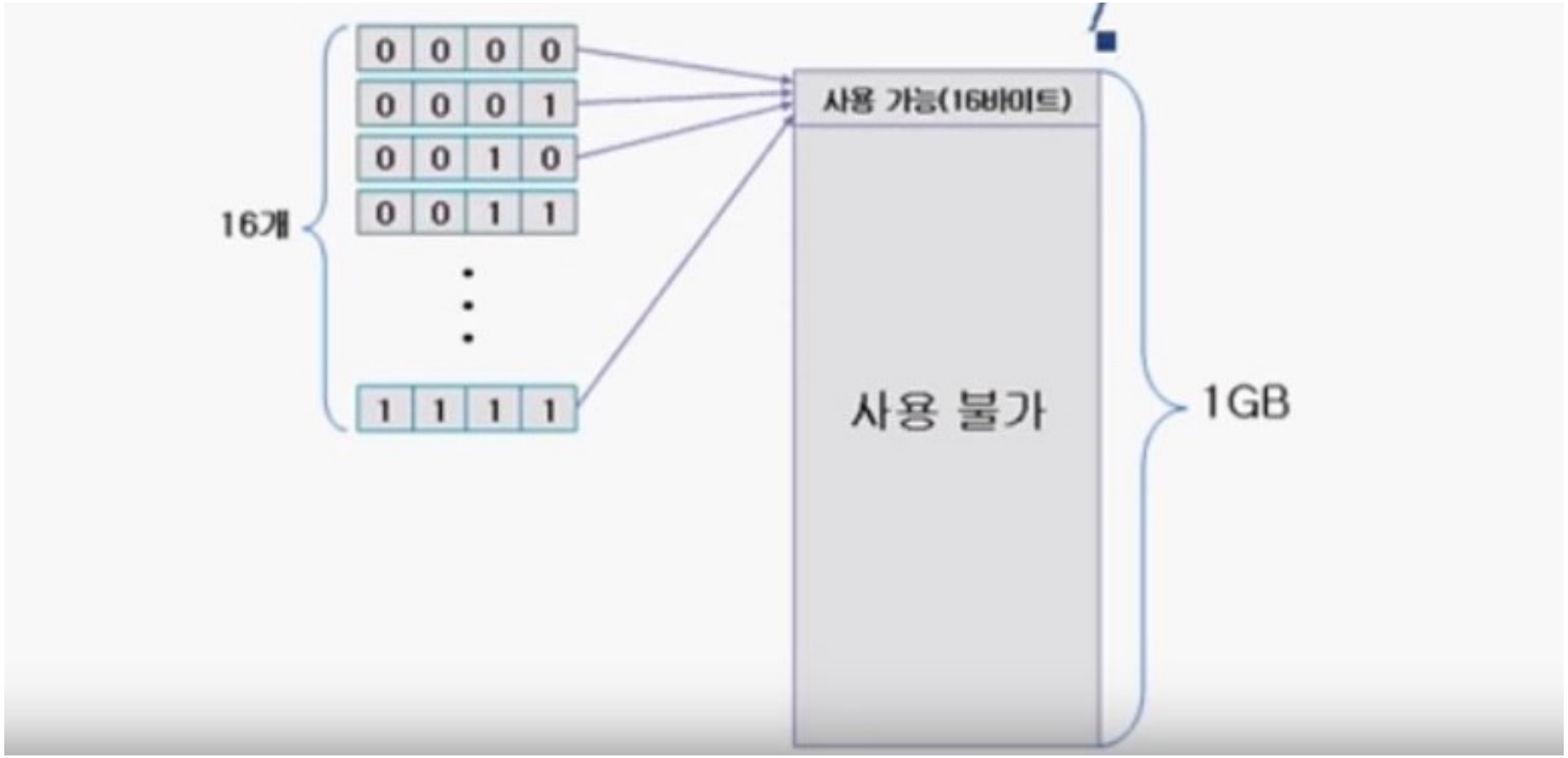
MAC 유저의
64비트 기반 프로그래밍

화생방 스터디
이원영

목차

01. 64비트와 32비트 컴퓨터



01. 64비트와 32비트 컴퓨터

```c
#include <stdio.h>

int main() {
    int num = 10;
    printf("변수 num의 값: %d\n", num);
    printf("변수 num의 주소값: %p\n", (void*)&num);
    return 0;
}
```

C/C++에서 주소값을 출력하면 몇 진수로 출력?

