ARM64 Assembly 기초

https://youtu.be/mVoXiiM1GT4

HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Introduce ARMv8

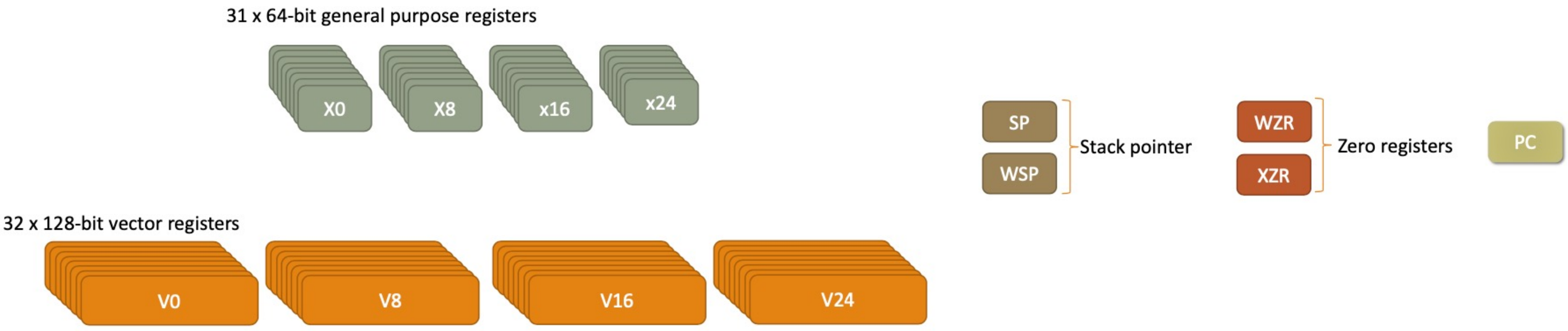
- 2011년에 발표된 ARM의 64-bit 아키텍처

- AArch64 혹은 ARM64은 64-bit 명령어셋 아키텍처이다.

- ARMv8은 AArch64를 지원하며, AArch32의 호환성을 제공한다.

ARMv8 Register

- 31개의 범용 64-bit 레지스터를 제공하며, 32개의 128-bit 벡터 레지스터를 제공한다.


31 x 64-bit general purpose registers
X0
X8
x16
x24
SP
WSP
Stack pointer
WZR
XZR
Zero registers
PC
32 x 128-bit vector registers
V0
V8
V16
V24

Vector Register

- 16B / 8B (Byte, 8-bit)

8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
								8	8	8	8	8	8	8	8

- 8H / 4H (Half Word, 16-bit)

16	16	16	16	16	16	16	16
				16	16	16	16

- 4S / 2S (Word, 32-bit)

32	32	32	32
		32	32

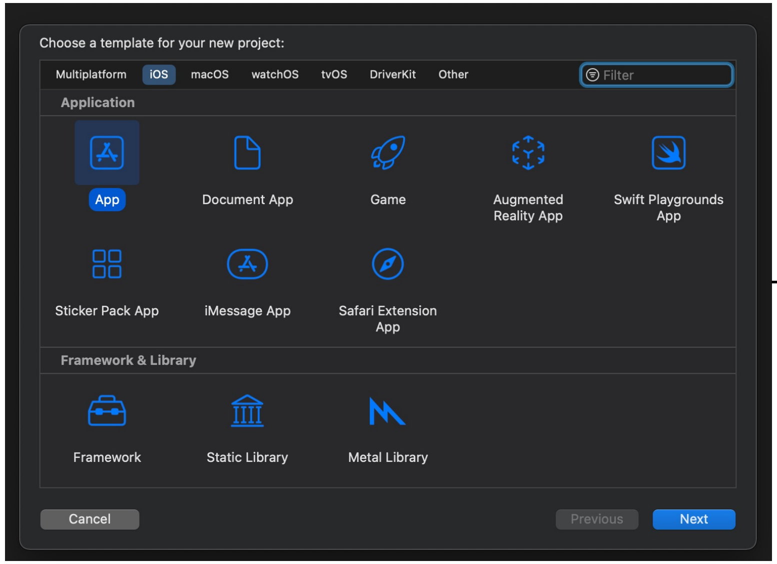
- 2D (Double Word, 64-bit)

64	64

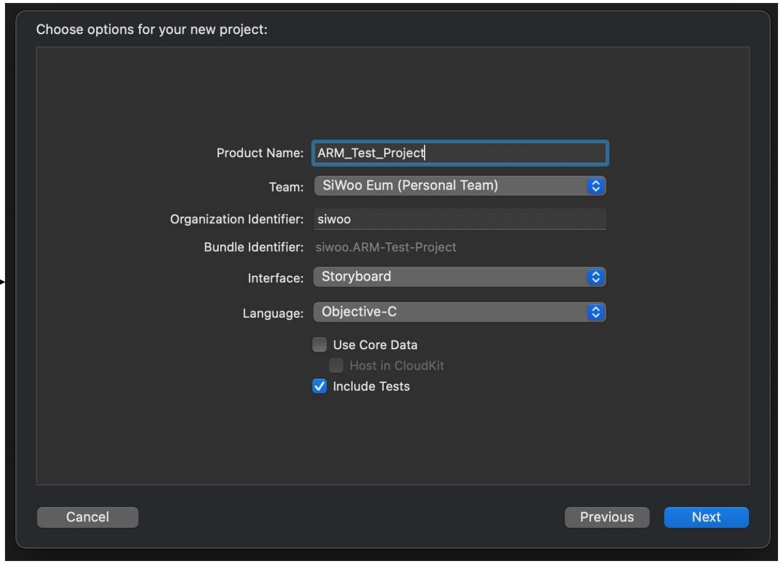
Programming Example

- Xcode 상에서 프로그래밍 가능

New Project -> IOS -> APP



Team 설정 -> Objective-C(Language)



