레지스터 목록

1. Register File

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

GISA는 16개의 범용 레지스터를 가진다. 각 레지스터는 32비트의 크기를 가진다.
R13 = 장거리 점프 주소계산 위한 레지스터.
R15 = 리턴 레지스터

2. Status Register

N	Z	C	V

산술 및 논리연산 수행 후 상태를 저장하는 레지스터이다. 각 레지스터는 1비트의 크기를 가진다.

3. Program Counter

다음 명령어의 주소를 저장하는 레지스터이다. 32비트의 크기를 가진다.

명령어 목록

1. 산술 명령어

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

추후 더 작성할 것:
Nzcv 세팅 조건 명령마다 명시하기
Branch 컨디션 코드 정의하기
나눗셈 명령어들 사용예시 추가

명령어 세부 정의

1. 산술 명령어

1.1 Move

MOV{I}{Z}	opcode	I	Z	23		20	19		16	15		12	11		0
MOV	000000	0	0	rD			reserved			rB			reserved		
MOVI	000000	1	0	rD			immB								
MOVIZ	000000	1	1	rD			immB								

Move 연산은 입력으로 받은 값 하나를 그대로 목적지에 저장하는 명령어이다.
※ 주의: Move는 SetCC mode를 지원하지 않으므로, S 모드비트 대신 Zero-extension 모드비트를 사용한다. 비활성화 시 20비트 즉시값을 부호확장해 목적지 레지스터에 저장하고, 활성화 시 0확장해 목적지 레지스터에 저장한다. Register mode에서는 확장이 존재하지 않으므로 Zero-extension 모드는 Immediate 모드에만 존재한다.

Move 명령어의 사용 형식 및 기능

- MOV rD, rB
  - ■ R[rD] <= R[rB]
- MOVI rD, immB
  - ■ R[rD] <= signExt(immB)
- MOVIZ rD, immB
  - ■ R[rD] <= zeroExt(immB)

Move 명령어의 사용 예시

R0 = 0, R1 = 7, R2 = 5 로 설정된 상태일 때,

- MOV R0, R2 -> 000000_00_0000_0000_0010_000000000000
  -> R0에 5를 저장.
- MOVI R0, #1,048,567(-9) -> 000000_10_0000_11111111111111110111
  -> R0에 -9를 저장.
- MOVIZ R0, #1,048,567(-9) -> 000000_11_0000_11111111111111110111
  -> R0에 1,048,567를 저장.

1.2 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] <= R[rA] + R[rB]
- ADDI rD, rA, immB
  - R[rD] <= R[rA] + signExt(immB)
- ADDS rD, rA, rB
  - R[rD] <= R[rA] + R[rB] & update NZCV register
- ADDIS rD, rA, immB
  - R[rD] <= R[rA] + signExt(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 = 12가 R0에 저장.
- ADDIS R0, R1, #6 -> 000001_11_0000_0001_0000000000000110
  -> 7 + 6 = 13이 R0에 저장, [N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = cout

V = (A[31] == B[31]) && (A[31] != result[31])

1.3 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] <= R[rA] - R[rB]
- SUBI rD, rA, immB
  - R[rD] <= R[rA] - signExt(immB)
- SUBS rD, rA, rB
  - R[rD] <= R[rA] - R[rB] & update NZCV register
- SUBIS rD, rA, immB
  - R[rD] <= R[rA] - signExt(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 = 2가 R0에 저장.
- SUBIS R0, R1, #9 -> 000010_11_0000_0001_0000000000001001
  -> 7 -9 = -2가 R0에 저장, [N = 1, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = cout

V = (A[31] != B[31]) && (A[31] != result[31])

1.4 Move High

MOVH{L}{K}	opcode	L	r	23	20	19	16	15	12	11	0
MOVH	000011	0	r	rD		reserved		immB			
MOVHL	000011	1	r	rD		immB					

Move High 연산은 입력으로 받은 즉시값을 목적지 레지스터의 상위 16/20비트에 저장하며, 기본적으로 하위 비트는 0으로 세팅하는 명령어이다.

※ 주의: Move High는 Register mode를 지원하지 않으므로, I 모드비트 대신 Long 모드비트를 사용한다. 비활성화 시 16비트 즉시값을 받아 목적지 레지스터의 [31:16]비트를 채우고, 활성화 시 20비트 즉시값을 받아 목적지 레지스터의 [31:12]비트를 채운다.

※ 주의: Move High는 SetCC mode를 지원하지 않으므로, S 모드비트 자리는 사용하지 않는다..

