ESP	스택 프레임의 끝 지점 주소 저장	EIP	다음 실행할 명령어주소 저장	<div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>
EAX	사칙연산 중 곱셈과 나눗셈에 사용되며, 함수의 반환값을 저장																			
EBX	ESI 또는 EDI 와 결합하여 인덱스에 사용되며, 산수와 변수를 저장																			
ECX	반복 카운터를 저장. ECX에 반복할 횟수를 지정하여 반복레지스터가 반복 수행.																			
EDX	EAX 보조, 부호 확장 명령 등에 사용																			
ESI	출발지 주소 저장																			
EDI	목적지 주소 저장. 주로 ESI 레지스터가 가리키는 주소 데이터 복사됨.																			
EBP	스택 프레임의 시작주소 저장(스택포인터의 기준점 저장)																			
ESP	스택 프레임의 끝 지점 주소 저장																			
EIP	다음 실행할 명령어주소 저장																			

2. 가상 메모리(5)

- 가상 메모리의 크기는 왜 포인터의 크기를 따라가는가?

범용 레지스터의 크기가 64비트 이므로 표현할 수 있는 바이트는 8 바이트에 해당 합니다.
그렇다면 하드웨어 특성상 여기서 64비트를 모두 활용하지 않는 것은 비용 낭비에 해당 할 것 입니다.

안 그래도 반도체 공정상 비싼 하드웨어에 속도가 빠른 범용 레지스터는 그 가격의 손해가 더욱 클 것입니다.
그래서 64비트를 다 사용하지 않고 35비트 정도만 사용한다면 엄청난 손해를 보는 격 입니다.

그러므로 64비트를 모두 포함하면서 효율적으로 사용 할 수 있는 방법을 찾고자 했습니다.
가만 보니 가상 메모리를 64비트로 표현하면 8 바이트를 모두 표현 할 수 있었던 것 입니다.

2. 가상 메모리(6)

- 물리 메모리의 최소 단위와 단편화 문제

물리 메모리의 최소 단위는 **페이지 프레임(Page Frame)**이라고 하여 **4 KB(4096 바이트)**를 기본 단위로 지정합니다.
이 때 우리가 메모리를 마구잡이로 할당 하고 관리하게 되면 어떤 일이 발생할지 생각해 보는 것 입니다.
프로그램에서 사용하는 메모리 크기가 80 바이트 짜리를 수 만번 할당 한다고 가정해 봅니다.

페이지 프레임에 80 바이트를 저장한다.
그리고 다른 페이지 프레임에 또 다시 80 바이트를 저장합니다.
단편화(Fragmentation)이 발생한다.

$4096 - 80 = 4012$ 바이트의 손해가 발생함

대충 곱하기 10000을 하면 $4012 \times 10000 \Rightarrow 40 \text{ MB}$ 의 손해가 발생합니다.
 $80 * 10000 = 800000 \Rightarrow$ 사용량은 800 KB 밖에 안됩니다.

이런 상황이 발생하다 보니 **메모리를 관리할 수 있는 시스템**이 필요해 졌다.
그것의 일환으로 **버디 할당자와 슬랩 할당자가 등장**하게 된다.

- 사용자가 최소할당 크기(4KB)보다 작은크기(ex : 40 byte or 80 byte 등)를 할당 요청할 경우,
4KB를 할당하면 **internal fragmentation(내부 단편화)**가 발생하고,
이를 해결하기 위해 **Slab Allocator(슬랩 할당자)**를 사용한다.
- 최소할당 크기(4KB) 보다 큰 크기(ex : 10 KB or 12 KB :: 등의 16KB 이하의 크기)을 할당 요청할 경우,
External Fragmentation(외부 단편화)가 발생하기 때문에
16KB를 할당해주는 Buddy Allocator(버디 할당자)를 사용한다.