Choose options for your new project:

Product Name: ARM_Test_Project

Team: SiWoo Eum (Personal Team)

Organization Identifier: siwoo

Bundle Identifier: siwoo.ARM-Test-Project

Interface: Storyboard

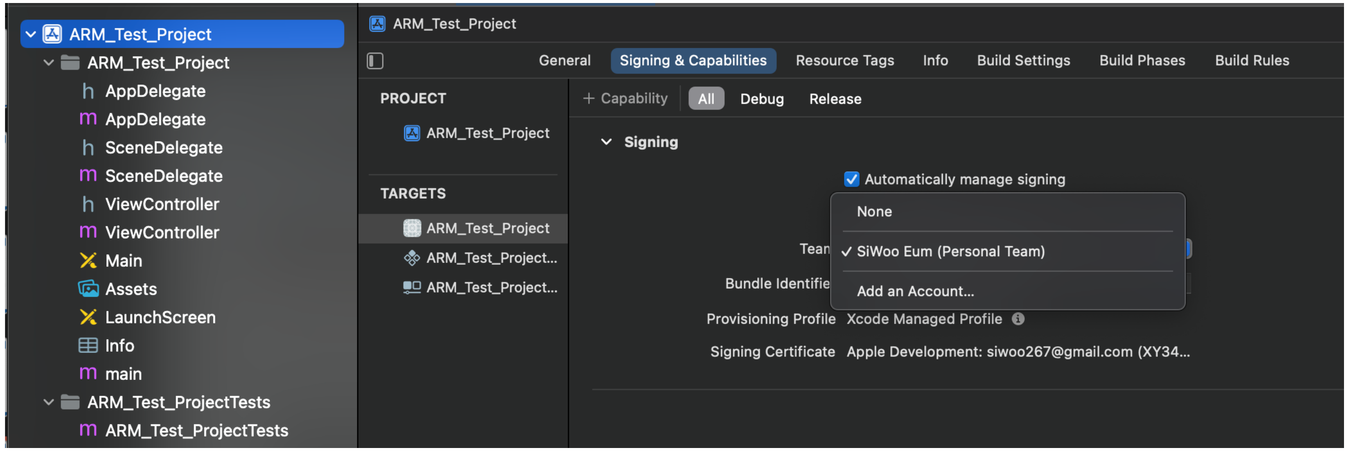
Language: Objective-C

□ Use Core Data

□ Host in CloudKit

☑ Include Tests

Programming Example



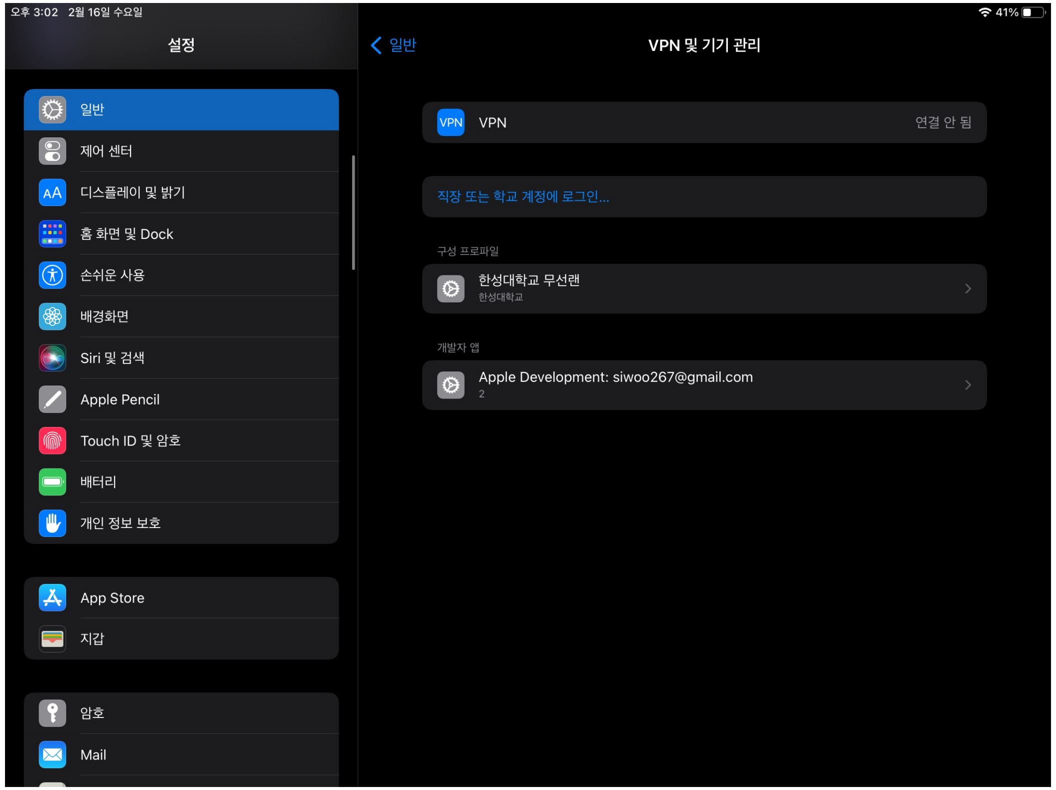
☑ Automatically manage signing

Team ✓ SiWoo Eum (Personal Team)

Bundle Identifie

Provisioning Profile Xcode Managed Profile

Signing Certificate Apple Development: siwoo267@gmail.com (XY34...



