Arm Processor Families

- **Cortex-A series**
  - High-performance processors for open OSs
  - Applications
    - Smartphones, digital TV, server solutions, home gateways
- **Cortex-R series**
  - Exceptional performance for real-time applications
  - Applications
    - Automotive braking systems and powertrains
- **Cortex-M series**
  - Cost-sensitive solutions for deterministic microcontroller applications
  - Applications
    - Microcontrollers, mixed signal devices, smart sensors, automotive body electronics, airbags

🟥	Cortex-A75	Cortex-A55	Cortex-A
🟥	Cortex-A73	Cortex-A53	
🟥	Cortex-A72	Cortex-A35	
🟥	Cortex-A57	Cortex-A32	
🟥	Cortex-A17	Cortex-A8	
🟥	Cortex-A15	Cortex-A7	
🟥	Cortex-A9	Cortex-A5	
🟨	Cortex-R8	Cortex-R52	Cortex-R
🟨	Cortex-R7	Cortex-R5	
🟨		Cortex-R4	
🟩	Cortex-M7	Cortex-M0	Cortex-M
🟩	Cortex-M4	Cortex-M23	
🟩	Cortex-M3	Cortex-M33	
🟩	Cortex-M1		
🟩	Cortex-M0+		
🟦	SC000		SecurCore
🟦	SC100		
🟦	SC300		
🟫	Arm11		Classic
🟫	Arm9		
🟫	Arm7		

As of Nov 2017

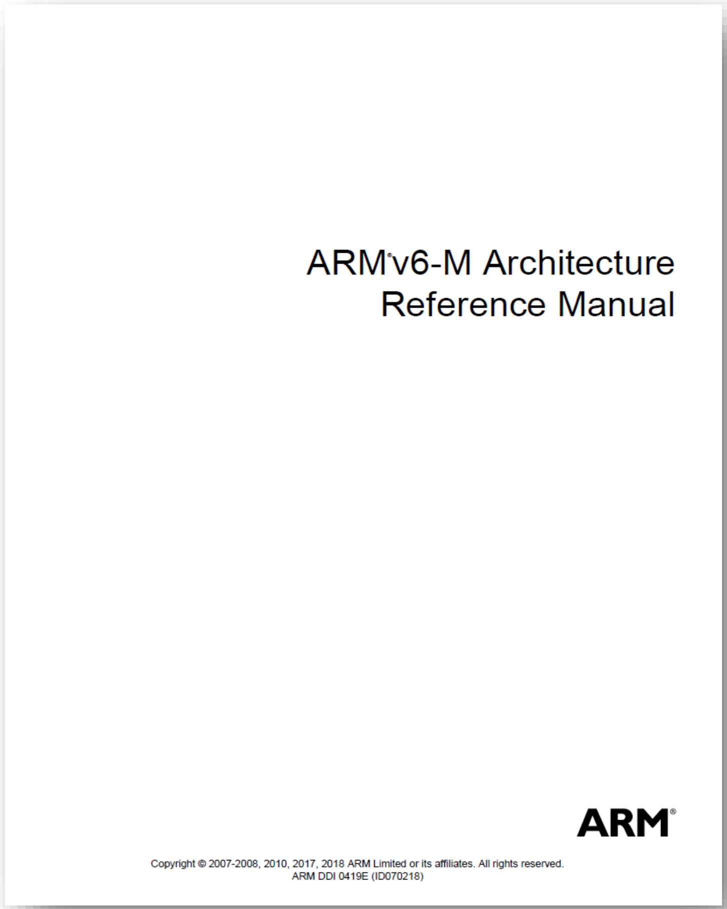
Arm Processor Families

- **SecurCore series**
  - High-security applications such as smartcards and e-government
- **Classic processors**
  - Include Arm7, Arm9, and Arm11 families

🟥	Cortex-A75	Cortex-A55	Cortex-A
🟥	Cortex-A73	Cortex-A53	
🟥	Cortex-A72	Cortex-A35	
🟥	Cortex-A57	Cortex-A32	
🟥	Cortex-A17	Cortex-A8	
🟥	Cortex-A15	Cortex-A7	
🟥	Cortex-A9	Cortex-A5	
🟨	Cortex-R8	Cortex-R52	Cortex-R
🟨	Cortex-R7	Cortex-R5	
🟨		Cortex-R4	
🟩	Cortex-M7	Cortex-M0	Cortex-M
🟩	Cortex-M4	Cortex-M23	
🟩	Cortex-M3	Cortex-M33	
🟩	Cortex-M1		
🟩	Cortex-M0+		
🟦	SC000		SecurCore
🟦	SC100		
🟦	SC300		
🟫	Arm11		Classic
🟫	Arm9		
🟫	Arm7		

