

# 명령어 목록

## 1. 산술 명령어

- 1.1 ADD ; 덧셈
- 1.2 SUB ; 뺄셈
- 1.3 MOV ; 값 복사
- 1.4 CMP ; 두 값 비교해 nzcvc 세팅 후 결과 버리기 (뺄셈 연산)
- 1.5 MUL ; 곱셈 후 하위 32비트 저장
- 1.6 MULH ; signed 곱셈 후 상위 32비트 저장
- 1.7 MULHU ; unsigned 곱셈 후 상위 32비트 저장
- 1.8 MULFX ; 16.16 고정소수점 signed 곱셈
- 1.9 DIV ; signed 나눗셈 후 몫 저장
- 1.10 DIVU ; unsigned 나눗셈 후 몫 저장
- 1.11 MOD ; signed 나눗셈 후 나머지 저장
- 1.12 MODU ; unsigned 나눗셈 후 나머지 저장

## 2. Shift 명령어

- 2.1 SHL ; 왼쪽으로 shift
- 2.2 ASR ; 부호 유지하며 오른쪽으로 shift
- 2.3 LSR ; 0으로 채우며 오른쪽으로 shift
- 2.4 ROL ; 왼쪽으로 rotate
- 2.5 ROR ; 오른쪽으로 rotate

## 3. 논리 명령어

- 3.1 AND ; 비트 AND
- 3.2 OR ; 비트 OR
- 3.3 XOR ; 비트 XOR
- 3.4 NOT ; 비트 NOT
- 3.5 BCHK ; 두 값 비트 단위로 비교해 nzcvc 세팅 후 결과 버리기 (AND 연산)

## 4. 메모리 명령어

- 4.1 LDR ; 메모리에서 32비트 값 로드
- 4.2 LDRB ; 메모리에서 8비트 값 0으로 확장해 로드
- 4.3 LDRSB ; 메모리에서 8비트 값 부호 확장해 로드
- 4.4 LDRH ; 메모리에서 16비트 값 0으로 확장해 로드
- 4.5 LDRSH ; 메모리에서 16비트 값 부호 확장해 로드
- 4.6 STR ; 메모리에 32비트 값 저장

4.7 STRB ; 메모리에 하위 8비트 값 저장

4.8 STRH ; 메모리에 하위 16비트 값 저장

## 5. 컨트롤 명령어

5.1 B ; 조건 분기

5.2 JMP ; 무조건 점프

## 6. 시스템 명령어

6.1 SYSCALL ; 시스템 콜

6.2 RETEXC ; 예외 처리 후 커널에서 유저로 복귀

6.3 SETINT ; 인터럽트 켜기

6.4 CLRINT ; 인터럽트 끄기

6.5 WAITINT ; 인터럽트 대기

6.6 LDRSYS ; 시스템 레지스터 로드

6.7 STRSYS ; 시스템 레지스터에 저장

---

더 작성할 것:

Nzcv 세팅 조건

# 명령어 세부 정의

## 1. 산술 명령어

### 1.1 Add

ADD{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
ADD	000001	0	0	rD			rA			rB			reserved		
ADDI	000001	1	0	rD			rA			immB					
ADDS	000001	0	1	rD			rA			rB			reserved		
ADDIS	000001	1	1	rD			rA			immB					

Add 연산은 입력으로 받은 두 값을 더해 목적지에 저장하는 명령어이다.

Add 명령어의 사용 형식 및 기능

- ADD rD, rA, rB

■  $R[rD] \leftarrow R[rA] + R[rB]$

- ADDI rD, rA, immB

■  $R[rD] \leftarrow R[rA] + \text{immB}$

- ADDS rD, rA, rB

■  $R[rD] \leftarrow R[rA] + R[rB]$  & update NZCV register

- ADDIS rD, rA, immB

■  $R[rD] \leftarrow R[rA] + \text{immB}$  & update NZCV register

Add 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 5 로 설정된 상태일 때,

- ADD R0, R1, R2 -> 000001\_00\_0000\_0001\_0010\_000000000000

->  $7 + 5 = 12$ 가 R0에 저장.