Move High 명령어의 사용 형식 및 기능

- MOVH rD, immB
  - ■ R[rD][31:16] <= immB & R[rD][15:0] <= 16'b0
- MOVHL rD, immB
  - ■ R[rD][31:12] <= immB & R[rD][11:0] <= 12'b0

Move High 명령어의 사용 예시

R0 = 29, R1 = 7, R2 = 5 로 설정된 상태일 때,

- MOVH R0, #59 -> 000011_00_0000_0000_0000000000111011
  -> R0에 0b0000_0000_0011_1011_0000_0000_0000_0000을 저장.
- MOVHL R0, #59 -> 000011_10_0000_0000_0000000000111011
  -> R0에 0b0000_0000_0000_0011_1011_0000_0000_0000을 저장.

1.5 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
  - ■ _ <= R[rA] - R[rB]
- CMPI rA, immB
  - ■ _ <= R[rA] - signExt(immB)
- CMPS rA, rB
  - ■ _ <= R[rA] - R[rB] & update NZCV register
- CMPIS rA, immB
  - ■ _ <= R[rA] - signExt(immB) & 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 = 2이므로, [N = 0, Z = 0, C = 1, V = 0]을 NZCV 레지스터에 저장.
- CMPIS R1, #9 -> 000100_11_0000_0001_0000000000001001
  -> 7 - 9 = -2이므로, [N = 1, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.
- CMPIS R1, #7 -> 000100_11_0000_0001_0000000000000111
  -> 7 - 7 = 0이므로, [N = 0, Z = 1, C = 1, V = 0]을 NZCV 레지스터에 저장.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = cout

V = (A[31] != B[31]) && (A[31] != result[31])

1.6 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] <= (R[rA] × R[rB])[31:0]
- MULI rD, rA, immB
  - ■ R[rD] <= (R[rA] × signExt(immB))[31:0]
- MULS rD, rA, rB
  - ■ R[rD] <= (R[rA] × R[rB])[31:0] & update NZCV register
- MULIS rD, rA, immB
  - ■ R[rD] <= (R[rA] × signExt(immB))[31:0] & 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 * 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 * 5 = 35가 R0에 저장.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = result[32]

V = (result[63:32] != 0)

1.7 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] <= (signed R[rA] × signed R[rB])[63:32]
- MULHI rD, rA, immB
  - R[rD] <= (signed R[rA] × signExt(immB))[63:32]
- MULHS rD, rA, rB
  - R[rD] <= (signed R[rA] × signed R[rB])[63:32] & update NZCV register
- MULHIS rD, rA, immB
  - R[rD] <= (signed R[rA] × signExt(immB))[63:32] & 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 * 0x7000_0000 = 0x0003_1000_0000 중 상위 32비트인 0x0000_0003이 R0에 저장.
- MULHIS R0, R1, #0xFFFF -> 000110_11_0000_0001_1111111111111111
  -> 0x7 * (-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.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 0

V = 0

1.8 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] <= (unsigned R[rA] × unsigned R[rB])[63:32]
- MULHUI rD, rA, immB
  - ■ R[rD] <= (unsigned R[rA] × zeroExt(immB))[63:32]
- MULHUS rD, rA, rB
  - ■ R[rD] <= (unsigned R[rA] × unsigned R[rB])[63:32] & update NZCV register
- MULHUIS rD, rA, immB
  - ■ R[rD] <= (unsigned R[rA] × zeroExt(immB))[63:32] & 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 * 0x7000_0000 = 0x0003_1000_0000 중 상위 32비트인 0x0000_0003이 R0에 저장.

- MULHUIS R0, R1, #0xFFFF -> 000111_11_0000_0001_1111111111111111

  -> 0x7 * 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.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 0

V = 0

1.9 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] <= (R[rA] × R[rB])[47:16]
- MULFXI rD, rA, immB
  - R[rD] <= (R[rA] × signExt(immB))[47:16]
- MULFXS rD, rA, rB
  - R[rD] <= (R[rA] × R[rB])[47:16] & update NZCV register
- MULFXIS rD, rA, immB
  - R[rD] <= (R[rA] × signExt(immB))[47:16] & update NZCV register

Multiplication Fixed-Point 명령어의 사용 예시

R0 = 0, R1 = 0x0001_8000 (1.5), R2 = 0x0002_0000 (2.0) 로 설정된 상태일 때,

- MULFX R0, R1, R2 -> 001000_00_0000_0001_0010_000000000000
  -> 0x0001_8000 * 0x0002_0000 = 0x0000_0003_0000_0000 중 [47:16]비트인 0x0003_0000 (3.0)이 R0에 저장.
- MULFXIS R0, R1, #0xFF80 (-0.5) -> 001000_11_0000_0001_1111111110000000
  -> 0x0001_8000 * 0xFFFF_8000 = 0xFFFF_FFFF_4000_0000 중 [47:16]비트인 0xFFFF_4000 (-0.75)이 R0에 저장,
  [N = 1, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = (result[15:0] != 0)

V = (result[63:48] != (16{result[47]}))

1.10 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			

Division 연산은 입력으로 받은 두 값을 나눠 목적지에 저장하는 명령어이다.
※ 주의: 0으로 나눌 시 결과값으로 0을 리턴한다.