As of Nov 2017

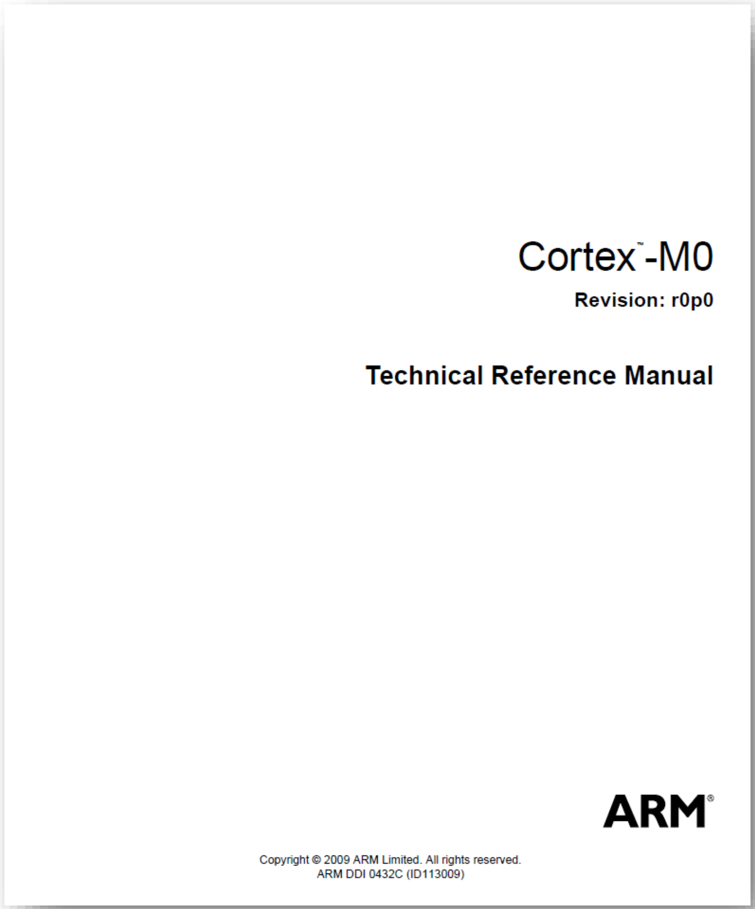
Cortex-M0 Overview



ARM®v6-M Architecture
Reference Manual

ARM®

Copyright © 2007-2008, 2010, 2017, 2018 ARM Limited or its affiliates. All rights reserved.
ARM DDI 0419E (ID070218)



