ARM® 및 Thumb®-2 명령어 세트
빠른 참조 카드

표기 규칙 줄이			
Rm {, <opsh>}	**레지스터, 선택적으로 상수에 의해 시프트됨** 표 참조	<reglist>	콤마로 구분되고 중괄호 {}로 묶인 레지스터 목록
<Operand2>	**유연한 피연산자 2** 표 참조. 시프트 및 회전은 Operand2의 일부로만 사용할 수 있음	<reglist-PC>	PC를 포함하지 않는 <reglist>와 같음
<fields>	**PSR 필드** 표 참조	<reglist+PC>	PC를 포함하는 <reglist>와 같음
<PSR>	APSR(응용 프로그램 상태 레지스터), CPSR(현재 프로세서 상태 레지스터) 또는 SPSR(저장된 프로세서 상태 레지스터)	<flags>	nzcvq(ALU 플래그 PSR[31:27]) 또는 g(SIMD GE 플래그 PSR[19:16])
C*, V*	아키텍처 버전 4 이하에서 플래그를 예상할 수 없음. 아키텍처 버전 5 이상에서 변경되지 않음	§	**ARM 아키텍처 버전** 표 참조
<Rs|sh>	Rs 또는 즉치 시프트 값일 수 있음. 각 시프트 유형에 허용되는 값과 동일함	+/-	+ 또는 –. +는 생략할 수 있음
	레지스터, 선택적으로 상수에 의해 시프트됨 표에 표시된 값과 같음	<iflags>	인터럽트 플래그. a, i, f(어보트, 인터럽트, 고속 인터럽트) 중 하나 이상
x,y	하프 레지스터 [15:0]을 의미하는 B 또는 [31:16]을 의미하는 T	<p_mode>	**프로세서 모드** 표 참조
<imm8m>	ARM: 8비트 값을 짝수 비트 수만큼 오른쪽으로 회전하여 얻은 32비트 상수	SPm	<p_mode>로 지정된 프로세서 모드의 SP
	Thumb: 8비트 값을 임의의 비트 수만큼 왼쪽으로 시프트하여 얻은 32비트 상수 또는	<lsb>	비트 필드의 최하위 비트
	0xXYXYXYXY, 0x00XY00XY 또는 0xXY00XY00 중 한 형식의 비트 패턴	<width>	비트 필드의 너비. <width> + <lsb>는 <= 32여야 함
<prefix>	**병렬 명령어의 접두사** 표 참조	{X}	X가 있으면 RsX는 Rs가 회전된 16비트이고 그렇지 않으면 RsX는 Rs임
{IA|IB|DA|DB}	증가 후, 증가 전, 감소 후 또는 감소 전	{!}	!가 있는 경우(사전 인덱싱됨) 데이터 전송 후 기본 레지스터 업데이트
	IB 및 DA는 Thumb 상태에서는 사용할 수 없음. 생략할 경우 기본적으로 IA로 설정됨	{S}	S가 있는 경우 조건 플래그 업데이트
<size>	B, SB, H 또는 SH. 각각 바이트, 부호 있는 바이트, 하프워드 및 부호 있는 하프워드를 의미함	{T}	T가 있는 경우 사용자 모드 권한
	SB 및 SH는 STR 명령어에서는 사용할 수 없음	{R}	R이 있으면 결과를 근사한 값으로 반올림하고 그렇지 않으면 결과를 자름

연산			어셈블러	S 업데이트	동작	메모
더하기	더하기		ADD{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn + Operand2	N
	carry 포함		ADC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn + Operand2 + Carry	N
	확장	T2	ADD Rd, Rn, #<imm12>		Rd := Rn + imm12, imm12 범위 0 ~ 4095	T, P
	포화 {doubled}	5E	Q{D}ADD Rd, Rm, Rn		Rd := SAT(Rm + Rn) doubled: Rd := SAT(Rm + SAT(Rn * 2))	Q
주소	PC에서 상대 주소		ADR Rd, <label>		Rd := <label>, 현재 명령어에서 <label> 범위의 경우 참고 정보 L 참조	N, L
빼기	빼기		SUB{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn – Operand2	N
	carry 포함		SBC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn – Operand2 – NOT(Carry)	N
	확장	T2	SUB Rd, Rn, #<imm12>		Rd := Rn – imm12, imm12 범위 0 ~ 4095	T, P
	역방향 빼기		RSB{S} Rd, Rn, <Operand2>	N Z C V	Rd := Operand2 – Rn	N
	carry를 포함한 역방향 빼기		RSC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Operand2 – Rn – NOT(Carry)	A
	포화 {doubled}	5E	Q{D}SUB Rd, Rm, Rn		Rd := SAT(Rm – Rn) doubled: Rd := SAT(Rm – SAT(Rn * 2))	Q
	스택이 없으면 예외 반환		SUBS PC, LR, #<imm8>	N Z C V	PC = LR – imm8, CPSR = SPSR(현재 모드), imm8 범위 0 ~ 255.	