Division 명령어의 사용 형식 및 기능

- DIV rD, rA, rB
  - ■ R[rD] <= (R[rA] ÷ R[rB])[31:0]
- DIVI rD, rA, immB
  - ■ R[rD] <= (R[rA] ÷ signExt(immB))[31:0]
- DIVS rD, rA, rB
  - ■ R[rD] <= (R[rA] ÷ R[rB])[31:0] & update NZCV register
- DIVIS rD, rA, immB
  - ■ R[rD] <= (R[rA] ÷ signExt(immB))[31:0] & update NZCV register

Divisoin 명령어의 사용 예시
나중에 작성할 것

NZCV 플래그 세팅 규칙
N = result[31]
Z = (result == 0)
C = (rBdata == 0)
V = (rAdata == 0x80000000 && rBdata == -1)

1.11 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			

Division Unsigned 연산은 입력으로 받은 두 값을 부호 없이 나눠 목적지에 저장하는 명령어이다.
※ 주의: 0으로 나눌 시 결과값으로 0을 리턴한다.

Division Unsigned 명령어의 사용 형식 및 기능

- DIVU rD, rA, rB
  - ■ R[rD] <= (R[rA] ÷ R[rB])[31:0]
- DIVUI rD, rA, immB
  - ■ R[rD] <= (R[rA] ÷ zeroExt(immB))[31:0]
- DIVUS rD, rA, rB
  - ■ R[rD] <= (R[rA] ÷ R[rB])[31:0] & update NZCV register
- DIVUIS rD, rA, immB
  - ■ R[rD] <= (R[rA] ÷ zeroExt(immB))[31:0] & update NZCV register

Divisoin Unsigned 명령어의 사용 예시
나중에 작성할 것

NZCV 플래그 세팅 규칙
N = result[31]
Z = (result == 0)
C = (rBdata == 0)
V = 0

1.12 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			

Modulo 연산은 입력으로 받은 두 값을 나눠 목적지에 저장하는 명령어이다.
※ 주의: 0으로 나눌 시 결과값으로 0을 리턴한다.

Modulo 명령어의 사용 형식 및 기능

- MOD rD, rA, rB
  - ■ R[rD] <= (R[rA] % R[rB])[31:0]
- MODI rD, rA, immB
  - ■ R[rD] <= (R[rA] % signExt(immB))[31:0]
- MODS rD, rA, rB
  - ■ R[rD] <= (R[rA] % R[rB])[31:0] & update NZCV register
- MODIS rD, rA, immB
  - ■ R[rD] <= (R[rA] % signExt(immB))[31:0] & update NZCV register

Modulo 명령어의 사용 예시
나중에 작성할 것

NZCV 플래그 세팅 규칙
N = result[31]
Z = (result == 0)
C = (rBdata == 0)
V = 0

1.13 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			

Modulo Unsigned 연산은 입력으로 받은 두 값을 나눠 목적지에 저장하는 명령어이다.
※ 주의: 0으로 나눌 시 결과값으로 0을 리턴한다.

Modulo Unsigned 명령어의 사용 형식 및 기능

- MODU rD, rA, rB
  - ■ R[rD] <= (R[rA] % R[rB])[31:0]
- MODUI rD, rA, immB
  - ■ R[rD] <= (R[rA] % zeroExt(immB))[31:0]
- MODUS rD, rA, rB
  - ■ R[rD] <= (R[rA] % R[rB])[31:0] & update NZCV register
- MODUIS rD, rA, immB
  - ■ R[rD] <= (R[rA] % zeroExt(immB))[31:0] & update NZCV register

Modulo Unsigned 명령어의 사용 예시
나중에 작성할 것

NZCV 플래그 세팅 규칙
N = result[31]
Z = (result == 0)
C = (rBdata == 0)
V = 0

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] <= R[rA] << R[rB]
- SHLI rD, rA, immB
  - ■ R[rD] <= R[rA] << zeroExt(immB)
- SHLS rD, rA, rB
  - ■ R[rD] <= R[rA] << R[rB] & update NZCV register
- SHLIS rD, rA, immB
  - ■ R[rD] <= R[rA] << zeroExt(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.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 마지막으로 밀려난 비트