Cortex™-M0

Revision: r0p0

Technical Reference Manual

ARM®

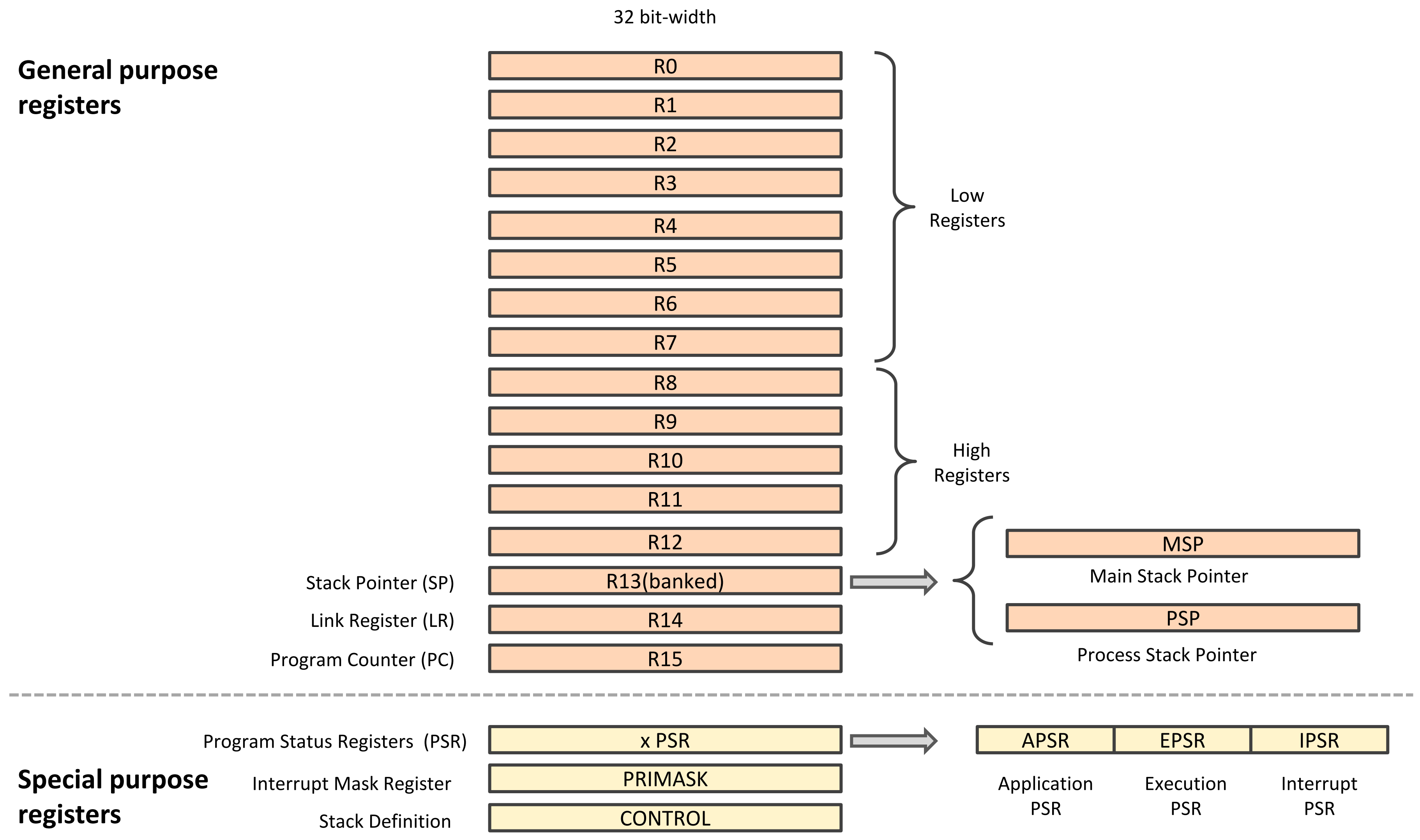
Copyright © 2009 ARM Limited. All rights reserved.
ARM DDI 0432C (ID113009)

ARMv6M ARM: https://developer.arm.com/documentation/ddi0419
Cortex-M0 TRM: https://developer.arm.com/documentation/ddi0432

Cortex-M0 Overview

- **Cortex-M0**
  - 32-bit reduced instruction set computing (RISC) processor
  - Load-store architecture
  - Von Neumann architecture
    - Both data and instructions share a single bus interface.
  - Instruction set
    - 56 instructions as a subset of Thumb-1 (16-bit) and Thumb-2 (16/32-bit)
  - Supported interrupts
    - Non-maskable interrupt (NMI) + 1 to 32 physical interrupts
  - Supports sleep modes

Cortex-M0 Registers



Cortex-M0 Instruction Set

Instruction set

- The Cortex-M0 processor uses a superset of 16-bit Thumb-1 instructions + minimum subset of 32-bit Thumb-2 instructions

16-bit Thumb Instructions Supported on Cortex-M0									
ADCS	ADDS	ADR	ANDS	ASRS	B	BIC	BLX	BKPT	BX
CMN	CMP	CPS	EORS	LDM	LDR	LDRH	LDRSH	LDRB	LDRSB
LSLS	LSRS	MOV	MVN	MULS	NOP	ORRS	POP	PUSH	REV
REV16	REVSH	ROR	RSB	SBCS	SEV	STM	STR	STRH	STRB
SUBS	SVC	SXTB	SXTH	TST	UXTB	UXTH	WFE	WFI	YIELD

32-bit Thumb-2 Instructions Supported on Cortex-M0									
BL	DSB	DMB	ISB	MRS	MSR				