VPN 연결 안 됨

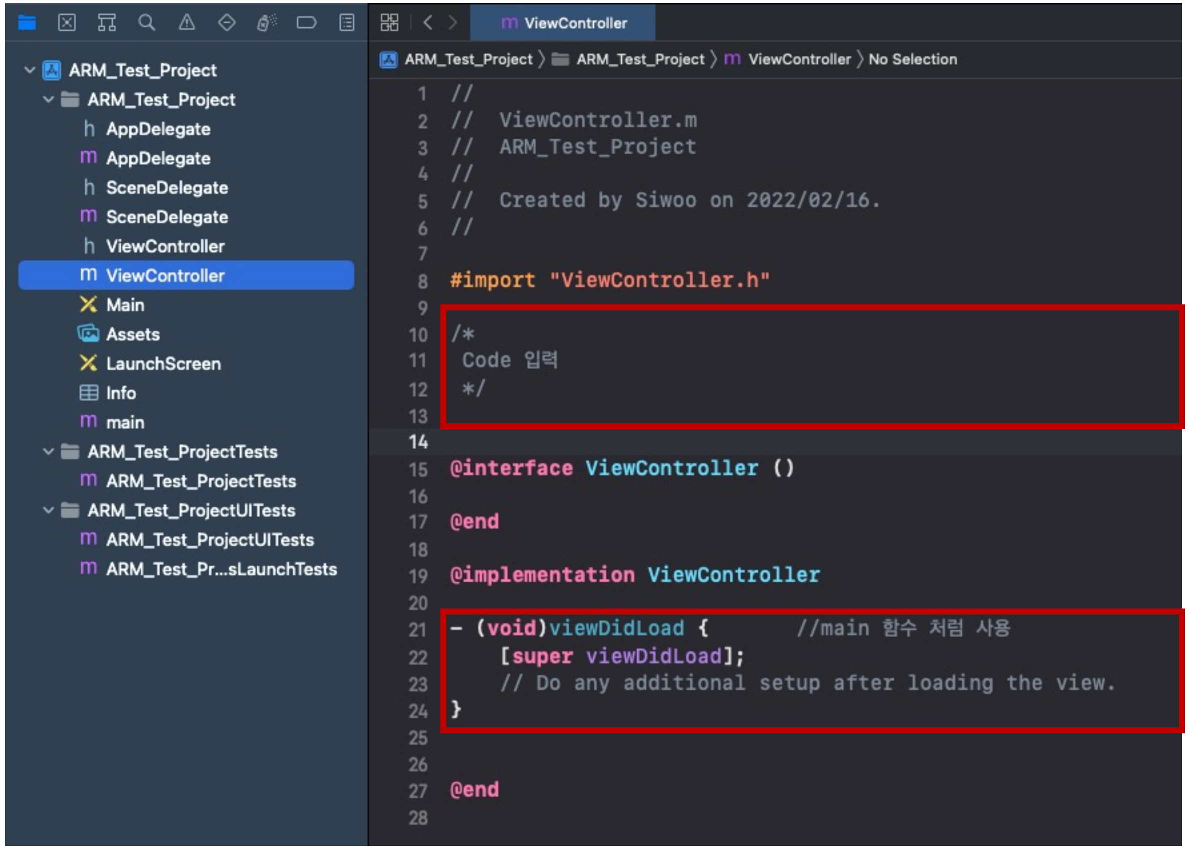
구성 프로파일

한성대학교 무선랜
한성대학교

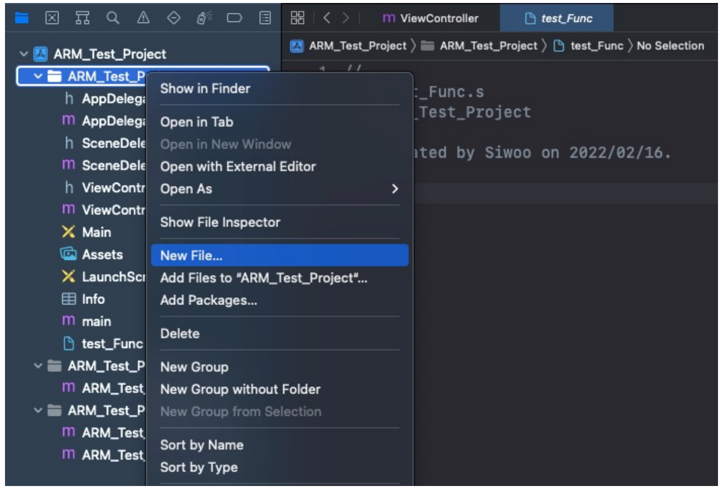
개발자 앱

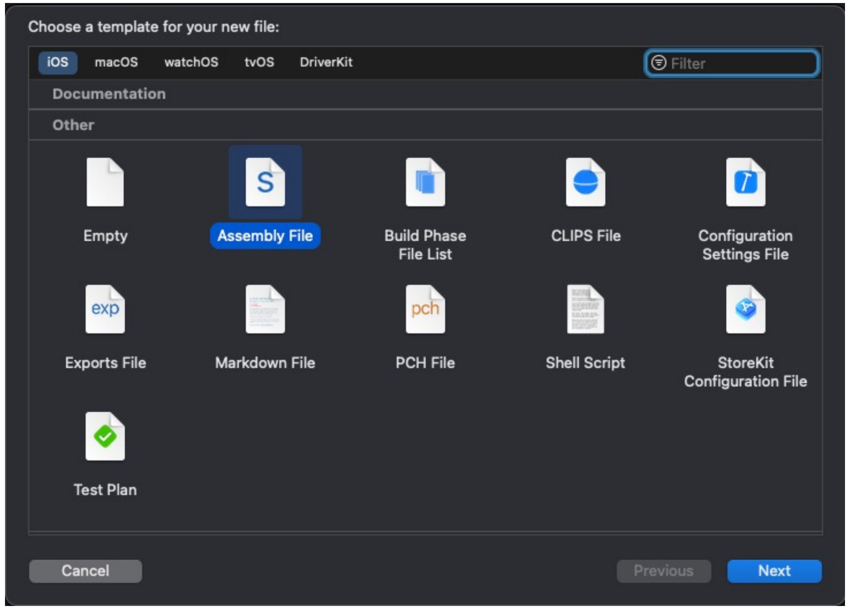
Apple Development: siwoo267@gmail.com
2

Programming Example

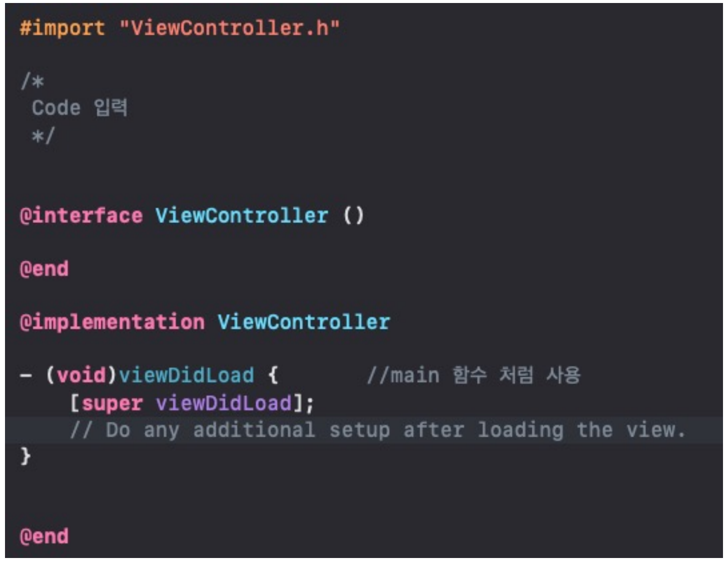


```
//
//  ViewController.m
//  ARM_Test_Project
//