V = 0

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] <= R[rA] >> R[rB]
- ASRI rD, rA, immB
  - ■ R[rD] <= R[rA] >> zeroExt(immB)
- ASRS rD, rA, rB
  - ■ R[rD] <= R[rA] >> R[rB] & update NZCV register
- ASRIS rD, rA, immB
  - ■ R[rD] <= R[rA] >> zeroExt(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.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 마지막으로 밀려난 비트

$V = 0$

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] <= R[rA] >> R[rB]
- LSRI rD, rA, immB
  - ■ R[rD] <= R[rA] >> zeroExt(immB)
- LSRS rD, rA, rB
  - ■ R[rD] <= R[rA] >> R[rB] & update NZCV register
- LSRIS rD, rA, immB
  - ■ R[rD] <= R[rA] >> zeroExt(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.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 마지막으로 밀려난 비트

$V = 0$

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] <= R[rA] ROL R[rB], Shift-out bits are rotated to Left side
- ROLI rD, rA, immB
  - ■ R[rD] <= R[rA] ROL zeroExt(immB), Shift-out bits are rotated to Left side
- ROLS rD, rA, rB
  - ■ R[rD] <= R[rA] ROL R[rB], Shift-out bits are rotated to Left side
    & update NZCV register
- ROLIS rD, rA, immB
  - ■ R[rD] <= R[rA] ROL zeroExt(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 = -105 (0b1001_0111)가 R0에 저장. (8bit 기준)
- ROLIS R0, R1, #1 -> 011011_11_0000_0001_0000000000000001
  -> 7 (0b0000_0111) ROL 1 = 14 (0b0000_1110)가 R0에 저장,
  [N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장. (8bit 기준)

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 마지막으로 회전한 비트

V = 0

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] <= R[rA] ROR R[rB], Shift-out bits are rotated to Right side
- RORI rD, rA, immB
  - ■ R[rD] <= R[rA] ROR zeroExt(immB), Shift-out bits are rotated to Right side
- RORS rD, rA, rB
  - ■ R[rD] <= R[rA] ROR R[rB], Shift-out bits are rotated to Right side
    & update NZCV register
- RORIS rD, rA, immB
  - ■ R[rD] <= R[rA] ROR zeroExt(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 = 94 (0b0101_1110)가 R0에 저장. (8bit 기준)
- RORIS R0, R1, #1 -> 011100_11_0000_0001_0000000000000001
  -> 7 (0b0000_0111) ROR 1 = -125 (0b1000_0011)가 R0에 저장,
  [N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장. (8bit 기준)

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 마지막으로 회전한 비트

V = 0

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] <= R[rA] & R[rB]
- ANDI rD, rA, immB
  - ■ R[rD] <= R[rA] & zeroExt(immB)
- ANDS rD, rA, rB
  - ■ R[rD] <= R[rA] & R[rB] & update NZCV register
- ANDIS rD, rA, immB
  - ■ R[rD] <= R[rA] & zeroExt(immB) & 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.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 0

V = 0

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] <= R[rA] | R[rB]
- ORI rD, rA, immB
  - R[rD] <= R[rA] | zeroExt(immB)
- ORS rD, rA, rB
  - R[rD] <= R[rA] | R[rB] & update NZCV register
- ORIS rD, rA, immB
  - R[rD] <= R[rA] | zeroExt(immB) & 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.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 0

$V = 0$

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] <= R[rA] ^ R[rB]
- XORI rD, rA, immB
  - R[rD] <= R[rA] ^ zeroExt(immB)
- XORS rD, rA, rB
  - R[rD] <= R[rA] ^ R[rB] & update NZCV register
- XORIS rD, rA, immB
  - R[rD] <= R[rA] ^ zeroExt(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) ^ 55 (0b0011_0111) = 90 (0b0101_1010)가 R0에 저장.
- XORIS R0, R1, #227 -> 100011_11_0000_0001_0000000011100011
  -> 109 (0b0110_1101) ^ 227 (0b1110_0011) = 142 (0b1000_1110)가 R0에 저장,
  [N = 0, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 0

$V = 0$

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	
NOTI	100100	1	0	rD		immB					
NOTS	100100	0	1	rD		reserved		rB		reserved	
NOTIS	100100	1	1	rD		immB					

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

Not 명령어의 사용 형식 및 기능

- NOT rD, rB
  - ■ R[rD] <= ~ R[rB]
- NOTI rD, immB
  - ■ R[rD] <= ~ zeroExt(immB)
- NOTS rD, rB
  - ■ R[rD] <= ~ R[rB] & update NZCV register
- NOTIS rD, immB
  - ■ R[rD] <= ~ zeroExt(immB) & 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
  -> ~109 (0b0110_1101) = -110 (0b1001_0010)가 R0에 저장.
- NOTI R0, #F32D6 -> 100100_10_0000_1111_0011_0010_1101_0110
  -> ~0xF32D6 (0b1111_0011_0010_1101_0110)
  = -996055 (0b1111_1111_1111_0000_1100_1101_0010_1001)가 R0에 저장.
- NOTS R0, R2 -> 100100_01_0000_0000_0010_000000000000
  -> ~55 (0b0011_0111) = -56 (0b1100_1000)가 R0에 저장,
  [N = 1, Z = 0, C = 0, V = 0]을 NZCV 레지스터에 저장.