Cortex-M0 Instruction Set

Instruction Type	Instructions
Move	MOV, MOVS, MRS, MSR
Load/Store	LDR, LDRH, LDRB, LDRSH, LDRSB, LDM, LDMIA, STR, STRH, STRB, STMIA
Stack	PUSH, POP
Add, Subtract, Multiply	ADDS, ADCS, ADR, SUBS, SBCS, RSBS, MULS
Compare	CMP, CMN
Logical	ANDS, ORRS, EORS, MVNS, BICS, TST
Shift and Rotate	ASRS, LSRS, LSLS, RORS
Reverse	REV, REV16, REVSH
Extend	SXTB, SXTH, UXTB, UXTH
Conditional Branch	B, B <cond>, BL, BX, BLX
Memory Barrier	DMB, DSB, ISB
Exception	SVC, CPS
Sleep Mode	WFI, WFE, SEV
Other	NOP, BKPT, YIELD

Cortex-M0 Instruction Set

Cortex-M0 Suffix

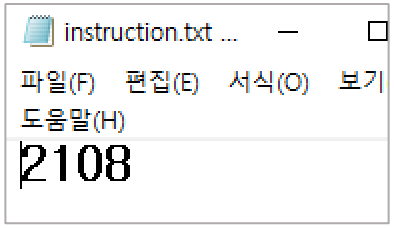
- Some instructions can be followed by suffixes to update processor flags or execute the instruction on a certain condition

Suffix	Description	Example	Example explanation
S	Update APSR (flags)	ADDS R1, #0x21	Add 0x21 to R1 and update APSR
EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE	Condition execution e.g., EQ= equal, NE= not equal, LT= less than	BNE label	Branch to the label if not equal

Cortex-M0 ISS

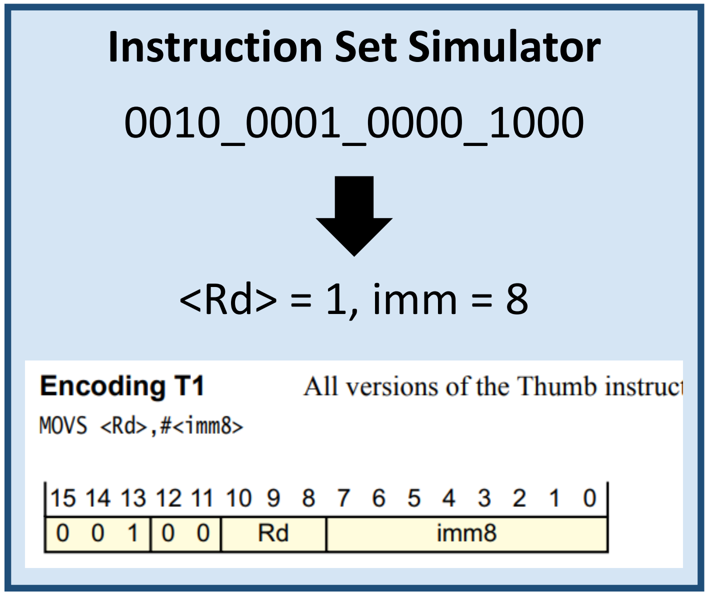
Cortex-M0 ISS

- **Cortex-M0 Instruction Set Simulator**
  - 명령어를 읽고 프로세서의 레지스터 변화 값을 보여주는 시뮬레이터



```
2108
```

instruction.txt



Instruction Set Simulator

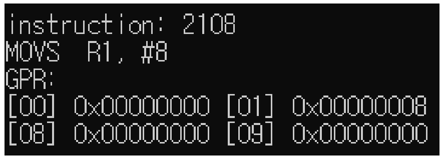
0010_0001_0000_1000

<Rd> = 1, imm = 8

Encoding T1 All versions of the Thumb instruct

`MOVS <Rd>,#<imm8>`

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	Rd			imm8							



```
instruction: 2108
MOVS  R1, #8
GPR:
[00] 0x00000000 [01] 0x00000008
[08] 0x00000000 [09] 0x00000000
```

print_status()