//  Created by Siwoo on 2022/02/16.
//

#import "ViewController.h"

/*
 Code 입력
 */

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {        //main 함수 처럼 사용
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}


@end
```

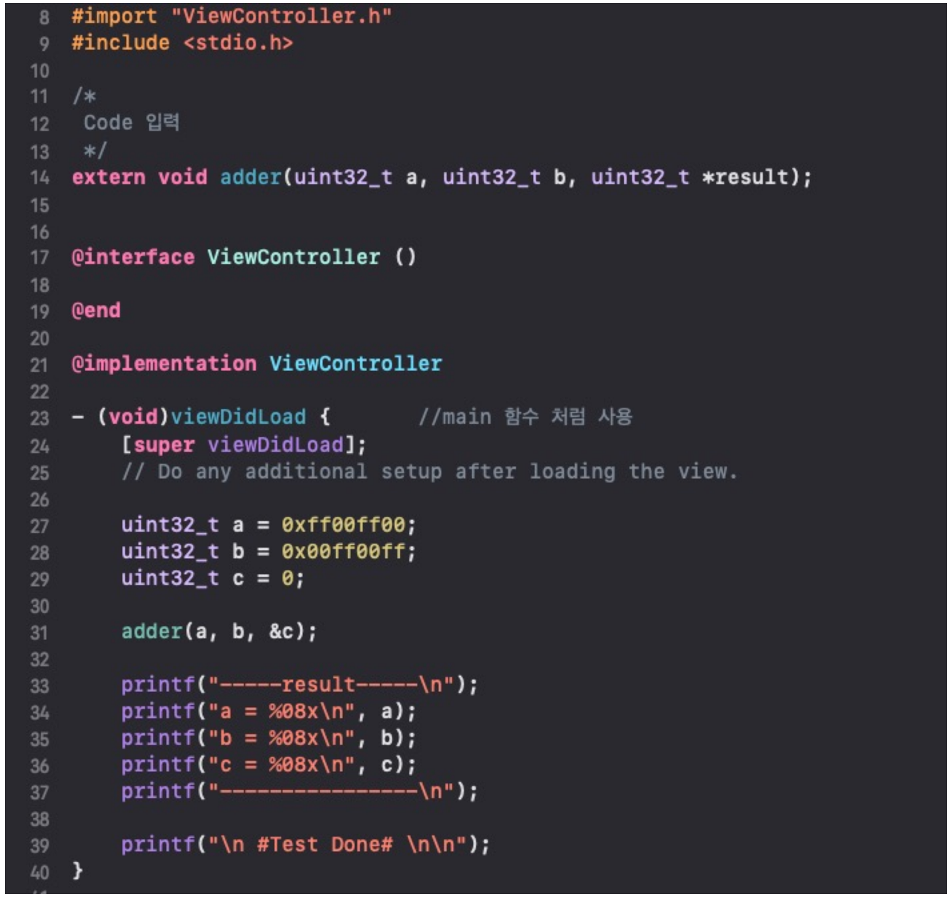




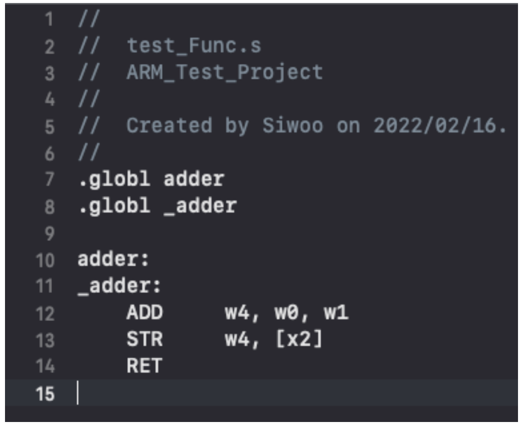
Programming Example (Scalar Register)