- ADDIS R0, R1, #6 -> 000001\_11\_0000\_0001\_0000000000000110

->  $7 + 6 = 13$ 이 R0에 저장, [N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

## 1.2 Subtract

SUB{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
SUB	000010	0	0	rD			rA			rB			reserved		
SUBI	000010	1	0	rD			rA			immB					
SUBS	000010	0	1	rD			rA			rB			reserved		
SUBIS	000010	1	1	rD			rA			immB					

Subtract 연산은 입력으로 받은 두 값을 빼 목적지에 저장하는 명령어이다.

Subtract 명령어의 사용 형식 및 기능

- SUB rD, rA, rB

■  $R[rD] \leftarrow R[rA] - R[rB]$

- SUBI rD, rA, immB

■  $R[rD] \leftarrow R[rA] - \text{immB}$

- SUBS rD, rA, rB

■  $R[rD] \leftarrow R[rA] - R[rB]$  & update NZCV register

- SUBIS rD, rA, immB

■  $R[rD] \leftarrow R[rA] - \text{immB}$  & update NZCV register

Subtract 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 5 로 설정된 상태일 때,

- SUB R0, R1, R2 -> 000010\_00\_0000\_0001\_0010\_000000000000

->  $7 - 5 = 2$ 가 R0에 저장.

- SUBIS R0, R1, #9 -> 000010\_11\_0000\_0001\_0000000000001001

->  $7 - 9 = -2$ 가 R0에 저장, [N = 1, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

### 1.3 Move

MOV{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
MOV	000011	0	0	rD			reserved			rB			reserved		
MOVI	000011	1	0	rD			immB								
MOVS	000011	0	1	rD			reserved			rB			reserved		
MOVIS	000011	1	1	rD			immB								

Move 연산은 입력으로 받은 값 하나를 그대로 목적지에 저장하는 명령어이다.

Move 명령어의 사용 형식 및 기능

- MOV rD, rB
  - $R[rD] \leftarrow R[rB]$
- MOVI rD, immB
  - $R[rD] \leftarrow \text{immB}$
- MOVS rD, rB
  - $R[rD] \leftarrow R[rB]$  & update NZCV register
- MOVIS rD, immB
  - $R[rD] \leftarrow \text{immB}$  & update NZCV register

Move 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 5 로 설정된 상태일 때,

- MOV R0, R2 -> 000011\_00\_0000\_0000\_0010\_000000000000
  - > R0에 5를 저장.
- MOVIS R0, #9 -> 000011\_11\_0000\_0000\_0000000000001001
  - > R0에 9를 저장, [N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

## 1.4 Compare

CMP{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
CMP	000100	0	0	reserved			rA			rB			reserved		
CMPI	000100	1	0	reserved			rA			immB					
CMPS	000100	0	1	reserved			rA			rB			reserved		
CMPIS	000100	1	1	reserved			rA			immB					

Compare 연산은 입력으로 받은 두 값을 비교해 NZCV 레지스터를 업데이트하는 명령어이다. Subtract 연산 수행 후, 컨트롤 시그널 Write Back Enable을 0으로 세팅해 연산 결과를 버리는 방식으로 동작한다.

※ 주의: S 모드비트가 1이어야 NZCV 레지스터가 업데이트되고, 0일 경우 NOP로 동작한다.

### Compare 명령어의 사용 형식 및 기능

- CMP rA, rB
  - $\_ \leq R[rA] - R[rB]$
- CMPI rA, immB
  - $\_ \leq R[rA] - \text{immB}$
- CMPS rA, rB
  - $\_ \leq R[rA] - R[rB] \ \& \ \text{update NZCV register}$
- CMPIS rA, immB
  - $\_ \leq R[rA] - \text{immB} \ \& \ \text{update NZCV register}$

### Compare 명령어의 사용 예시

R1 = 7, R2 = 5 로 설정된 상태일 때,

- CMPS R1, R2 -> 000100\_01\_0000\_0001\_0010\_000000000000
  - >  $7 - 5 = 2$ 이므로, [N = 0, Z = 0, C = 1, V = 0]을 NZCV 레지스터에 저장.
- CMPIS R1, #9 -> 000100\_11\_0000\_0001\_0000000000001001
  - >  $7 - 9 = -2$ 이므로, [N = 1, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.
- CMPIS R1, #7 -> 000100\_11\_0000\_0001\_0000000000000111
  - >  $7 - 7 = 0$ 이므로, [N = 0, Z = 1, C = 1, V = 0]을 NZCV 레지스터에 저장.