Cortex-M0 Instruction Set Simulator

Instructions

- **Cortex-M0 명령어**
  - 총 56개의 명령어 중 9개의 명령어를 지원하는 시뮬레이터 설계

1. ADDS <Rd>, <Rn>, #<imm3>
ex) ADDS R5, R1, #5 → <R5> = <R1> + 5

15 … 9	8 … 6	5 … 3	2 … 0
14_{ten}	imm3	Rn	Rd
7 bits	3 bits	3 bits	3 bits

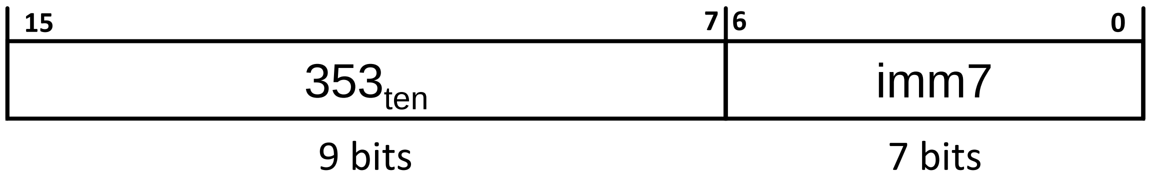
2. SUBS <Rd>, <Rn>, <Rm>
ex) SUBS R3, R6, R2 → <R3> = <R6> - <R2>

15 … 9	8 … 6	5 … 3	2 … 0
13_{ten}	Rm	Rn	Rd
7 bits	3 bits	3 bits	3 bits

Instructions

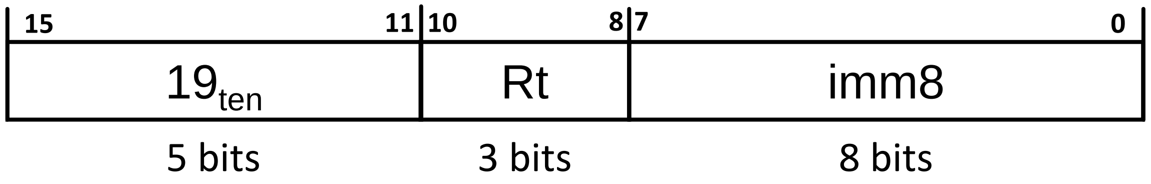
3. SUB SP, SP, #<imm7>

ex) SUB SP, SP, #12 → SP = SP - 12



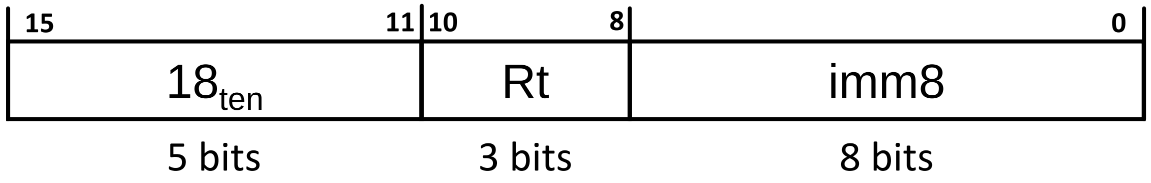
4. LDR <Rt>, [SP{, #<imm8>}]

ex) LDR R5, [SP, #8] → <R5> = data of (SP + 8) address



5. STR <Rt>, [SP{, #<imm8>}]

ex) STR R4, [SP, #4] → <R4> = data of (SP + 4) address



Instructions

6. MOVS <Rd>, #<imm8>

ex) MOVS R0, #1 → R0 = 1

15 … 11	10 … 8	7 … 0
4_{ten}	Rd	imm8
5 bits	3 bits	8 bits

7. BX <Rm>

ex) BX LR → <R15> = <LR>

15 … 10	9 … 8	7 … 7	6 … 3	2 … 0
142_{ten}	3_{ten}	0	Rm	(0)(0)(0)
6 bits	2 bits	1 bit	4 bits	3 bits

8. ANDS <Rdn>, <Rm>

ex) ANDS R4, R5 → <R4> = <R4> & <R5>

15 … 10	9 … 6	5 … 3	2 … 0
16_{ten}	0	Rm	Rdn
5 bits	4 bits	3 bits	3 bits

Instructions

9. ORRS <Rdn>, <Rm>

ex) ORRS R6, R7 → <R6> = <R6> | <R7>

15 10	9 6	5 3	2 0
16_{ten}	12_{ten}	Rm	Rdn
5 bits	4 bits	3 bits	3 bits

Function and Variable

- **Functions**
  - main
    - ISS의 전체적인 동작 실행
  - GPR_Initialization
    - General Purpose Register(GPR) 값을 0으로 초기화
  - IM_Initialization
    - Instruction Memory(IM) 값을 0으로 초기화
  - IM_Load
    - “instruction.txt”에 적혀있는 instruction을 읽어와서 IM에 저장
  - Execution
    - IM에서 instruction을 불러와서 명령 실행