아래와 같이 **작은 단위의 메모리 할당이 필요한 경우 슬랩 할당자**가 동작해서 쪼개서 값을 나눠주며
반대 값이 **큰 메모리를 할당 할 경우엔 버디 할당자가 동작**해서 여러 페이지 프레임을 제공해 주는 방식으로 동작하게 된다.

프로세스들이 여러 개가 같이 돌기 시작하는 상황이다.
이 경우엔 물리 메모리에 서로 값을 쓰겠다고 충돌이 미친듯이 발생하게 된다.
이것을 전반적으로 유연하게 관리해줄 무언가가 필요 했고,
그것으로 페이징 정책을 택해 가상 메모리 ↔ 물리 메모리 방식을 채택 했다고 봐도 무방하다.

2. 가상 메모리(7)

- 그냥 다이렉트 맵핑 하면 되지 가상 메모리가 왜 필요 한가?

다이렉트로 맵핑 하였는데, 다음에 들어오는 데이터가 기존에 다이렉트 맵핑한 메모리 데이터 보다 더 큰 것이 들어 온다면???

답이 없는 상황이 도출 된다는 것을 바로 알 수 있다.

끝도 없이 대규모 데이터가 빈번하게 들어오는 레이더 시스템이나 영상 시스템 같은 경우엔 다이렉트 맵핑을 한다는 것 자체가 말도 안되는 이야기가 되는 것이다.

가상 메모리 시스템에서는 물리 메모리로 변환한 데이터를 페이지라고 하는데 필요에 따라서 /swap 파티션에 이 페이지 데이터를 캐싱 해두는 목적으로 활용하게 된다.

- 메모리 할당

1. 페이지징 기법 이란?

- └ 컴퓨터가 메인 메모리에서 사용하기 위해 2차 기억장치로부터 데이터를 저장하고 검색하는 메모리 관리 기법이다.
즉, 가상기억장치를 모두 같은 크기의 블록으로 편성하여 운용하는 기법이다.
이때의 일정한 크기를 가진 블록을 페이지(page)라고 한다.
주소공간을 페이지 단위로 나누고 실제 기억공간은 페이지 크기와 같은 프레임으로 나누어 사용한다.

2. 프레임과 페이지

- └ 프레임과 페이지는 메모리를 일정한 크기의 공간으로 나누어 관리하는 단위이며, 프레임과 페이지의 크기는 같다.
 - ▶ 프레임(Frame) : 물리 메모리를 일정한 한 크기로 나눈 블록이다.
 - ▶ 페이지(Page) : 가상 메모리를 일정한 한 크기로 나눈 블록이다.

3. C 코딩 예제(1)

- For, if 문 예제 (0~20 미만 숫자 중 짝수만 출력 하기)

```
#include <stdio.h>

int main(void)
{
    int i;

    for(i=0; i<20; i++)
    {
        // !는 not 연산으로 결과 자체를 부정합니다.
        // 비트 연산자는 아니며 논리 연산자 입니다.
        // 비트 연산자 not은 ~로 사용합니다.

        // %는 나머지 연산자로 정수론에서는 모듈이라고 부르는 녀석입니다.
        // 즉 2로 나누는 나머지 값을 보겠다는 것인데
        // 짝수는 2로 나눈 나머지가 무조건 0 입니다.
        // 이 결과 자체를 not을 통해 부정하니 0(거짓)이 1(참)이 됩니다.
        // 그래서 짝수만 출력하는 구조를 가지게 됩니다.
        if(!(i%2))
        {
            printf("%d ", i);
        }
    }
    printf("\n");

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./for_if_test
0 2 4 6 8 10 12 14 16 18
```

3. C 코딩 예제(2)

- For, if 문 예제 (0~20 미만 숫자 중 짝수만 출력 하기), +2 씩 하는 방법

```
#include <stdio.h>

int main(void)
{
    int i;

    for(i=0; i<20; i += 2)
    {
        printf("%d ", i);
    }
    printf("\n");

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./for_if_test2
0 2 4 6 8 10 12 14 16 18
```