## 1.5 Multiplication

MUL{I}{S}	opcode	I	S	23	20	19	16	15	12	11	0
MUL	000101	0	0	rD		rA		rB		reserved	
MULI	000101	1	0	rD		rA		immB			
MULS	000101	0	1	rD		rA		rB		reserved	
MULIS	000101	1	1	rD		rA		immB			

Multiplication 연산은 입력으로 받은 두 값을 곱해 하위 32비트를 목적지에 저장하는 명령어이다.

### Multiplication 명령어의 사용 형식 및 기능

- MUL rD, rA, rB

■  $R[rD] \leftarrow (R[rA] \times R[rB])[31:0]$

- MULI rD, rA, immB

■  $R[rD] \leftarrow (R[rA] \times \text{immB})[31:0]$

- MULS rD, rA, rB

■  $R[rD] \leftarrow (R[rA] \times R[rB])[31:0] \ \& \ \text{update NZCV register}$

- MULIS rD, rA, immB

■  $R[rD] \leftarrow (R[rA] \times \text{immB})[31:0] \ \& \ \text{update NZCV register}$

### Multiplication 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 0x7000\_0000 로 설정된 상태일 때,

- MULS R0, R1, R2 -> 000101\_01\_0000\_0001\_0010\_000000000000

->  $0x7 \times 0x7000\_0000 = 0x0003\_1000\_0000$  중 하위 32비트인 0x1000\_0000이 R0에 저장,  
[N = 0, Z = 0, C = 0, V = 1]을 NZCV 레지스터에 저장.

- MULI R0, R1, #5 -> 000101\_10\_0000\_0001\_0000000000000101

->  $7 \times 5 = 35$ 가 R0에 저장.

## 1.6 Multiplication High

MULH{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
MULH	000110	0	0	rD			rA			rB			reserved		
MULHI	000110	1	0	rD			rA			immB					
MULHS	000110	0	1	rD			rA			rB			reserved		
MULHIS	000110	1	1	rD			rA			immB					

Multiplication High 연산은 입력으로 받은 두 값을 부호를 포함해 곱해 상위 32비트를 목적지에 저장하는 명령어이다.

### Multiplication High 명령어의 사용 형식 및 기능

- MULH rD, rA, rB

■  $R[rD] \leftarrow (\text{signed } R[rA] \times \text{signed } R[rB])[63:32]$

- MULHI rD, rA, immB

■  $R[rD] \leftarrow (\text{signed } R[rA] \times \text{sign-ext}(\text{immB}))[63:32]$

- MULHS rD, rA, rB

■  $R[rD] \leftarrow (\text{signed } R[rA] \times \text{signed } R[rB])[63:32] \text{ \& update NZCV register}$

- MULHIS rD, rA, immB

■  $R[rD] \leftarrow (\text{signed } R[rA] \times \text{sign-ext}(\text{immB}))[63:32] \text{ \& update NZCV register}$

### Multiplication High 명령어의 사용 예시

R0 = 0, R1 = 0x7, R2 = 0x7000\_0000 로 설정된 상태일 때,

- MULH R0, R1, R2 -> 000110\_00\_0000\_0001\_0010\_000000000000

->  $0x7 \times 0x7000\_0000 = 0x0003\_1000\_0000$  중 상위 32비트인 0x0000\_0003이 R0에 저장.

- MULHIS R0, R1, #0xFFFF -> 000110\_11\_0000\_0001\_1111111111111111

->  $0x7 \times (-1) = -7 = 0xFFFF\_FFFF\_FFFF\_FFF9$  중 상위 32비트인 0xFFFF\\_FFFF이 R0에 저장,

[N = 1, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.