Functions and Variables

- **Variables**
  - int GPR[16]
    - 16개의 GPR을 나타내는 배열
  - short Instruction_Mem[1024]
    - “instruction.txt”에서 읽은 Instruction을 저장하는 배열 (2 x 1024 = 2048bytes)
  - int Instruction_count
    - “instruction.txt”에 적혀 있는 명령어의 개수

Template

- **main**

```
int main(void) {

    print_logo();                  //로고 출력

    GPR_Initialization();      //General Purpose Register(GPR) 초기화
    IM_Initialization();        //Instruction Memory(IM) 초기화
    print_status();              //현재 GPR 상태 출력

    IM_Load("instruction.txt");            //실행할 명령어

    for (int i = 0; i < Instruction_count; i++) {    //명령어 개수만큼 실행
        Execution(i);             //명령어를 해석하고 명령 실행
        print_status();
    }

    return 0;
}
```

Template

- **print_logo**

```
// 로고 생성
void print_logo()
{
    // 학번, 이름 수정
    printf("**********************************************\n");
    printf("*                                            *\n");
    printf("*     Simple cortex_m0 Instruction Set Simulator     *\n");
    printf("*                                            *\n");
    printf("*                                            *\n");
    printf("*                  전자공학과 xxxxxxxxx 황용택  *\n");
    printf("**********************************************\n");

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

Template

- **print_status**

```
//GPR 값 출력
void print_status() {

    printf("\nGPR:\n");
    for (int i = 0; i < 16; i++)
    {
        printf("[%02d] %08X", i, GPR[i]);
        if (i % 8 < 7)
            printf(" ");
        else
            printf("\n");
    }
}
```

Template

- **GPR_Initialization**

```
void GPR_Initialization(void) {
    for (int i = 0; i < 16; i++)
    {
        GPR[i] = 0;        //General Purpose Register를 0으로 초기화
    }
}
```

- **IM_Initialization**

```
void IM_Initialization(void) {
    for (int i = 0; i < 1024; i++)
    {
        Instruction_Mem[i] = 0;   //Instruction Memory를 0으로 초기화
    }
}
```

Template

- **IM_Load(1/2)**

```
void IM_Load(char *instruction_file) {
    char instruction[5];
    short temp;
    FILE* fp = NULL;
    fp = fopen(instruction_file, "r");                    //instruction file(instruction.txt) 열기

    for (int i = 0; i < 1024; i++)
    {
        if (fgets(instruction, sizeof(char) * 5, fp))    //명령어를 정상적으로 읽었다면
        {
            Instruction_count++;                          //명령어의 개수 증가
            for (int j = 0; j < 4; j++)                   //4번 반복
            {
                //문자(영어)를 16진수 변환
                if (instruction[i] >= 'A') temp = instruction[j] - 'A' + 10;
                else temp = instruction[j] - '0';                //문자(숫자)를 16진수 변환
                //각각의 16진수 값을 IM과 OR 연산하여 저장
                Instruction_Mem[i] = Instruction_Mem[i] | (temp<<(4*(3-j)));
            }
```

Template

- **IM_Load(2/2)**

```
            fseek(fp, 2, SEEK_CUR);             //다음 명령어를 읽기 위해 2byte 이동
        }
        else break;                             //instruction.txt의 마지막까지 읽으면 break 실행
    }

    fclose(fp);
}
```

Template

- **Execution(1/2)**

```
void Execution(int Instruction_number) {
    short instruction = 0;
    unsigned int opcode = 0;
    int rm = 0;
    int rn = 0;
    int rd = 0;
    int imm = 0;