3. C 코딩 예제(3)

- For 문 예제 (0~9 까 출력 하기)

```
#include <stdio.h>

int main(void)
{
    int i;

    printf("0 ~ 9까지 출력해보자!\n");
    // 이것을 공식적으로 허용하는 것은 C++ 부터 입니다.
    // 그래서 표준을 따르는 리눅스에서는 이것의 컴파일이 막힙니다.
    // 특히나 임베디드 개발 할 때 이부분은 컴파일 오류가 나는 부분이니 주의 하도록 합시다.
    // 아래와 같이 for 문 안에 int형으로 동시에 초기화 하고 저런 부분을 애기 하는 것

    // for(int i=0; i<10; i++)
    // for문의 경우엔 구조가 꽤 복잡하다.

    // for문을 작성하는 방법
    // 1. for () {} 를 적는다.
    // 2. 필요한 초기화 조건을 작성한다. 없으면 안적어도 된다.
    // 3. 반복시킬 때 사용할 조건을 적는다. 없으면 무한 반복이 된다.
    // 4. 증감부에 증가시키거나 감소시킬 내용을 작성한다. 없으면 안적어도 된다.
    // 5. 조건이 만족할 경우 구동될 코드를 중괄호 내부에 작성한다.

    // for문의 동작 방식
    // 1. 최초 진입시 초기화 코드가 실행됨
    // 2. 조건을 보고 조건이 참 이라면 중괄호 코드를 실행함
    // 3. 중괄호 코드가 실행된 이후 증감부의 내용을 실행함
    // 4. 조건이 거짓이 될 때 까지 2번, 3번 내용을 반복함
    // 5. 조건이 거짓이 되면 for 문이 끝난다.

    // i++에서 ++은 더하기 1과 같다.
    // --는 빼기 1과 같다.
    // i += 2 는 i = i + 2와 같다.
    // i -= 3 은 i = i - 3과 같다.
    // i *= 10은 i = i * 10과 같다.
    for(i=0; i<10; i++)
    {
        printf("%d ", i);
    }
    printf("\n");

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./for_test
0 ~ 9까지 출력해보자!
0 1 2 3 4 5 6 7 8 9
```


3. C 코딩 예제(4)

• if 문 예제 (숫자 대소 비교 하기)

```
#include <stdio.h>

int main(void)
{
    int num1, num2;

    // 아래와 같은 형식으로 정수 두 개를 입력 받을 수 있습니다.
    printf("두 개의 정수를 입력 하세요: ");
    scanf("%d %d", &num1, &num2);

    printf("num1 = %d\n", num1);
    printf("num2 = %d\n", num2);

    // if 문을 작성하는 방법
    // 1. 먼저 if ( ) {} 를 적습니다.
    // 2. 소괄호 내부에 사용하고자 하는 조건을 적습니다.
    //    조건은 대소 비교 등등을 사용 할 수 있습니다.
    //    (같냐 ==, 다르냐 !=, 크거나 같냐 >=, 작거나 같냐 <= 등을 사용 할 수 있음)
    // 3. 조건을 적었으면, 아래쪽에 if문 내부에서 사용하려는 본문을 작성합니다.
    //    조건이 만족되었을 경우 구동될 코드를 적습니다.
    //    조건이 만족되지 않았을 경우엔 else로 이동합니다.
    if (num1 > num2)
    {
        printf("num1(%d)가 num2(%d) 보다 큼니다.\n", num1, num2);
    }
    else
    {
        printf("num2(%d)가 num1(%d) 보다 큼니다.\n", num2, num1);
    }

    // 여기엔 문제점이 있습니다. ====> 같은 경우엔 처리가 이상하게 됩니다.

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./if_test
두 개의 정수를 입력 하세요: 1 2
num1 = 1
num2 = 2
num2(2)가 num1(1) 보다 큼니다.
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./if_test
두 개의 정수를 입력 하세요: 3 1
num1 = 3
num2 = 1
num1(3)가 num2(1) 보다 큼니다.
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./if_test
두 개의 정수를 입력 하세요: 2 2
num1 = 2
num2 = 2
num2(2)가 num1(2) 보다 큼니다.
```