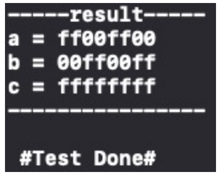
```
#import "ViewController.h"

/*
 Code 입력
 */


@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {        //main 함수 처럼 사용
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}


@end
```



```
#import "ViewController.h"
#include <stdio.h>

/*
 Code 입력
 */
extern void adder(uint32_t a, uint32_t b, uint32_t *result);


@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {        //main 함수 처럼 사용
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    uint32_t a = 0xff00ff00;
    uint32_t b = 0x00ff00ff;
    uint32_t c = 0;

    adder(a, b, &c);

    printf("-----result-----\n");
    printf("a = %08x\n", a);
    printf("b = %08x\n", b);
    printf("c = %08x\n", c);
    printf("----------------\n");

    printf("\n #Test Done# \n\n");
}
```

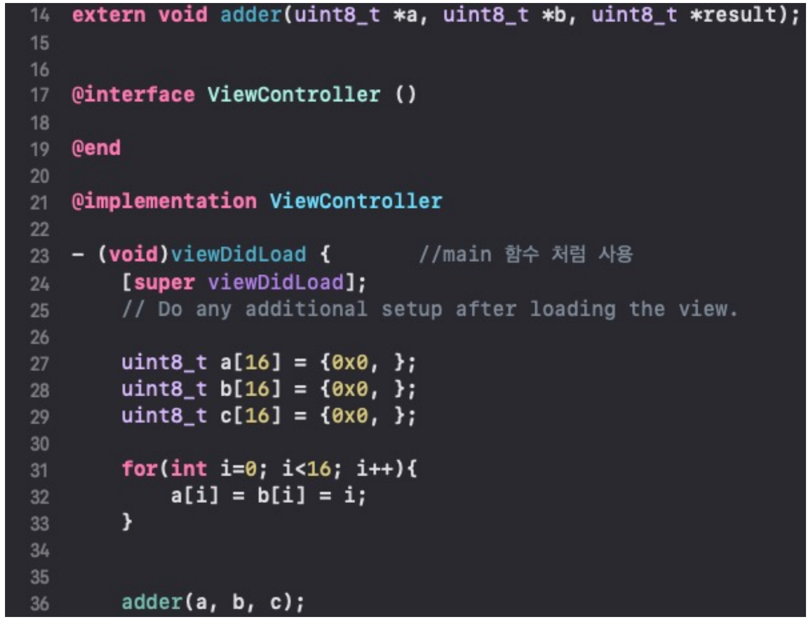


```
//
//  test_Func.s
//  ARM_Test_Project
//
//  Created by Siwoo on 2022/02/16.
//
.globl adder
.globl _adder

adder:
_adder:
    ADD     w4, w0, w1
    STR     w4, [x2]
    RET

```

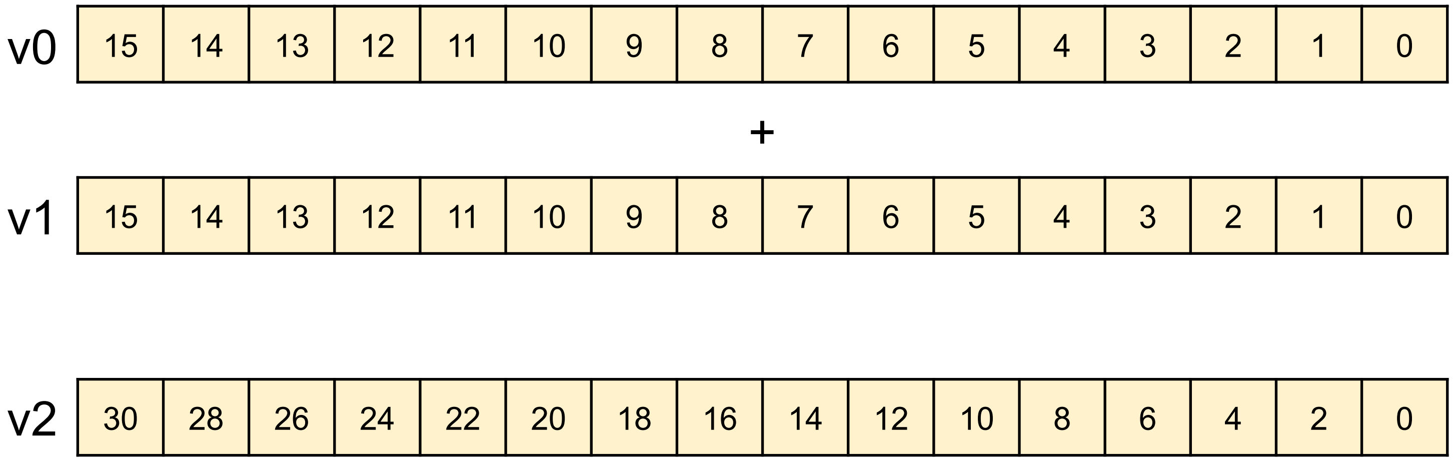


```
-----result-----
a = ff00ff00
b = 00ff00ff
c = ffffffff
----------------

 #Test Done#
```