## 1.7 Multiplication High Unsigned

MULHU{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
MULHU	000111	0	0	rD			rA			rB			reserved		
MULHUI	000111	1	0	rD			rA			immB					
MULHUS	000111	0	1	rD			rA			rB			reserved		
MULHUIS	000111	1	1	rD			rA			immB					

Multiplication High Unsigned 연산은 입력으로 받은 두 값을 부호 없이 곱해 상위 32비트를 목적지에 저장하는 명령어이다.

Multiplication High unsigned 명령어의 사용 형식 및 기능

- MULHU rD, rA, rB

■  $R[rD] \leftarrow (\text{unsigned } R[rA] \times \text{unsigned } R[rB])[63:32]$

- MULHUI rD, rA, immB

■  $R[rD] \leftarrow (\text{unsigned } R[rA] \times \text{zero-ext}(\text{immB}))[63:32]$

- MULHUS rD, rA, rB

■  $R[rD] \leftarrow (\text{unsigned } R[rA] \times \text{unsigned } R[rB])[63:32] \text{ \& update NZCV register}$

- MULHUIS rD, rA, immB

■  $R[rD] \leftarrow (\text{unsigned } R[rA] \times \text{zero-ext}(\text{immB}))[63:32] \text{ \& update NZCV register}$

Multiplication High Unsigned 명령어의 사용 예시

R0 = 0, R1 = 0x7, R2 = 0x7000\_0000 로 설정된 상태일 때,

- MULHU R0, R1, R2 -> 000111\_00\_0000\_0001\_0010\_000000000000

->  $0x7 \times 0x7000\_0000 = 0x0003\_1000\_0000$  중 상위 32비트인 0x0000\_0003이 R0에 저장.

- MULHUIS R0, R1, #0xFFFF -> 000111\_11\_0000\_0001\_1111111111111111

->  $0x7 \times 0x0000\_FFFF = 0x0000\_0000\_0006\_FFF9$  중 상위 32비트인 0x0000\_0000이 R0에 저장,  
[N = 0, Z = 1, C = 0, V = 0]을 NZCV 레지스터에 저장.

## 1.8 Multiplication Fixed-Point

MULFX{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
MULFX	001000	0	0	rD			rA			rB			reserved		
MULFXI	001000	1	0	rD			rA			immB					
MULFXS	001000	0	1	rD			rA			rB			reserved		
MULFXIS	001000	1	1	rD			rA			immB					

Multiplication Fixed-Point 연산은 입력으로 받은 두 16.16 고정 소수점 값을 부호를 포함해 곱해, 64비트 결과값 중 중간 32비트(비트 47~16)를 목적지에 저장하는 명령어이다.

※ 주의: immediate 모드에서의 즉시값 16비트는 8.8 형식으로 해석되며, 16.16으로 부호 확장되어 연산된다.

### Multiplication Fixed-Point 명령어의 사용 형식 및 기능

- MULFX rD, rA, rB
  - $R[rD] \leftarrow (R[rA] \times R[rB])[47:16]$
- MULFXI rD, rA, immB
  - $R[rD] \leftarrow (R[rA] \times \text{sign-ext}(\text{immB}))[47:16]$
- MULFXS rD, rA, rB
  - $R[rD] \leftarrow (R[rA] \times R[rB])[47:16] \text{ \& update NZCV register}$
- MULFXIS rD, rA, immB
  - $R[rD] \leftarrow (R[rA] \times \text{sign-ext}(\text{immB}))[47:16] \text{ \& update NZCV register}$

### Multiplication Fixed-Point 명령어의 사용 예시

## 1.9 Division

DIV{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
DIV	001001	0	0	rD			rA			rB			reserved		
DIVI	001001	1	0	rD			rA			immB					
DIVS	001001	0	1	rD			rA			rB			reserved		
DIVIS	001001	1	1	rD			rA			immB					

### 1.10 Division Unsigned

DIVU{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
DIVU	001010	0	0	rD			rA			rB			reserved		
DIVUI	001010	1	0	rD			rA			immB					
DIVUS	001010	0	1	rD			rA			rB			reserved		
DIVUIS	001010	1	1	rD			rA			immB					