병렬 산술	하프워드 더하기	6	<prefix>ADD16 Rd, Rn, Rm		Rd[31:16] := Rn[31:16] + Rm[31:16], Rd[15:0] := Rn[15:0] + Rm[15:0]	G
	하프워드 빼기	6	<prefix>SUB16 Rd, Rn, Rm		Rd[31:16] := Rn[31:16] – Rm[31:16], Rd[15:0] := Rn[15:0] – Rm[15:0]	G
	바이트 더하기	6	<prefix>ADD8 Rd, Rn, Rm		Rd[31:24] := Rn[31:24] + Rm[31:24], Rd[23:16] := Rn[23:16] + Rm[23:16], Rd[15:8] := Rn[15:8] + Rm[15:8], Rd[7:0] := Rn[7:0] + Rm[7:0]	G
	바이트 빼기	6	<prefix>SUB8 Rd, Rn, Rm		Rd[31:24] := Rn[31:24] – Rm[31:24], Rd[23:16] := Rn[23:16] – Rm[23:16], Rd[15:8] := Rn[15:8] – Rm[15:8], Rd[7:0] := Rn[7:0] – Rm[7:0]	G
	하프워드 전환, 더하기, 빼기	6	<prefix>ASX Rd, Rn, Rm		Rd[31:16] := Rn[31:16] + Rm[15:0], Rd[15:0] := Rn[15:0] – Rm[31:16]	G
	하프워드 전환, 빼기, 더하기	6	<prefix>SAX Rd, Rn, Rm		Rd[31:16] := Rn[31:16] – Rm[15:0], Rd[15:0] := Rn[15:0] + Rm[31:16]	G
	부호 없는 절대차의 합	6	USAD8 Rd, Rm, Rs		Rd := Abs(Rm[31:24] – Rs[31:24]) + Abs(Rm[23:16] – Rs[23:16]) + Abs(Rm[15:8] – Rs[15:8]) + Abs(Rm[7:0] – Rs[7:0])	
	및 누산	6	USADA8 Rd, Rm, Rs, Rn		Rd := Rn + Abs(Rm[31:24] – Rs[31:24]) + Abs(Rm[23:16] – Rs[23:16]) + Abs(Rm[15:8] – Rs[15:8]) + Abs(Rm[7:0] – Rs[7:0])	
포화	워드에 대한 부호 있는 포화, 오른쪽 시프트	6	SSAT Rd, #<sat>, Rm{, ASR <sh>}		Rd := SignedSat((Rm ASR sh), sat). <sat> 범위 1-32, <sh> 범위 1-31	Q, R
	워드에 대한 부호 있는 포화, 왼쪽 시프트	6	SSAT Rd, #<sat>, Rm{, LSL <sh>}		Rd := SignedSat((Rm LSL sh), sat). <sat> 범위 1 ~ 32, <sh> 범위 0 ~ 31	Q
	두 하프워드에 대한 부호 있는 포화	6	SSAT16 Rd, #<sat>, Rm		Rd[31:16] := SignedSat(Rm[31:16], sat), Rd[15:0] := SignedSat(Rm[15:0], sat). <sat> 범위 1-16	Q
	워드에 대한 부호 없는 포화, 오른쪽 시프트	6	USAT Rd, #<sat>, Rm{, ASR <sh>}		Rd := UnsignedSat((Rm ASR sh), sat). <sat> 범위 0 ~ 31, <sh> 범위 1 ~ 31	Q, R