Programming Example (Vector Register)



```
extern void adder(uint8_t *a, uint8_t *b, uint8_t *result);


@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {        //main 함수 처럼 사용
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    uint8_t a[16] = {0x0, };
    uint8_t b[16] = {0x0, };
    uint8_t c[16] = {0x0, };

    for(int i=0; i<16; i++){
        a[i] = b[i] = i;
    }


    adder(a, b, c);
```

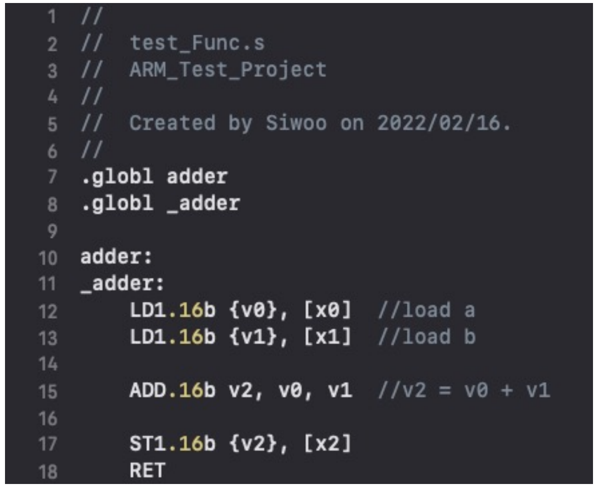


v0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0

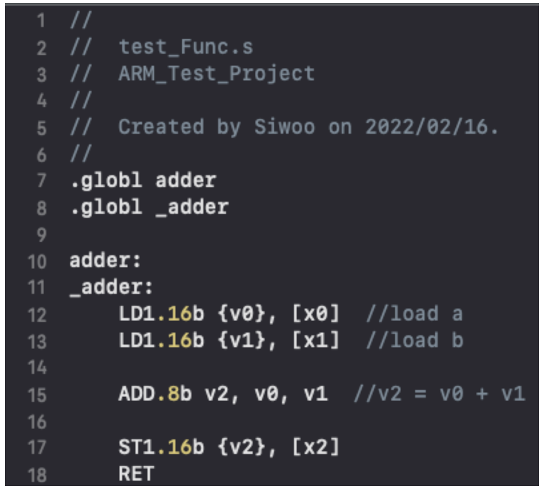
+

v1 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0

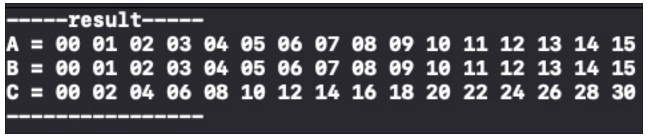
v2 | 30 | 28 | 26 | 24 | 22 | 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 | 0



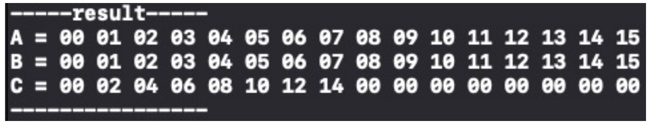
```
//
//  test_Func.s
//  ARM_Test_Project
//
//  Created by Siwoo on 2022/02/16.
//
.globl adder
.globl _adder

adder:
_adder:
    LD1.16b {v0}, [x0]  //load a
    LD1.16b {v1}, [x1]  //load b

    ADD.16b v2, v0, v1  //v2 = v0 + v1

    ST1.16b {v2}, [x2]
    RET
```



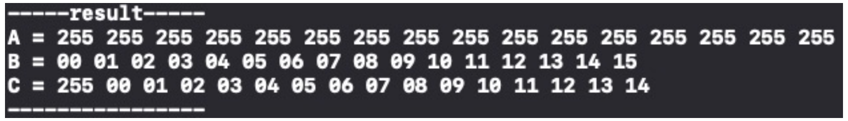
```
//
//  test_Func.s
//  ARM_Test_Project
//
//  Created by Siwoo on 2022/02/16.
//
.globl adder
.globl _adder

adder:
_adder:
    LD1.16b {v0}, [x0]  //load a
    LD1.16b {v1}, [x1]  //load b

    ADD.8b v2, v0, v1  //v2 = v0 + v1

    ST1.16b {v2}, [x2]
    RET
```



```
-----result-----
A = 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
B = 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
C = 00 02 04 06 08 10 12 14 16 18 20 22 24 26 28 30
----------------
```

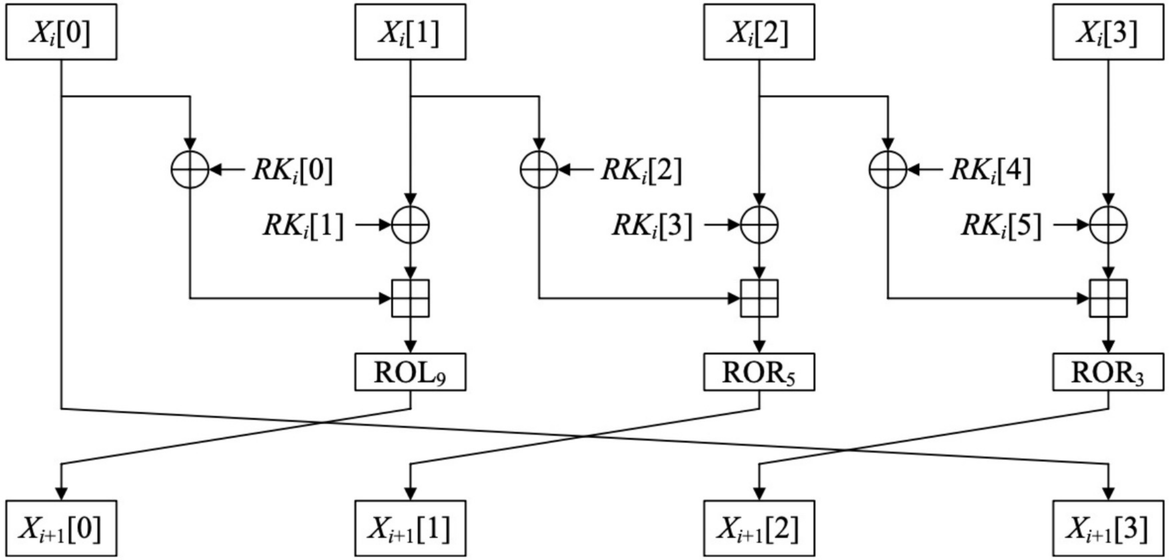


```
-----result-----
A = 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
B = 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
C = 00 02 04 06 08 10 12 14 00 00 00 00 00 00 00 00
----------------
```