### 1.11 Modulo

MOD{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
MOD	001011	0	0	rD			rA			rB			reserved		
MODI	001011	1	0	rD			rA			immB					
MODS	001011	0	1	rD			rA			rB			reserved		
MODIS	001011	1	1	rD			rA			immB					

## 1.12 Modulo Unsigned

MODU{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
MODU	001100	0	0	rD			rA			rB			reserved		
MODUI	001100	1	0	rD			rA			immB					
MODUS	001100	0	1	rD			rA			rB			reserved		
MODUIS	001100	1	1	rD			rA			immB					

## 2. Shift 명령어

### 2.1 Shift Left

SHL{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
SHL	011000	0	0	rD			rA			rB			reserved		
SHLI	011000	1	0	rD			rA			immB					
SHLS	011000	0	1	rD			rA			rB			reserved		
SHLIS	011000	1	1	rD			rA			immB					

Shift Left 연산은 입력 A를 입력 B만큼, 왼쪽으로 shift하여 목적지에 저장하는 명령어이다.

Shift Left 명령어의 사용 형식 및 기능

- SHL rD, rA, rB

■  $R[rD] \leftarrow R[rA] \ll R[rB]$

- SHLI rD, rA, immB

■  $R[rD] \leftarrow R[rA] \ll \text{immB}$

- SHLS rD, rA, rB

■  $R[rD] \leftarrow R[rA] \ll R[rB]$  & update NZCV register

- SHLIS rD, rA, immB

■  $R[rD] \leftarrow R[rA] \ll \text{immB}$  & update NZCV register

Shift Left 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 2 로 설정된 상태일 때,

- SHL R0, R1, R2 -> 011000\_00\_0000\_0001\_0010\_000000000000

-> 7 (0b0000\_0111) << 2 = 28 (0b0001\_1100)이 R0에 저장.

- SHLIS R0, R1, #4 -> 011000\_11\_0000\_0001\_0000000000000100

-> 7 (0b0000\_0111) << 4 = 112 (0b0111\_0000)가 R0에 저장,

[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

## 2.2 Arithmetic Shift Right

ASR{I}{S}	opcode	I	S	23	20	19	16	15	12	11	0
ASR	011001	0	0	rD		rA		rB		reserved	
ASRI	011001	1	0	rD		rA		immB			
ASRS	011001	0	1	rD		rA		rB		reserved	
ASRIS	011001	1	1	rD		rA		immB			

Arithmetic Shift Right 연산은 입력 A를 입력 B만큼, 부호 비트를 유지하며 우측으로 shift하여 목적지에 저장하는 명령어이다.

Arithmetic Shift Right 명령어의 사용 형식 및 기능

- ASR rD, rA, rB

■  $R[rD] \leftarrow R[rA] \gg R[rB]$

- ASRI rD, rA, immB

■  $R[rD] \leftarrow R[rA] \gg \text{immB}$

- ASRS rD, rA, rB

■  $R[rD] \leftarrow R[rA] \gg R[rB]$  & update NZCV register

- ASRIS rD, rA, immB

■  $R[rD] \leftarrow R[rA] \gg \text{immB}$  & update NZCV register

Arithmetic Shift Right 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 2, R3 = -8 로 설정된 상태일 때,

- ASR R0, R3, R2 -> 011001\_00\_0000\_0011\_0010\_000000000000

-> -8 (0b1111\_1000) >> 2 = -2 (0b1111\_1110)가 R0에 저장.

- ASRIS R0, R1, #1 -> 011001\_11\_0000\_0001\_0000000000000001

-> 7 (0b0000\_0111) >> 1 = 3 (0b0000\_0011)가 R0에 저장,

[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.



## 2.3 Logical Shift Right

LSR{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
LSR	011010	0	0	rD			rA			rB			reserved		
LSRI	011010	1	0	rD			rA			immB					
LSRS	011010	0	1	rD			rA			rB			reserved		
LSRIS	011010	1	1	rD			rA			immB					

Logical Shift Right 연산은 입력 A를 입력 B만큼, 부호 비트를 유지하지 않고 우측으로 shift하여 목적지에 저장하는 명령어이다.