01. 64비트와 32비트 컴퓨터

```c
#include <stdio.h>

int main() {
    int num = 10;
    printf("변수 num의 값: %d\n", num);
    printf("변수 num의 주소값: %p\n", (void*)&num);
    return 0;
}
```

A

A의 출력값이 [0xA]라면 최소 몇 비트 컴퓨터?

01. 64비트와 32비트 컴퓨터

```c
#include <stdio.h>

int main() {
    int num = 10;
    printf("변수 num의 값: %d\n", num);
    printf("변수 num의 주소값: %p\n", (void*)&num);
    return 0;
}
```

A

A의 출력값이 [0x200]라면 최소 몇 비트 컴퓨터?

01. 64비트와 32비트 컴퓨터

```c
#include <stdio.h>

int main() {
    int num = 10;
    printf("변수 num의 값: %d\n", num);
    printf("변수 num의 주소값: %p\n", (void*)&num);
    return 0;
}
```

A

A의 출력값이 [0x7FF]라면 최소 몇 비트 컴퓨터?

02. Polymorphic 자료형

Mac OS에서 C++ 파일 실행하기

```
# 만약 컴파일러가 설치 안되어 있다면
$ xcode-select --install

# example.cpp 파일 컴파일 실행 (compliedExample 이름으로 실행 파일 생성)
$ clang++ -std=c++17 -o compliedExample example.cpp

# 실행파일 실행
$ ./compliedExample
```

$ xcode-select --install
$ clang++ -std=c++17 -o compliedExample example.cpp
$ ./compliedExample

- 만약 Mac OS에서 텍스트 편집기로 .cpp 파일을 작성하였다면, 위와 같이 실행
- 물론 Visual Studio 설치하면 제일 좋음
- 예제 실행 말고는 C/C++을 사용하지 않는다면 텍스트 편집기로도 충분!

02. Polymorphic 자료형- Mac OS 예제

```c
#include <stdio.h>
#include <tchar.h>
#include <windows.h>

UINT CalDistance (UINT a, UINT b) {
    return a-b;
}

int _tmain(void) {
    INT val1 = 10;
    INT val2 = 20;

    _tprintf(_T("Position %u, %u \n"), (UINT)&val1, (U
    _tprintf(
        _T("distance : %u \n"),
        CalDistance((UINT)&val1, (UNIT)&val2)
    );

    return 0;
}
```

```c
#include <stdio.h>

uintptr_t CalDistance(int val1, int val2) {
    return val1 - val2;
}

int main(void) {
    uintptr_t val1 = 10;
    uintptr_t val2 = 20;

    printf("Position %p, %p \n", &val1, &val2);
    printf("distance : %lu \n", CalDistance((uintptr_t) &val1, (uintptr_t) &val2));

    return 0;
}
```

UINT_PTR == uintptr_t

포인터 값을 저장하는 데 사용할 수 있으며 표준 데이터 형식

이므로 다른 코드와 호환성이 높음

02. Polymorphic 자료형- Mac OS 예제

```c
#include <stdio.h>

uintptr_t CalDistance(int val1, int val2) {
    return val1 - val2;
}

int main(void) {
    int32_t val1 = 10;
    int32_t val2 = 20;

    printf("Position %p, %p \n", &val1, &val2);
    printf("distance : %lu \n", CalDistance((uintptr

    return 0;
}
```

```c
#include <stdio.h>

uintptr_t CalDistance(int val1, int val2) {
    return val1 - val2;
}

int main(void) {
    int64_t val1 = 10;
    int64_t val2 = 20;

    printf("Position %p, %p \n", &val1, &val2)
    printf("distance : %lu \n", CalDistance((u

    return 0;
}
```