3. C 코딩 예제(5)

- if 문 예제 (숫자 대소 비교 하기) - 향상된 버전.

```
#include <stdio.h>

int main(void)
{
    int num1, num2;

    // 아래와 같은 형식으로 정수 두 개를 입력 받을 수 있습니다.
    printf("두 개의 정수를 입력하세요 : ");
    scanf("%d %d", &num1, &num2);

    printf("num1 = %d\n", num1);
    printf("num2 = %d\n", num2);

    // 범주를 여기서 크게 잡아 버리면
    // 아래쪽에서 범주가 작은 녀석들을
    // 큰 녀석이 포함관계로 전부 덮어버릴 수가 있으니 주의

    // 입력 값이 0 ~ 99
    // if (100 > num)
    // else if(50 > num)
    // else if(20 > num)

    // 입력 값이 0 ~ 99
    // if (20 > num)
    // else if(50 > num)
    // else if(100 > num)
    if (num1 > num2)
    {
        printf("num1(%d)가 num2(%d) 보다 큼니다.\n", num1, num2);
    }
    else if(num1 < num2)
    {
        printf("num2(%d)가 num1(%d) 보다 큼니다.\n", num2, num1);
    }
    else
    {
        printf("num1(%d)와 num2(%d)는 서로 같습니다.\n", num1, num2);
    }
    // 첫 번째 조건이 만족되지 않았을 경우
    // 또 다른 조건을 추가하고 싶다면 else if 형식을 사용하면 됩니다.

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./if_test_improve
두 개의 정수를 입력하세요 : 2 1
num1 = 2
num2 = 1
num1(2)가 num2(1) 보다 큼니다.
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./if_test_improve
두 개의 정수를 입력하세요 : 1 2
num1 = 1
num2 = 2
num2(2)가 num1(1) 보다 큼니다.
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./if_test_improve
두 개의 정수를 입력하세요 : 1 1
num1 = 1
num2 = 1
num1(1)와 num2(1)는 서로 같습니다.
```

3. C 코딩 예제(6)

• Scanf 예제

```
#include <stdio.h>

int main(void)
{
    int num;

    // 여기서 보이는 &는 주소값을 주세요 ~
    // 라는 의미를 가지고 있는 녀석으로
    // 메모리 주소를 달라는 뜻 입니다.

    // 즉, num 변수에 주소에 제가 키보드로 입력한 값을 넣어 주세요가 scanf 입니다.
    printf("원하는 정수 값을 입력해 보세요: ");
    scanf("%d", &num);

    printf("입력한 정수값은 = %d\n", num);

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./scanf_test
원하는 정수 값을 입력해 보세요: 7
입력한 정수값은 = 7
```

4. C 코딩 연습

실습을 하면서 학습했던 if, for문을 활용한 문제

2. 0 ~ 100까지 숫자중 홀수만 출력하시오.
3. 위의 문제를 풀 때 한 번에 붙어서 나오면 가독성이 떨어지니 10개 단위로 출력해주세요.
4. 1 ~ 50까지 숫자의 합을 구하는 프로그램을 작성하세요.
5. 1 ~ 33까지의 숫자중 3의 배수의 합만 구해보세요.
6. 1 ~ 100의 숫자중 2의 배수의 합과 3의 배수의 합을 각각 구해봅시다.
7. 피보나치 수열의 n 번째 항을 구하는 프로그램을 만들어봅시다.(배운 내용으로만 가능)
8. 1, 1, 1, 1, 2, 3, 4, 5, 7, 10, 14, 19, 26, 36, 50, ... 으로 진행되는 수열이 있다.
여기서 25번째 항의 숫자값을 구해봅시다.