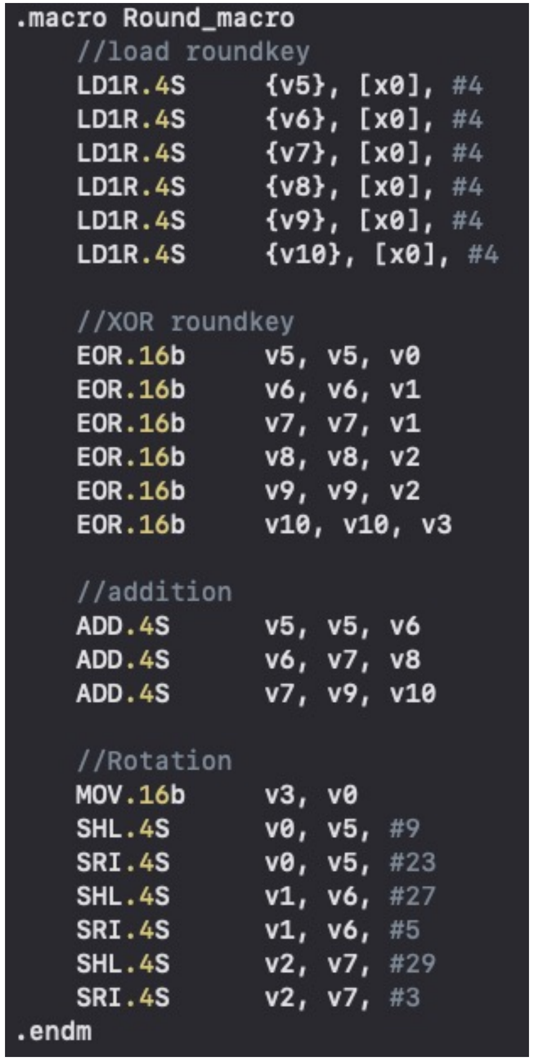


```
-----result-----
A = 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
B = 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
C = 255 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14
----------------
```

LEA 구현 실습



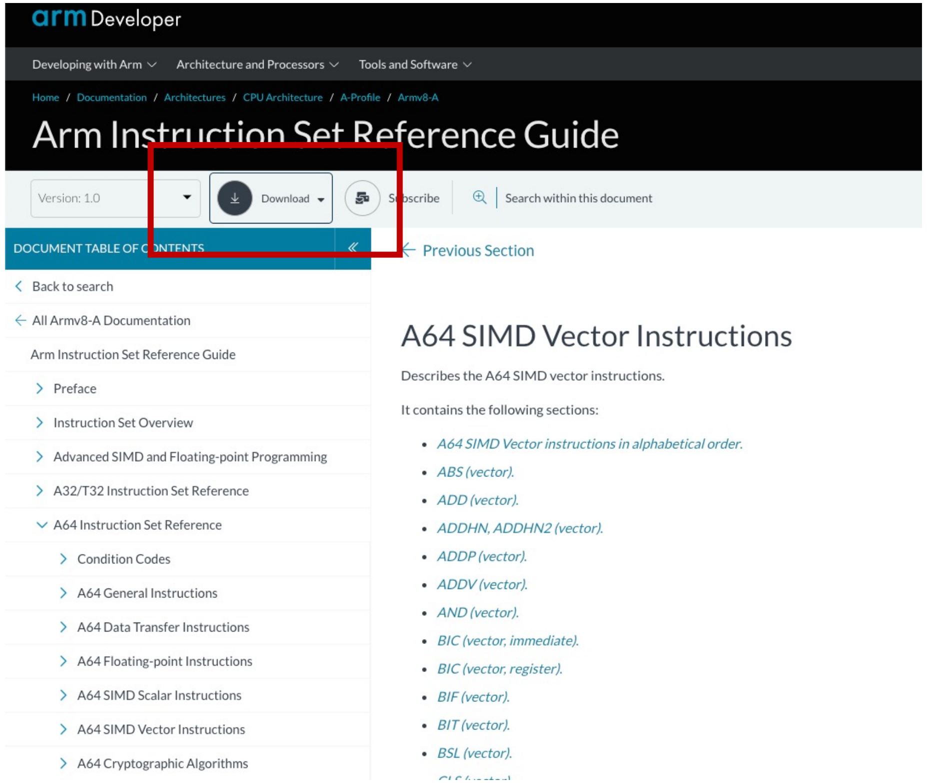
X[0]	X[0]	X[0]	X[0]
X[1]	X[1]	X[1]	X[1]
X[2]	X[2]	X[2]	X[2]
X[3]	X[3]	X[3]	X[3]



```
.macro Round_macro
    //load roundkey
    LD1R.4S       {v5}, [x0], #4
    LD1R.4S       {v6}, [x0], #4
    LD1R.4S       {v7}, [x0], #4
    LD1R.4S       {v8}, [x0], #4
    LD1R.4S       {v9}, [x0], #4
    LD1R.4S       {v10}, [x0], #4

    //XOR roundkey
    EOR.16b       v5, v5, v0
    EOR.16b       v6, v6, v1
    EOR.16b       v7, v7, v1
    EOR.16b       v8, v8, v2
    EOR.16b       v9, v9, v2
    EOR.16b       v10, v10, v3

    //addition
    ADD.4S        v5, v5, v6
    ADD.4S        v6, v7, v8
    ADD.4S        v7, v9, v10

    //Rotation
    MOV.16b       v3, v0
    SHL.4S        v0, v5, #9
    SRI.4S        v0, v5, #23
    SHL.4S        v1, v6, #27
    SRI.4S        v1, v6, #5
    SHL.4S        v2, v7, #29
    SRI.4S        v2, v7, #3
.endm
```