//Instruction Memory에서 실행할 명령어 읽기
    instruction = Instruction_Mem[Instruction_number];
    printf("instruction: %x\n", instruction);

    opcode = instruction >> 14;          //MSB 2비트를 읽어서 instruction class 구분
    if (opcode == 0) {                   //MSB 2비트가 0일 때
        opcode = instruction >> 11;      //MSB 5비트의 값을 확인
        if (opcode == 4){                //MSB 5비트의 값이 4일 때
            opcode = MOVS;               //opcode를 MOVS로 판별 (헤더파일에 선언)
    }
```

Template

▪ Execution(2/2)

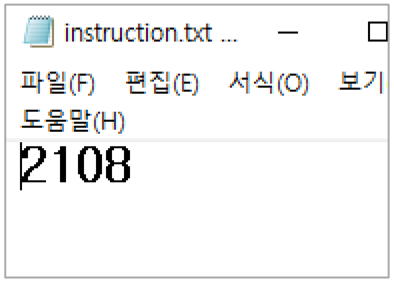
```
   switch (opcode) {
   case MOVS:                                //MOVS명령어
      rd = (instruction >> 8) & 7;    //rd 계산
      imm = (instruction)& 255;      //imm 계산
      GPR[rd] = imm;                       //레지스터의 값을 연산
      printf("MOVS  R%d, #%d", rd, imm);
      GPR[15] += 2;                        //PC값 업데이트
      break;

   //ADDS
   //SUBS
   //SUB
   //LDR
   //STR
   //BX
   //ANDS
   //ORRS
   }
}
```

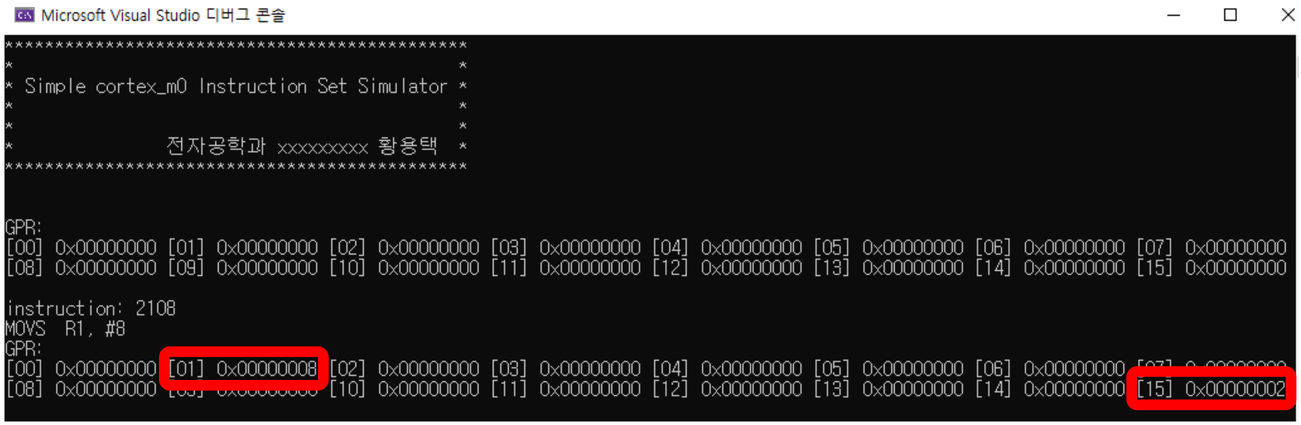
Template

▪ Template 실행 결과

- "instruction.txt"파일에 2108(MOVS R1, #8) 작성 후 ISS 실행
- 실행 결과, 1번 GPR의 값이 8로 변화하는 것을 확인, PC값인 15번 GPR의 값 변화 확인



```
2108
```



```
**********************************************
*                                            *
* Simple cortex_m0 Instruction Set Simulator *
*                                            *
*                                            *
*               전자공학과 xxxxxxxxx 황용택  *
**********************************************


GPR:
[00] 0x00000000 [01] 0x00000000 [02] 0x00000000 [03] 0x00000000 [04] 0x00000000 [05] 0x00000000 [06] 0x00000000 [07] 0x00000000
[08] 0x00000000 [09] 0x00000000 [10] 0x00000000 [11] 0x00000000 [12] 0x00000000 [13] 0x00000000 [14] 0x00000000 [15] 0x00000000