4. C 코딩 연습 (1) ◆ 0 ~ 100 까지 홀수만 출력하기 & 가독성을 위해 10개 단위로 출력하기.

```
#include <stdio.h>

int main(void)
{
    int i;
    int count=1;

    for(i=0;i<100;i++)
    {
        //2로 나누었을 때, 짝수는 숫자가 0으로 떨어진다.
        //그런데 0은 거짓이므로 if문 안으로 들어가지 않는다.
        //따라서 홀수 일 경우 나머지가 1이 되어 참이 되고
        //if문이 동작하여 그 숫자를 출력하게 된다.
        if(i%2)
        {
            printf("%d ",i);
            count++;

            //홀수의 숫자를 계속 프린트 하다가 10번 이상 숫자를 찍을 경우
            //if문으로 들어가서 다음줄로 커서를 옮기고 count는 다시 제로 셋팅 한다.
            if(count>10)
            {
                printf("\n");
                count=1;
            }
        }
        printf("\n");

        return 0;
    }
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_1
1 3 5 7 9 11 13 15 17 19
21 23 25 27 29 31 33 35 37 39
41 43 45 47 49 51 53 55 57 59
61 63 65 67 69 71 73 75 77 79
81 83 85 87 89 91 93 95 97 99
```

4. C 코딩 연습 (2) ◆ 1 ~ 50까지의 숫자의 합 구하기.

```
#include <stdio.h>

int main(void)
{
    int i;
    int sum = 0;

    // i 변수는 1 ~ 50 까지의 값을 순차적으로 넣게 된다.
    // sum += i;는 sum = sum + i 가 되므로
    // sum = 0(sum의 초기 값) + 1 => sum : 1
    // sum = 1(sum의 이전 값) + 2 => sum : 3
    // sum = 3(sum의 이전 값) + 3 => sum : 6
    // 이런식으로 50 까지의 합을 sum에 축적하게 된다.

    for(i=1; i<=50; i++)
    {
        sum += i;
    }
    printf("1에서 50까지의 합은 %d 입니다. \n", sum);

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_2
1에서 50까지의 합은 1275 입니다.
```


4. C 코딩 연습 (3) ◆ 1 ~ 33 까지의 숫자 중 3의 배수의 합만 구하기

```
#include <stdio.h>

int main(void)
{
    int i;
    int sum = 0;

    // 1 ~ 33 까지의 숫자를 i 에 대입
    for(i=1;i<=33;i++)
    {
        // 3의 배수 일 때 if 문 진입
        if(!(i%3))
        {
            // 진입한 3의 배수를 sum에 합산
            sum += i;
        }
    }
    printf("1 ~ 33 까지의 숫자 중 3의 배수의 합은 %d 입니다. \n", sum);

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_3
1 ~ 33 까지의 숫자 중 3의 배수의 합은 198 입니다.
```

4. C 코딩 연습 (4) ◆ 1 ~ 100 까지의 숫자 중 2의 배수의 합과 3의 배수의 합을 각각 구해보자.

```
#include <stdio.h>

int main(void)
{
    int i, i_2, i_3;
    int sum_2=0;
    int sum_3=0;

    // 3의 배수의 범위가 더 좁은 조건이기 때문에 2의 배수 보다 더 위에 if문을 배치함

    // 1 ~ 100 까지의 숫자를 i 에 대입
    for(i=1;i<=100;i++)
    {
        // 3의 배수 일 때 if 문 진입
        if(!(i%3))
        {
            // 3의 배수의 숫자를 i_3으로 전달
            i_3 = i;
            // i_3에 들어간 3의 배수를 sum_3 으로써 합산
            sum_3 += i_3;
        }
        // 2의 배수 일 때 if 문 진입
        else if(!(i%2))
        {
            // 2의 배수의 숫자를 i_2으로 전달
            i_2 = i;
            // i_2에 들어간 3의 배수를 sum_2으로써 합산
            sum_2 += i_2;
        }
    }
    printf("1 ~ 100 까지의 숫자 중 3의 배수의 합은 %d\n", sum_3);
    printf("1 ~ 100 까지의 숫자 중 2의 배수의 합은 %d\n", sum_2);

    return 0;
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_4
1 ~ 100 까지의 숫자 중 3의 배수의 합은 1683
1 ~ 100 까지의 숫자 중 2의 배수의 합은 1734
```