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 0

V = 0

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
  - ■ _ <= R[rA] & R[rB]
- BCHKI rA, immB
  - ■ _ <= R[rA] & zeroExt(immB)
- BCHKS rA, rB
  - ■ _ <= R[rA] & R[rB] & update NZCV register
- BCHKIS rA, immB
  - ■ _ <= R[rA] & zeroExt(immB) & 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)

NZCV 플래그 세팅 규칙

N = result[31]

Z = (result == 0)

C = 0

V = 0

4. 메모리 명령어

4.1 Load

LDR{I | R}	opcode	I	R	23		20	19		16	15		12	11		0
LDR	101000	0	0	rD			rA			immB					
LDRI	101000	1	0	rD			immB								
LDRR	101000	0	1	rD			immB								

Load 연산은 메모리의 특정 주소에 저장된 32비트 값을 읽어와 레지스터에 저장하는 연산이다.
Load 명령어는 Displacement mode, Immediate mode, PC-Relative mode 중 하나로 동작한다.

Load 명령어의 사용 형식 및 기능

- LDR rD, rA, immB
  - ■ R[rD] <= M[R[rA] + signExt(immB)] (Displacement mode)
- LDRI rD, immB
  - ■ R[rD] <= M[zeroExt(immB)] (Immediate mode)
- LDRR rD, immB
  - ■ R[rD] <= M[PC + signExt(immB)] (pc-relative mode)

Load 명령어의 사용 예시

R0 = 0, R1 = 109, PC = 56 로 설정되었고,

M[256] = 93, M[200] = 71, M[419] = 52 가 저장된 상태일 때,

- LDR R0, R1, #310 -> 101000_00_0000_0001_0000000100110110
  -> M[109 + 310] = M[419] = 52가 R0에 저장.
- LDRI R0, #200 -> 101000_10_0000_0000000000011001000
  -> M[200] = 71가 R0에 저장.
- LDRR R0, #200 -> 101000_01_0000_0000000000011001000
  -> M[56 + 200] = M[256] = 93가 R0에 저장.

4.2 Load Byte

LDRB{I | R}	opcode	I	R	23	20	19	16	15	12	11	0
LDRB	101001	0	0	rD		rA		immB			
LDRBI	101001	1	0	rD		immB					
LDRBR	101001	0	1	rD		immB					

Load Byte 연산은 메모리의 특정 주소에 저장된 8비트 값을 읽어와 0으로 확장해 레지스터에 저장하는 연산이다. Load Byte 명령어는 Displacement mode, Immediate mode, PC-Relative mode 중 하나로 동작한다.

Load Byte 명령어의 사용 형식 및 기능

- LDRB rD, rA, immB
  - ■ R[rD] <= M[R[rA] + signExt(immB)] (Displacement mode)
- LDRBI rD, immB
  - ■ R[rD] <= M[zeroExt(immB)] (Immediate mode)
- LDRBR rD, immB
  - ■ R[rD] <= M[PC + signExt(immB)] (pc-relative mode)

Load Byte 명령어의 사용 예시

R0 = 0, R1 = 109, PC = 56 로 설정되었고,

M[256] = 0x93, M[200] = 0x71, M[419] = 0xD2 가 저장된 상태일 때,

- LDRB R0, R1, #310 -> 101001_00_0000_0001_0000000100110110
  -> M[109 + 310] = M[419] = 0xD2 -> 0x000000D2가 R0에 저장.
- LDRBI R0, #200 -> 101001_10_0000_0000000000011001000
  -> M[200] = 0x71 -> 0x00000071가 R0에 저장.
- LDRBR R0, #200 -> 101001_01_0000_0000000000011001000
  -> M[56 + 200] = M[256] = 0x93 -> 0x00000093가 R0에 저장.

4.3 Load Signed Byte

LDRSB{I | R}	opcode	I	R	23		20	19		16	15		12	11		0
LDRSB	101010	0	0	rD			rA			immB					
LDRSBI	101010	1	0	rD			immB								
LDRSBR	101010	0	1	rD			immB								

Load Signed Byte 연산은 메모리의 특정 주소에 저장된 8비트 값을 읽어와 부호 확장해 레지스터에 저장하는 연산이다. Load Signed Byte 명령어는 Displacement mode, Immediate mode, PC-Relative mode 중 하나로 동작한다.