instruction: 2108
MOVS  R1, #8
GPR:
[00] 0x00000000 [01] 0x00000008 [02] 0x00000000 [03] 0x00000000 [04] 0x00000000 [05] 0x00000000 [06] 0x00000000 [07] 0x00000000
[08] 0x00000000 [09] 0x00000000 [10] 0x00000000 [11] 0x00000000 [12] 0x00000000 [13] 0x00000000 [14] 0x00000000 [15] 0x00000002
```

2108 (MOVS R1, #8)의 ISS 실행 결과

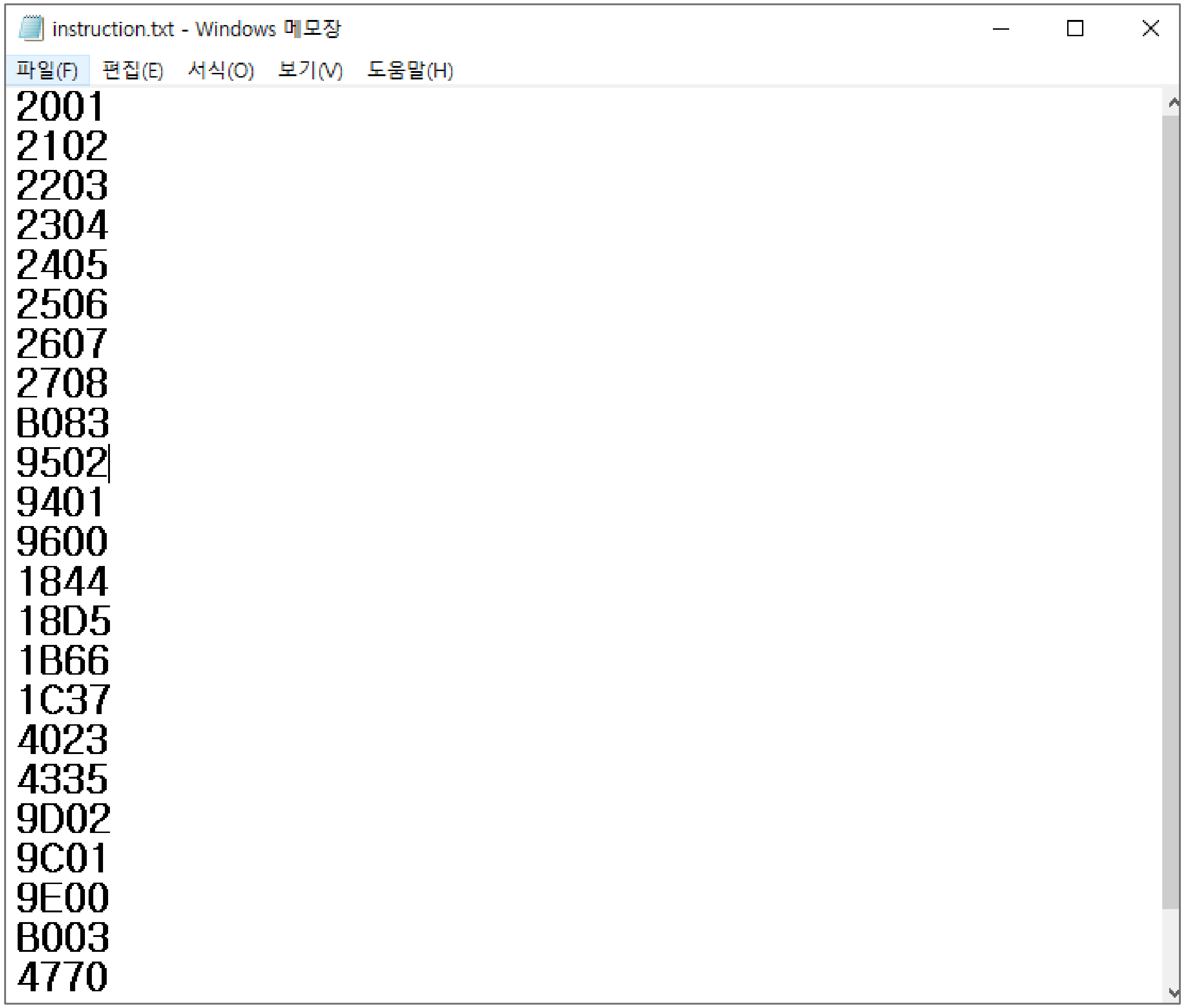
Task

Task

- **Execution 완성**
  - ADDS
  - SUBS
  - SUB
  - LDR
  - STR
  - BX
  - ANDS
  - ORRS
- **Data Memory 구현**
  - 선언
  - 초기화
  - 동작 검증 (LDR, STR)

Task

- **최종 검증 (Instruction.txt)**


instruction.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
2001
2102
2203
2304
2405
2506
2607
2708
B083
9502
9401
9600
1844
18D5
1B66
1C37
4023
4335
9D02
9C01
9E00
B003
4770
```