Arm Instruction Set Reference Guide

- Arm에서 사용하는 명령어 설명서
  - https://developer.arm.com/documentation/100076/0100/a64-instruction-set-reference/a64-simd-vector-instructions


arm Developer
Developing with Arm
Architecture and Processors
Tools and Software
Home / Documentation / Architectures / CPU Architecture / A-Profile / Armv8-A
Version: 1.0
Download
S bscribe
Search within this document
DOCUMENT TABLE OF C NTENTS
Previous Section

Arm Instruction Set Reference Guide

- Back to search
- All Armv8-A Documentation
- Arm Instruction Set Reference Guide
  - Preface
  - Instruction Set Overview
  - Advanced SIMD and Floating-point Programming
  - A32/T32 Instruction Set Reference
  - A64 Instruction Set Reference
    - Condition Codes
    - A64 General Instructions
    - A64 Data Transfer Instructions
    - A64 Floating-point Instructions
    - A64 SIMD Scalar Instructions
    - A64 SIMD Vector Instructions
    - A64 Cryptographic Algorithms

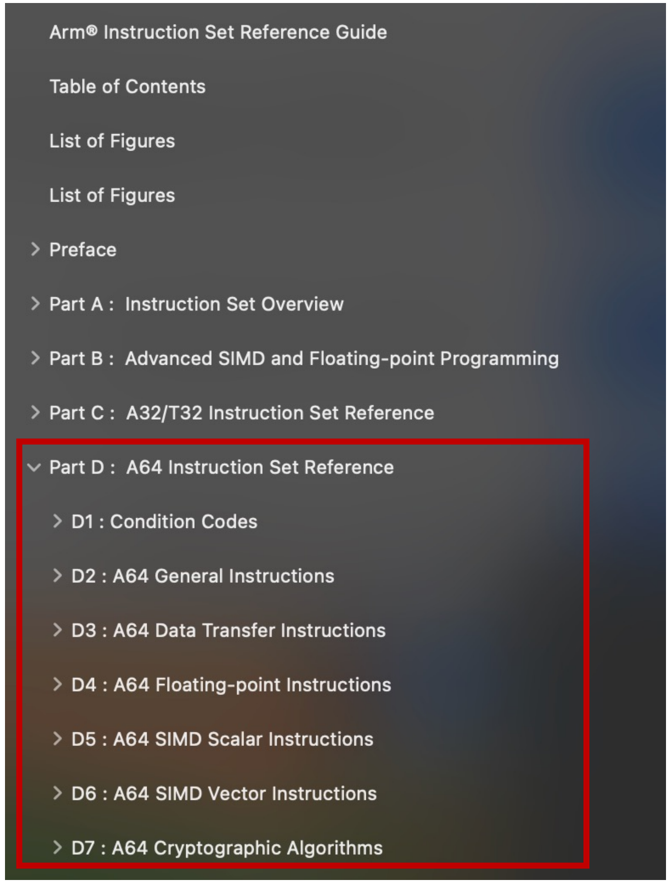
A64 SIMD Vector Instructions

Describes the A64 SIMD vector instructions.

It contains the following sections:

- *A64 SIMD Vector instructions in alphabetical order.*
- *ABS (vector).*
- *ADD (vector).*
- *ADDHN, ADDHN2 (vector).*
- *ADDP (vector).*
- *ADDV (vector).*
- *AND (vector).*
- *BIC (vector, immediate).*
- *BIC (vector, register).*
- *BIF (vector).*
- *BIT (vector).*
- *BSL (vector).*

Arm Instruction Set Reference Guide

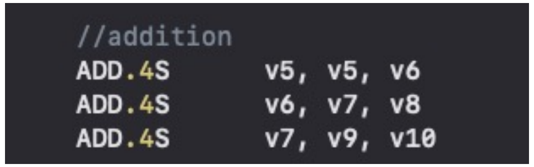


ADD (vector)

Add (vector).

Syntax

```
ADD Vd.T, Vn.T, Vm.T
```



```
//addition
ADD.4S        v5, v5, v6
ADD.4S        v6, v7, v8
ADD.4S        v7, v9, v10
```

Where:

Vd

Is the name of the SIMD and FP destination register.

T

Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn

Is the name of the first SIMD and FP source register.

Vm

Is the name of the second SIMD and FP source register.

Usage

Add (vector). This instruction adds corresponding elements in the two source SIMD and FP registers, places the results into a vector, and writes the vector to the destination SIMD and FP register.

Depending on the settings in the CPACR_EL1, CPTR_EL2, and CPTR_EL3 registers, and the current Security state and Exception level, an attempt to execute the instruction might be trapped.

감사합니다