02. Polymorphic 자료형- Mac OS 예제

```c
#include <stdio.h>

uintptr_t CalDistance(int val1, int val2) {
    return val1 - val2;
}

int main(void) {
    int32_t val1 = 10;
    int32_t val2 = 20;

    printf("Position %p, %p \n", &val1, &val2);
    printf("distance : %lu \n", CalDistance((uintptr

    return 0;
}
```

```
➜  system-programing ./polymor
Position 0x16db63188, 0x16db63184
distance : 4
```

```c
#include <stdio.h>

uintptr_t CalDistance(int val1, int val2) {
    return val1 - val2;
}

int main(void) {
    int64_t val1 = 10;
    int64_t val2 = 20;

    printf("Position %p, %p \n", &val1, &val2)
    printf("distance : %lu \n", CalDistance((u

    return 0;
}
```

```
➜  system-programing ./polymor
Position 0x16b493180, 0x16b493178
distance : 8
```

02. Polymorphic 자료형- Mac OS 예제

```c
#include <stdio.h>

uintptr_t CalDistance(int val1, int val2) {
    return val1 - val2;
}

int main(void) {
    uintptr_t val1 = 10;
    uintptr_t val2 = 20;

    printf("Position %p, %p \n", &val1, &val2);
    printf("distance : %lu \n", CalDistance((uintptr_t) &val1, (uintptr_t) &val2));

    return 0;
}
```

출력값은?

02. Polymorphic 자료형- Mac OS 예제

```c
#include <stdio.h>

uintptr_t CalDistance(int val1, int val2) {
    return val1 - val2;
}

int main(void) {
    uintptr_t val1 = 10;
    uintptr_t val2 = 20;

    printf("Position %p, %p \n", &val1, &val2);
    printf("distance : %lu \n", CalDistance((uintptr_t) &val1, (uintptr_t) &val2));

    return 0;
}
```

```
● ➜  system-programing ./polymor
Position 0x16b493180, 0x16b493178
distance : 8
```

03. 오류의 확인

```c
#include <stdio.h>
#include <tchar.h>
#include <windows.h>

int _tmain(void) {
HANDLE hFile = CreateFile (
_T("ABC.DAT"), GENERIC_READ, FILE_SHARE_READ,
NULL, OPEN_EXISTINGS, FILE_ATTRIBUTE_NORMAL,
NULL
);

if (hFile == INVALID_HANDLE_VALUE) {
    _tprint( _T("error code: %d \n"), GetLastErro
    return 0;
}
return 0;
}
```

```
[실행 결과]
error code: 2
```

```c
#include <stdio.h>
#include <errno.h>
#include <string.h>

int main() {
    FILE *fp;
    fp = fopen("nonexistentfile.txt", "r");

    if (fp == NULL) {
        printf("error code: %d\n", errno);
        printf("error message: %s\n", strerror(errno));
    }

    return 0;
}
```

03. 오류의 확인

```c
#include <stdio.h>
#include <tchar.h>
#include <windows.h>

int _tmain(void) {
HANDLE hFile = CreateFile (
_T("ABC.DAT"), GENERIC_READ, FILE_SHARE_READ,
NULL, OPEN_EXISTINGS, FILE_ATTRIBUTE_NORMAL,
NULL
);

if (hFile == INVALID_HANDLE_VALUE) {
    _tprint( _T("error code: %d \n"), GetLastErro
    return 0;
}
return 0;
}
```

[실행 결과]

```
error code: 2
```

```c
#include <stdio.h>
#include <errno.h>
#include <string.h>

int main() {
    FILE *fp;
    fp = fopen("nonexistentfile.txt", "r");

    if (fp == NULL) {
        printf("error code: %d\n", errno);
        printf("error message: %s\n", strerror(errno));
    }

    return 0;
}
```

```
→  system-programing ./getLastError
error code: 2
error message: No such file or directory
```