Logical Shift Right 명령어의 사용 형식 및 기능

- LSR rD, rA, rB

■  $R[rD] \leftarrow R[rA] \gg R[rB]$

- LSRI rD, rA, immB

■  $R[rD] \leftarrow R[rA] \gg \text{immB}$

- LSRS rD, rA, rB

■  $R[rD] \leftarrow R[rA] \gg R[rB]$  & update NZCV register

- LSRIS rD, rA, immB

■  $R[rD] \leftarrow R[rA] \gg \text{immB}$  & update NZCV register

Logical Shift Right 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 2, R3 = -8 로 설정된 상태일 때,

- LSR R0, R3, R2 -> 011010\_00\_0000\_0011\_0010\_000000000000

-> -8 (0b1111\_1000) >> 2 = 62 (0b0011\_1110)가 R0에 저장. (8bit 기준)

- LSRIS R0, R1, #1 -> 011010\_11\_0000\_0001\_0000000000000001

-> 7 (0b0000\_0111) >> 1 = 3 (0b0000\_0011)가 R0에 저장,

[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

## 2.4 Rotate Left

ROL{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
ROL	011011	0	0	rD			rA			rB			reserved		
ROLI	011011	1	0	rD			rA			immB					
ROLS	011011	0	1	rD			rA			rB			reserved		
ROLIS	011011	1	1	rD			rA			immB					

Rotate Left 연산은 입력 A를 입력 B만큼, 비트를 좌측으로 순환시켜 목적지에 저장하는 명령어이다.

Rotate Left 명령어의 사용 형식 및 기능

- ROL rD, rA, rB

■  $R[rD] \leftarrow R[rA] \text{ ROL } R[rB]$ , Shift-out bits are rotated to Left side

- ROLI rD, rA, immB

■  $R[rD] \leftarrow R[rA] \text{ ROL } \text{immB}$ , Shift-out bits are rotated to Left side

- ROLS rD, rA, rB

■  $R[rD] \leftarrow R[rA] \text{ ROL } R[rB]$ , Shift-out bits are rotated to Left side

& update NZCV register

- ROLIS rD, rA, immB

■  $R[rD] \leftarrow R[rA] \text{ ROL } \text{immB}$ , Shift-out bits are rotated to Left side

& update NZCV register

Rotate Left 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 3, R3 = -14 로 설정된 상태일 때,

- ROL R0, R3, R2 -> 011011\_00\_0000\_0011\_0010\_000000000000

-> -14 (0b1111\_0010) ROL 3 = 94 (0b0101\_1110)가 R0에 저장. (8bit 기준)

- ROLIS R0, R1, #1 -> 011011\_11\_0000\_0001\_0000000000000001

-> 7 (0b0000\_0111) ROL 1 = -125 (0b1000\_0011)가 R0에 저장,

[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장. (8bit 기준)

## 2.5 Rotate Right

ROR{I}{S}	opcode	I	S	23	20	19	16	15	12	11	0
ROR	011100	0	0	rD		rA		rB		reserved	
RORI	011100	1	0	rD		rA		immB			
RORS	011100	0	1	rD		rA		rB		reserved	
RORIS	011100	1	1	rD		rA		immB			

Rotate Right 연산은 입력 A를 입력 B만큼, 비트를 우측으로 순환시켜 목적지에 저장하는 명령어이다.

Rotate Right 명령어의 사용 형식 및 기능

- ROR rD, rA, rB
  - $R[rD] \leftarrow R[rA] \text{ ROR } R[rB]$ , Shift-out bits are rotated to Right side
- RORI rD, rA, immB
  - $R[rD] \leftarrow R[rA] \text{ ROR } \text{immB}$ , Shift-out bits are rotated to Right side
- RORS rD, rA, rB
  - $R[rD] \leftarrow R[rA] \text{ ROR } R[rB]$ , Shift-out bits are rotated to Right side
  - & update NZCV register
- RORIS rD, rA, immB
  - $R[rD] \leftarrow R[rA] \text{ ROR } \text{immB}$ , Shift-out bits are rotated to Right side
  - & update NZCV register

Rotate Right 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 3, R3 = -14 로 설정된 상태일 때,