Load Signed Byte 명령어의 사용 형식 및 기능

- LDRSB rD, rA, immB
  - ■ R[rD] <= M[R[rA] + signExt(immB)] (Displacement mode)
- LDRSBI rD, immB
  - ■ R[rD] <= M[zeroExt(immB)] (Immediate mode)
- LDRSBR rD, immB
  - ■ R[rD] <= M[PC + signExt(immB)] (pc-relative mode)

Load Signed Byte 명령어의 사용 예시

R0 = 0, R1 = 109, PC = 56 로 설정되었고,

M[256] = 0x93, M[200] = 0x71, M[419] = 0xD2 가 저장된 상태일 때,

- LDRSB R0, R1, #310 -> 101010_00_0000_0001_0000000100110110
  -> M[109 + 310] = M[419] = 0xD2 -> 0xFFFFFFD2가 R0에 저장.
- LDRSBI R0, #200 -> 101010_10_0000_0000000000011001000
  -> M[200] = 0x71 -> 0x00000071가 R0에 저장.
- LDRSBR R0, #200 -> 101010_01_0000_0000000000011001000
  -> M[56 + 200] = M[256] = 0x93 -> 0xFFFFFF93가 R0에 저장.

4.4 Load Halfword

LDRH{I | R}	opcode	I	R	23	20	19	16	15	12	11	0
LDRH	101011	0	0	rD		rA		immB			
LDRHI	101011	1	0	rD		immB					
LDRHR	101011	0	1	rD		immB					

Load Halfword 연산은 메모리의 특정 주소에 저장된 16비트 값을 읽어와 0으로 확장해 레지스터에 저장하는 연산이다. Load Halfword 명령어는 Displacement mode, Immediate mode, PC-Relative mode 중 하나로 동작한다.

Load Halfword 명령어의 사용 형식 및 기능

- LDRH rD, rA, immB
  - ■ R[rD] <= M[R[rA] + signExt(immB)] (Displacement mode)
- LDRHI rD, immB
  - ■ R[rD] <= M[zeroExt(immB)] (Immediate mode)
- LDRHR rD, immB
  - ■ R[rD] <= M[PC + signExt(immB)] (pc-relative mode)

Load Halfword 명령어의 사용 예시

R0 = 0, R1 = 109, PC = 56 로 설정되었고,

M[256] = 0x93E5, M[200] = 0x71A9, M[419] = 0xD215 가 저장된 상태일 때,

- LDRH R0, R1, #310 -> 101011_00_0000_0001_0000000100110110
  -> M[109 + 310] = M[419] = 0xD215 -> 0x0000D215가 R0에 저장.
- LDRHI R0, #200 -> 101011_10_0000_0000000000011001000
  -> M[200] = 0x71A9 -> 0x000071A9가 R0에 저장.
- LDRHR R0, #200 -> 101011_01_0000_0000000000011001000
  -> M[56 + 200] = M[256] = 0x93E5 -> 0x000093E5가 R0에 저장.

4.5 Load Signed Halfword

LDRSH{I | R}	opcode	I	R	23	20	19	16	15	12	11	0
LDRSH	101100	0	0	rD		rA		immB			
LDRSHI	101100	1	0	rD		immB					
LDRSHR	101100	0	1	rD		immB					

Load Signed Halfword 연산은 메모리의 특정 주소에 저장된 16비트 값을 읽어와 부호 확장해 레지스터에 저장하는 연산이다. Load Signed Halfword 명령어는 Displacement mode, Immediate mode, PC-Relative mode 중 하나로 동작한다.

Load Signed Halfword 명령어의 사용 형식 및 기능

- LDRSH rD, rA, immB
  - ■ R[rD] <= M[R[rA] + signExt(immB)] (Displacement mode)
- LDRSHI rD, immB
  - ■ R[rD] <= M[zeroExt(immB)] (Immediate mode)
- LDRSHR rD, immB
  - ■ R[rD] <= M[PC + signExt(immB)] (pc-relative mode)

Load Signed Halfword 명령어의 사용 예시

R0 = 0, R1 = 109, PC = 56 로 설정되었고,

M[256] = 0x93E5, M[200] = 0x71A9, M[419] = 0xD215 가 저장된 상태일 때,

- LDRSH R0, R1, #310 -> 101100_00_0000_0001_0000000100110110
  -> M[109 + 310] = M[419] = 0xD215 -> 0xFFFFD215가 R0에 저장.
- LDRSHI R0, #200 -> 101100_10_0000_0000000000011001000
  -> M[200] = 0x71A9 -> 0x000071A9가 R0에 저장.
- LDRSHR R0, #200 -> 101100_01_0000_0000000000011001000
  -> M[56 + 200] = M[256] = 0x93E5 -> 0xFFFF93E5가 R0에 저장.