04. 리눅스 프로세서 아키텍처

04. 리눅스 프로세서 아키텍처

04. 리눅스 프로세서 아키텍처

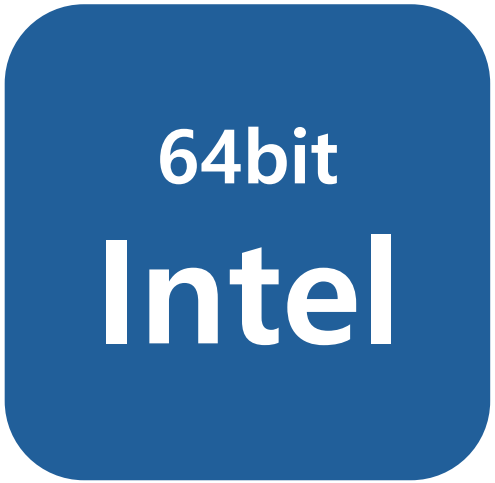
IA-32
(또는 x32)
x86, i386, i686

IA: Intel Architecture-32라는 뜻

- 아키텍처는 같지만 CPU에 따라 다르게 표현
- 예를 들어, 'i386'은 인텔 80386 호환을,
'i686'은 인텔 펜티엄 2 이상 호환을 의미

04. 리눅스 프로세서 아키텍처

IA-32
(또는 **x32**)
x86, i386, i686



IA-64
(또는 **x32**)

04. 리눅스 프로세서 아키텍처

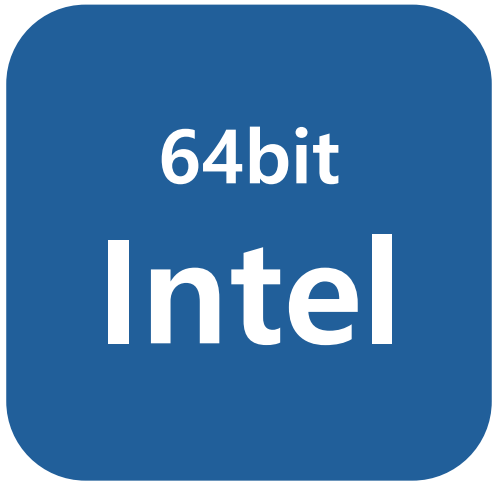
IA-32
(또는 **x32**)
x86, i386, i686

IA-64
(또는 **x32**)

IA-32랑 호환 앙대안대

04. 리눅스 프로세서 아키텍처

IA-32
(또는 x32)

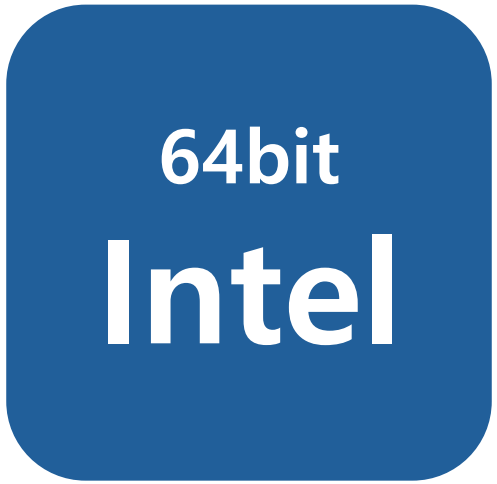


IA-64
(또는 x32)

AMD64

04. 리눅스 프로세서 아키텍처

IA-32
(또는 **x32**)



IA-64
(또는 **x32**)

AMD64

- IA-32랑 미친 호환성
- AMD64는 AMD CPU 전용x

04. 리눅스 프로세서 아키텍처

IA-32
(또는 x32)
x86, i386, i686

IA-64
(또는 x32)