- ROR R0, R3, R2 -> 011100\_00\_0000\_0011\_0010\_000000000000
  - > -14 (0b1111\_0010) ROR 3 = 151 (0b1001\_0111)가 R0에 저장. (8bit 기준)
- RORIS R0, R1, #1 -> 011100\_11\_0000\_0001\_0000000000000001
  - > 7 (0b0000\_0111) ROR 1 = 14 (0b0000\_1110)가 R0에 저장,
  - [N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장. (8bit 기준)

### 3. 논리 명령어

#### 3.1 And

AND{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
AND	100001	0	0	rD			rA			rB			reserved		
ANDI	100001	1	0	rD			rA			immB					
ANDS	100001	0	1	rD			rA			rB			reserved		
ANDIS	100001	1	1	rD			rA			immB					

And 연산은 입력으로 받은 두 값을 비트단위 AND 하여 목적지에 저장하는 명령어이다.

And 명령어의 사용 형식 및 기능

- AND rD, rA, rB
  - $R[rD] \leftarrow R[rA] \& R[rB]$
- ANDI rD, rA, immB
  - $R[rD] \leftarrow R[rA] \& \text{immB}$
- ANDS rD, rA, rB
  - $R[rD] \leftarrow R[rA] \& R[rB] \& \text{update NZCV register}$
- ANDIS rD, rA, immB
  - $R[rD] \leftarrow R[rA] \& \text{immB} \& \text{update NZCV register}$

And 명령어의 사용 예시

R0 = 0, R1 = 109, R2 = 55 로 설정된 상태일 때,

- AND R0, R1, R2 -> 100001\_00\_0000\_0001\_0010\_000000000000
  - > 109 (0b0110\_1101) & 55 (0b0011\_0111) = 37 (0b0010\_0101)가 R0에 저장.
- ANDIS R0, R1, #227 -> 100001\_11\_0000\_0001\_0000000011100011
  - > 109 (0b0110\_1101) & 227 (0b1110\_0011) = 97 (0b0110\_0001)가 R0에 저장,  
[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

### 3.2 Or

OR{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
OR	100010	0	0	rD			rA			rB			reserved		
ORI	100010	1	0	rD			rA			immB					
ORS	100010	0	1	rD			rA			rB			reserved		
ORIS	100010	1	1	rD			rA			immB					

Or 연산은 입력으로 받은 두 값을 비트단위 OR 하여 목적지에 저장하는 명령어이다.

Or 명령어의 사용 형식 및 기능

- OR rD, rA, rB

■  $R[rD] \leftarrow R[rA] \mid R[rB]$

- ORI rD, rA, immB

■  $R[rD] \leftarrow R[rA] \mid \text{immB}$

- ORS rD, rA, rB

■  $R[rD] \leftarrow R[rA] \mid R[rB] \ \& \ \text{update NZCV register}$

- ORIS rD, rA, immB

■  $R[rD] \leftarrow R[rA] \mid \text{immB} \ \& \ \text{update NZCV register}$

Or 명령어의 사용 예시

R0 = 0, R1 = 109, R2 = 55 로 설정된 상태일 때,

- OR R0, R1, R2 -> 100010\_00\_0000\_0001\_0010\_000000000000

-> 109 (0b0110\_1101) | 55 (0b0011\_0111) = 127 (0b0111\_1111)가 R0에 저장.

- ORIS R0, R1, #227 -> 100010\_11\_0000\_0001\_0000000011100011

-> 109 (0b0110\_1101) | 227 (0b1110\_0011) = 239 (0b1110\_1111)가 R0에 저장,

[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

### 3.3 Xor

XOR{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
XOR	100011	0	0	rD			rA			rB			reserved		
XORI	100011	1	0	rD			rA			immB					
XORS	100011	0	1	rD			rA			rB			reserved		
XORIS	100011	1	1	rD			rA			immB					

Xor 연산은 입력으로 받은 두 값을 비트단위 XOR 하여 목적지에 저장하는 명령어이다.