4. C 코딩 연습 (5) ◆ 피보나치 수열 n 번째 항 구하는 코딩

```
// 피보나치 수열 1(첫번째), 1(두번째), 2, 3, 5, 8, 13, 21, ...  
// n항의 값이 (n-1),(n-2)의 합으로 이루어 진다.  
// 예를 들어 t1, t2가 0번째, 1번째 항 이라면 더해진 값은 2번째 항이 되며  
// 3번째 항은 1번째, 2번째 항의 합이 된다.  
// next = t1 + t2;  
// t1 = t2;  
// t2 = next;
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i, n, next;
```

```
    // 1번째 값
```

```
    int t1 = 1;
```

```
    // 2번째 값
```

```
    int t2 = 1;
```

```
    printf("출력할 피보나치 수열의 항의 N항 : ");
```

```
    // 몇 번째 항까지 출력 할 건지 입력
```

```
    scanf("%d",&n);
```

```
    printf("피보나치 수열 : %d\n",n);
```

```
    // N 항까지 for문 돌려서 출력
```

```
    for(i=1; i<n; i++)
```

```
    {
```

```
        printf("%d ", t1);
```

```
        next = t1 + t2;
```

```
        t1 = t2;
```

```
        t2 = next;
```

```
    }
```

```
    printf("%d ", t1);
```

```
    printf("\n");
```

```
    printf("따라서 N 번째 항의 값은 : %d !\n ", t1);
```

```
    return 0;
```

```
}
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_5
```

```
출력할 피보나치 수열의 항의 N항 : 7
```

```
피보나치 수열 : 7
```

```
1 1 2 3 5 8 13
```

```
따라서 N 번째 항의 값은 : 13 !
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_5
```

```
출력할 피보나치 수열의 항의 N항 : 9
```

```
피보나치 수열 : 9
```

```
1 1 2 3 5 8 13 21 34
```

```
따라서 N 번째 항의 값은 : 34 !
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_5
```

```
출력할 피보나치 수열의 항의 N항 : 10
```

```
피보나치 수열 : 10
```

```
1 1 2 3 5 8 13 21 34 55
```

```
따라서 N 번째 항의 값은 : 55 !
```

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_5
```

```
출력할 피보나치 수열의 항의 N항 : 24
```

```
피보나치 수열 : 24
```

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
```

```
따라서 N 번째 항의 값은 : 46368 !
```

4. C 코딩 연습 (6) ◆ 1, 1, 1, 1, 2, 3, 4, 5, 7, 10, 14, 19, 26, 36, 50, ... 으로 진행 되는 수열의 25번째 항을 구하여라.

```
// 일단 규칙을 찾아 보는게 먼저이다.
// Arr[0] ...
// 1 1 1 1 2 3 4 5 7 10 14 19 26 36 50
// 0 0 0 1 1 1 1 2 3 4 5 7 10 14 19
// 위와 같은 규칙을 알 수 있다. 값들의 합산 차이를 보니 규칙되는 수열 수의 n-4항 만큼의 숫자를 그대로 더하는 것을 알 수 있다.
// 위의 수열을 a 라고 생각해 보자. 그것의 5번째를 a[5]라고 나타내 본다.
// 즉, a[n] = a[n-1] + a[n-4]
// ex : a[5] = a[4] + a[1] = 1 + 1 = 2
// ex : a[8] = a[7] + a[4] = 4 + 1 = 5
// ex : a[13] = a[12] + a[9] = 19 + 7 = 26

#include <stdio.h>

int main(void)
{
    int n,i;
    int a;
    int b;
    int Arr[100]={1,1,1,1,2,3,4,5,7,10,14,19,26,36,50};
    int over;

    printf("몇 번째 항의 값을 원하십니까 : ");
    scanf("%d",&n);

    for(i=0;i<=n;i++)
    {
        //n=i;
        a=n-1;
        b=n-4;
        if(a<0 || b<0)
        {
            Arr[n]=1;
        }
        else
        {
            Arr[n]=Arr[a]+Arr[b];
            // printf("%d + %d = %d ",Arr[a], Arr[b], Arr[n]);
        }
        // a[5]는 6번째(n+1) 항이다.
        n = n-1;
        printf("N 번째 항의 값은 %d 이다. \n",Arr[n]);

        return 0;
    }
}
```