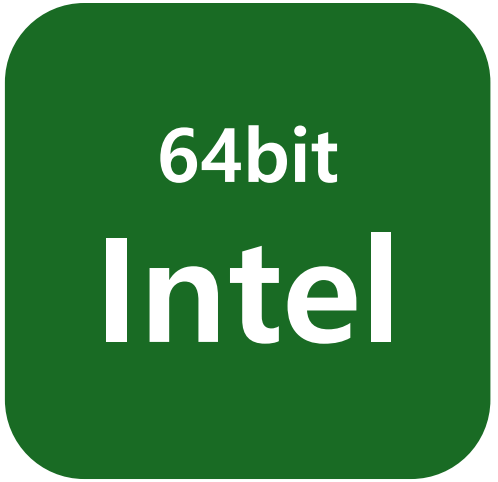
IA-32e
EM64T
Intel64

04. 리눅스 프로세서 아키텍처

IA-32
(또는 **x32**)
x86, i386, i686

IA-64
(또는 **x32**)

AMD64



IA-32e
EM64T
Intel64

- AMD64 라이선스 활용
- IA-32 호환

04. 리눅스 프로세서 아키텍처

ARMv7

04. 리눅스 프로세서 아키텍처

- 저전력 중심
- 주로 모바일 기기

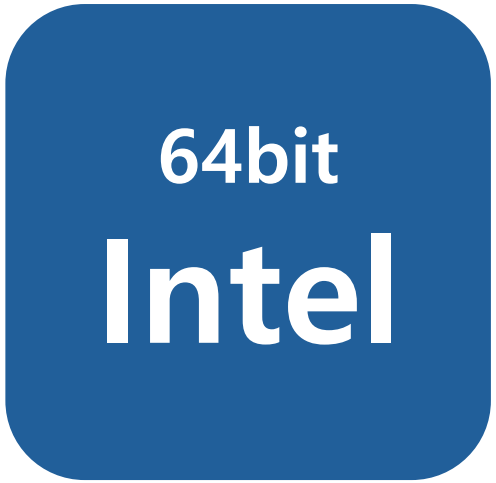
ARMv7

04. 리눅스 프로세서 아키텍처

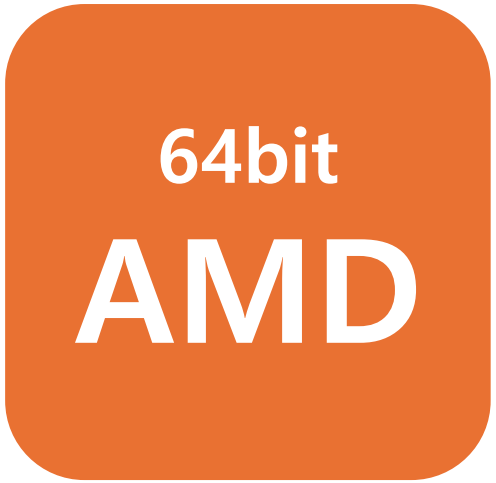
ARMv7

ARMv8
ARM64
Aarch64

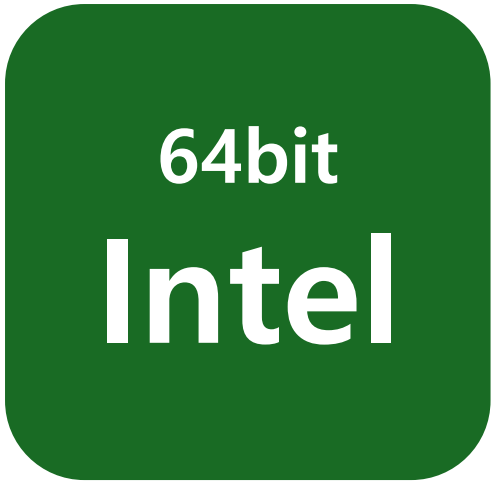
04. 리눅스 프로세서 아키텍처



IA-64
(또는 x32)



AMD64



IA-32e
EM64T
Intel64

ARMv8
ARM64
Aarch64

M1은 어떤 프로세서 기반?

04. 리눅스 프로세서 아키텍처

04. 리눅스 프로세서 아키텍처

전력 스펙 수정 & Apple SW 통합 개발

= 앱등이 무한생성

Q & A