4.6 Store

STR{I | R}	opcode	I	R	23	20	19	16	15	12	11	0
STR	101101	0	0	immB		rA		rB		immB	
STRI	101101	1	0	immB				rB		immB	
STRR	101101	0	1	immB				rB		immB	

Store 연산은 레지스터에 저장된 32비트 값을 메모리의 특정 주소에 저장하는 연산이다. Store 명령어는 Displacement mode, Immediate mode, PC-Relative mode 중 하나로 동작한다.
Displacement mode의 즉시값은 {[23:20], [11:0]}으로 받는다. Immediate mode, PC-Relative mode의 즉시값은 {[23:16], [11:0]}으로 받는다.

Store 명령어의 사용 형식 및 기능

- STR rB, rA, immB
  - ■ M[R[rA] + signExt(immB)] <= R[rB] (Displacement mode)
- STRI rB, immB
  - ■ M[zeroExt(immB)] <= R[rB] (Immediate mode)
- STRR rB, immB
  - ■ M[PC + signExt(immB)] <= R[rB] (pc-relative mode)

Store 명령어의 사용 예시

R1 = 109, R2 = 87, R3 = 0x12345678, PC = 56 로 설정된 상태일 때,

- STR R2, R1, #310 -> 101101_00_0000_0001_0010_000100110110
  -> R[2] = 87이 M[109 + 310] = M[419]에 저장.
- STRI R2, #200 -> 101101_10_00000000_0010_000011001000
  -> R[2] = 87이 M[200]에 저장.
- STRR R3, #200 -> 101101_01_00000000_0011_000011001000
  -> R[3] = 0x12345678이 M[56 + 200]= M[256]에 저장.

4.7 Store Byte

STRB{I | R}	opcode	I	R	23	20	19	16	15	12	11	0
STRB	101110	0	0	immB		rA		rB		immB	
STRBI	101110	1	0	immB				rB		immB	
STRBR	101110	0	1	immB				rB		immB	

Store Byte 연산은 레지스터에 저장된 32비트 값의 하위 8비트를 메모리의 특정 주소에 저장하는 연산이다. Store Byte 명령어는 Displacement mode, Immediate mode, PC-Relative mode 중 하나로 동작한다.
Displacement mode의 즉시값은 {[23:20], [11:0]}으로 받는다. Immediate mode, PC-Relative mode의 즉시값은 {[23:16], [11:0]}으로 받는다.

Store Byte 명령어의 사용 형식 및 기능

- STRB rB, rA, immB
  - ■ M[R[rA] + signExt(immB)] <= R[rB] (Displacement mode)
- STRBI rB, immB
  - ■ M[zeroExt(immB)] <= R[rB] (Immediate mode)
- STRBR rB, immB
  - ■ M[PC + signExt(immB)] <= R[rB] (pc-relative mode)

Store Byte 명령어의 사용 예시

R1 = 109, R2 = 87, R3 = 0x12345678, PC = 56 로 설정된 상태일 때,

- STRB R2, R1, #310 -> 101110_00_0000_0001_0010_000100110110
  -> R[2] = 87이 M[109 + 310] = M[419]에 저장.
- STRBI R2, #200 -> 101110_10_00000000_0010_000011001000
  -> R[2] = 87이 M[200]에 저장.
- STRBR R3, #200 -> 101110_01_00000000_0011_000011001000
  -> R[3] = 0x12345678 -> 0x78이 M[56 + 200]= M[256]에 저장.

4.8 Store Halfword

STRH{I | R}	opcode	I	R	23	20	19	16	15	12	11	0
STRH	101111	0	0	immB		rA		rB		immB	
STRHI	101111	1	0	immB				rB		immB	
STRHR	101111	0	1	immB				rB		immB	

Store Halfword 연산은 레지스터에 저장된 32비트 값의 하위 16비트를 메모리의 특정 주소에 저장하는 연산이다. Store Halfword 명령어는 Displacement mode, Immediate mode, PC-Relative mode 중 하나로 동작한다.
Displacement mode의 즉시값은 {[23:20], [11:0]}으로 받는다. Immediate mode, PC-Relative mode의 즉시값은 {[23:16], [11:0]}으로 받는다.