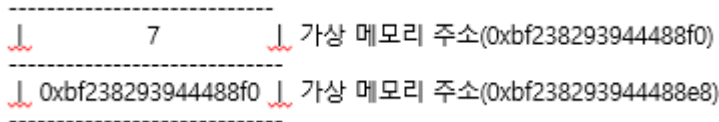
```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_6
몇 번째 항의 값을 원하십니까 : 7
N 번째 항의 값은 4 이다.
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_6
몇 번째 항의 값을 원하십니까 : 13
N 번째 항의 값은 26 이다.
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_6
몇 번째 항의 값을 원하십니까 : 15
N 번째 항의 값은 50 이다.
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/homework/c/02$ ./practice_6
몇 번째 항의 값을 원하십니까 : 16
N 번째 항의 값은 0 이다.
```

얻는 건 아무것도 없다.

Q. 질문

1. 특정한 타입과 프로토 타입은 다른 의미 인가요?

2. DRAM에서 저절로 방전되는 것을 막기 위해 Refresh 동작을 한다고 하는데, 그 동작 역시 Word Line, Bit Line을 통해서 적절히 하는 것 인가요?(프로그래밍이 되어 있다고 보면 되는 건가요?)

3. 

↳ 여기에서 끝자리가 f0 → e8 이 되는 계산은 어떻게 되는 것 인가요?

4. 가상메모리에서 가상메모리 할당 규칙을 1 bit -> 2^1 byte 같은 방식으로 나타내도록 한다면, 계산상 32bit가 4GB가 되었습니다.
이러한 규칙은 실제로도 통용 되는 것 인가요? 예를 들어 1bit -> 2^{10} byte 의 규칙으로 생각한다면 32bit의 값은 40GB가 되는데, 가상이기 때문에 그렇게 생각해도 무방 한 건가요?

5. 가상메모리에서 32bit가 4GB라 가정하면 윈도우나 리눅스 등에서 한 프로세스가 실행하는 가상 메모리가 이정도 사용된다는 말인가요?
그렇다면 여러가지 프로그램(프로세스)가 동시에 실행될 때 (ex:10개의 프로세스) 40GB의 가상메모리가 사용되는 것 인가요? 그리고 가상메모리 용량은 한계가 없나요??

6. 메모리에서 64비트를 모두 사용하면서 효율적으로 사용 할 수 있는 방법이 가상메모리를 64비트로 표현하는 것 이라 하셨는데, 가상 메모리를 사용 했을 때 효율적 일 수 있는 이유는 바로 DRAM에 데이터를 쓰게 되면 그 보다 많은 양의 데이터가 한번에 들어 왔을 때 처리하기가 힘들기 때문인가요?
만약 그렇다면 한번에 32 byte의 데이터가 들어왔다고 생각 했을 때, 가상메모리를 8 byte(64bit)씩 4개로 32 byte로 구성하고 이것을 차례로 8 byte(64bit)씩 DRAM에 쓰는 것이라고 생각하면 될까요?

7. 리눅스 커널, 메모리 할당에 관한 내용은 Level 2에서 더 자세히 배우게 될까요?

8. 마지막 연습문제에서 숫자의 대한 규칙을 for 나 if 문 만으로 나타내기가 어려워서 배열로 나타내어 보긴 했는데, 그마저도 잘 되진 않은 것 같습니다.. 어떤식에 중점을 두는게 좋을지 혹은 팁이 있는지 궁금합니다..!