Xor 명령어의 사용 형식 및 기능

- XOR rD, rA, rB

■  $R[rD] \leftarrow R[rA] \wedge R[rB]$

- XORI rD, rA, immB

■  $R[rD] \leftarrow R[rA] \wedge \text{immB}$

- XORS rD, rA, rB

■  $R[rD] \leftarrow R[rA] \wedge R[rB]$  & update NZCV register

- XORIS rD, rA, immB

■  $R[rD] \leftarrow R[rA] \wedge \text{immB}$  & update NZCV register

Xor 명령어의 사용 예시

R0 = 0, R1 = 109, R2 = 55 로 설정된 상태일 때,

- XOR R0, R1, R2 -> 100011\_00\_0000\_0001\_0010\_000000000000

-> 109 (0b0110\_1101)  $\wedge$  55 (0b0011\_0111) = 90 (0b0101\_1010)가 R0에 저장.

- XORIS R0, R1, #227 -> 100011\_11\_0000\_0001\_0000000011100011

-> 109 (0b0110\_1101)  $\wedge$  227 (0b1110\_0011) = 142 (0b1000\_1110)가 R0에 저장,

[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

### 3.4 Not

NOT{I}{S}	opcode	I	S	23	20	19	16	15	12	11	0
NOT	100100	0	0	rD	reserved			rB	reserved		
NOTS	100100	0	1	rD	reserved			rB	reserved		

Not 연산은 입력값 B에 대해 각 비트의 보수를 구해 목적지에 저장하는 단항 명령어이다.

#### Not 명령어의 사용 형식 및 기능

- NOT rD, rB

■  $R[rD] \leftarrow \sim R[rB]$

- NOTS rD, rB

■  $R[rD] \leftarrow \sim R[rB]$  & update NZCV register

#### Not 명령어의 사용 예시

R0 = 0, R1 = 109, R2 = 55 로 설정된 상태일 때,

- NOT R0, R1 -> 100100\_00\_0000\_0000\_0001\_000000000000  
->  $\sim 109$  (0b0110\_1101) = -110 (0b1001\_0010)가 R0에 저장.
- NOTS R0, R2 -> 100100\_11\_0000\_0000\_0010\_000000000000  
->  $\sim 55$  (0b0011\_0111) = -56 (0b1100\_1000)가 R0에 저장,  
[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

### 3.5 Bit Check

BCHK{I}{S}	opcode	I	S	23		20	19		16	15		12	11		0
BCHK	100101	0	0	reserved			rA			rB			reserved		
BCHKI	100101	1	0	reserved			rA			immB					
BCHKS	100101	0	1	reserved			rA			rB			reserved		
BCHKIS	100101	1	1	reserved			rA			immB					

Bit Check 연산은 입력으로 받은 두 값을 비트 단위로 비교해 NZCV 레지스터를 업데이트하는 명령어이다. AND 연산 수행 후, 컨트롤 시그널 Write Back Enable을 0으로 세팅해 연산 결과를 버리는 방식으로 동작한다.

※ 주의: S 모드비트가 1이어야 NZCV 레지스터가 업데이트되고, 0일 경우 NOP로 동작한다.

#### Bit Check 명령어의 사용 형식 및 기능

- BCHK rA, rB

■  $\_ \leq R[rA] \& R[rB]$

- BCHKI rA, immB

■  $\_ \leq R[rA] \& immB$

- BCHKS rA, rB

■  $\_ \leq R[rA] \& R[rB] \& \text{update NZCV register}$

- BCHKIS rA, immB

■  $\_ \leq R[rA] \& immB \& \text{update NZCV register}$

#### Bit Check 명령어의 사용 예시

R1 = 237, R2 = 55 로 설정된 상태일 때,

- BCHKS R1, R2 -> 100101\_01\_0000\_0001\_0010\_000000000000

-> 237 (0b1110\_1101) & 55 (0b0011\_0111) = 37 (0b0010\_0101)이므로,

[N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장. (8bit 기준)

- BCHKIS R1, #227 -> 100101\_11\_0000\_0001\_0000000011100011

-> 237 (0b1110\_1101) & 227 (0b1110\_0011) = 225 (0b1110\_0001)이므로,

[N = 1, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장. (8bit 기준)