Store Halfword 명령어의 사용 형식 및 기능

- STRH rB, rA, immB
  - ■ M[R[rA] + signExt(immB)] <= R[rB] (Displacement mode)
- STRHI rB, immB
  - ■ M[zeroExt(immB)] <= R[rB] (Immediate mode)
- STRHR rB, immB
  - ■ M[PC + signExt(immB)] <= R[rB] (pc-relative mode)

Store Halfword 명령어의 사용 예시

R1 = 109, R2 = 87, R3 = 0x12345678, PC = 56 로 설정된 상태일 때,

- STRH R2, R1, #310 -> 101111_00_0000_0001_0010_000100110110
  -> R[2] = 87이 M[109 + 310] = M[419]에 저장.
- STRHI R2, #200 -> 101111_10_00000000_0010_000011001000
  -> R[2] = 87이 M[200]에 저장.
- STRHR R3, #200 -> 101111_01_00000000_0011_000011001000
  -> R[3] = 0x12345678 -> 0x5678이 M[56 + 200]= M[256]에 저장.

5. 컨트롤 명령어

5.1 Branch

B{cond}	opcode	25		4	3		0
B	110010	immB			cond		

Branch 연산은 조건에 따라 프로그램의 흐름을 PC-relative 주소로 변경하는 연산으로, 주로 for, while, else-if 등에서 발생하는 단거리 분기에 사용된다. Condition의 종류로는 ARM과 동일한 15개를 가진다.

Branch 명령어의 사용 형식 및 기능

- B{cond} immB
  - If (eval (NZCV reg, cond)) then {PC <= PC + signExt(immB)}

Branch 명령어의 사용 예시

PC = 56, NZCV reg = [0, 1, 0, 0] 로 설정된 상태일 때,

- BEQ #-36 -> 110010_11111111111111111011100_0000
  -> EQ: Z bit == 1. 조건을 만족한다.
  -> PC + (-36) = 20이 PC에 저장.
- BNE #-36 -> 110010_11111111111111111011100_0001
  -> NE: Z bit == 0. 조건을 만족하지 않는다.
  -> 기본 PC 증가 동작인 PC + 4 = 60이 PC에 저장.

코드	접미사	의미	플래그 조건
0000	EQ	Equal	Z == 1
0001	NE	Not Equal	Z == 0
0010	CS/HS	Carry Set/Unsigned Higher or Same	C == 1
0011	CC/LO	Carry Clear/Unsigned Lower	C == 0
0100	MI	Minus	N == 1
0101	PL	Plus	N == 0
0110	VS	Overflow Set	V == 1
0111	VC	Overflow Clear	V == 0
1000	HI	Unsigned Higher	C == 1 AND Z == 0
1001	LS	Unsigned Lower or Same	C == 0 AND Z == 1
1010	GE	Signed Greater or Equal	N == V
1011	LT	Signed Less Than	N != V
1100	GT	Signed Greater Than	Z == 0 AND N == V
1101	LE	Signed Less or Equal	Z == 1 AND N != V
1110	AL	Always	1

5.2 Jump

JMP{I}{L}	opcode	I	L	23	20	19	16	15	12	11	0
JMP	110011	0	0	reserved		rA		immB			
JMPI	110011	1	0	immB							
JMPL	110011	0	1	rD		rA		immB			
JMPIL	110011	1	1	rD		immB					

Jump 연산은 조건 없이 프로그램의 흐름을 변경하는 연산으로, 주로 함수 호출 등에서 발생하는 장거리 분기에 사용된다. Jump 명령어는 Return mode, Immediate mode, Register-Link mode, Immediate-Link mode 중 하나로 동작한다.
※ 주의: Return mode, Register-Link mode에서의 즉시값은 20비트 이상의 즉시값으로 점프하는 경우 MOVH와 함께 사용된다. 따라서 레지스터에 저장된 주소로 점프하기 위해서는 즉시값을 0으로 설정해야 한다.

Jump 명령어의 사용 형식 및 기능

- JMP rA, immB
  - ■ PC <= R[rA] + zeroExt(immB)
- JMPI immB
  - ■ PC <= zeroExt(immB)
- JMPL rD, rA, immB
  - ■ R[rD] <= PC + 4

    PC <= R[rA] + zeroExt(immB)
- JMPIL rD, immB
  - ■ R[rD] <= PC + 4

    PC <= zeroExt(immB)

Jump 명령어의 사용 예시

R6 = 0, PC = 56로 설정된 상태일 때,

- JMPIL R6, #72 -> 110011_11_0110_0000000000000001001000

  -> 56 + 4 = 60을 R6에 저장.

  -> 72를 PC에 저장.
- JMP R6, #0 -> 110011_00_0000_0000_0110_000000000000

  -> 60을 PC에 저장. (순차적으로 실행한 경우)

6. 시스템 명령어

추가 학습 후 OS 개발 전 작성예정