	워드에 대한 부호 없는 포화, 왼쪽 시프트	6	USAT Rd, #<sat>, Rm{, LSL <sh>}		Rd := UnsignedSat((Rm LSL sh), sat). <sat> 범위 0 ~ 31, <sh> 범위 0 ~ 31	Q
	두 하프워드에 대한 부호 없는 포화	6	USAT16 Rd, #<sat>, Rm		Rd[31:16] := UnsignedSat(Rm[31:16], sat), Rd[15:0] := UnsignedSat(Rm[15:0], sat). <sat> 범위 0-15	Q

ARM 및 Thumb-2 명령어 세트
빠른 참조 카드

연산			어셈블러	S 업데이트	동작	메모
곱하기	곱하기		`MUL{S} Rd, Rm, Rs`	N Z C*	Rd := (Rm * Rs)[31:0] (Rs가 Rd인 경우 S는 Thumb-2에 사용할 수 있음)	N, S
	및 누산		`MLA{S} Rd, Rm, Rs, Rn`	N Z C*	Rd := (Rn + (Rm * Rs))[31:0]	S
	및 빼기	T2	`MLS Rd, Rm, Rs, Rn`		Rd := (Rn – (Rm * Rs))[31:0]	
	부호 없는 long		`UMULL{S} RdLo, RdHi, Rm, Rs`	N Z C* V*	RdHi,RdLo := unsigned(Rm * Rs)	S
	long에 대한 부호 없는 누산		`UMLAL{S} RdLo, RdHi, Rm, Rs`	N Z C* V*	RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)	S
	long에 대한 부호 없는 이중 누산	6	`UMAAL RdLo, RdHi, Rm, Rs`		RdHi,RdLo := unsigned(RdHi + RdLo + Rm * Rs)	
	long에 대한 부호 있는 곱하기		`SMULL{S} RdLo, RdHi, Rm, Rs`	N Z C* V*	RdHi,RdLo := signed(Rm * Rs)	S
	및 long 누산		`SMLAL{S} RdLo, RdHi, Rm, Rs`	N Z C* V*	RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)	S
	16 * 16비트	5E	`SMULxy Rd, Rm, Rs`		Rd := Rm[x] * Rs[y]	
	32 * 16비트	5E	`SMULWy Rd, Rm, Rs`		Rd := (Rm * Rs[y])[47:16]	
	16 * 16비트 및 누산	5E	`SMLAxy Rd, Rm, Rs, Rn`		Rd := Rn + Rm[x] * Rs[y]	Q
	32 * 16비트 및 누산	5E	`SMLAWy Rd, Rm, Rs, Rn`		Rd := Rn + (Rm * Rs[y])[47:16]	Q
	16 * 16비트 및 long 누산	5E	`SMLALxy RdLo, RdHi, Rm, Rs`		RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]	
	이중 부호 있는 곱하기, 더하기	6	`SMUAD{X} Rd, Rm, Rs`		Rd := Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]	Q
	및 누산	6	`SMLAD{X} Rd, Rm, Rs, Rn`		Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]	Q
	및 long 누산	6	`SMLALD{X} RdLo, RdHi, Rm, Rs`		RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]	
	이중 부호 있는 곱하기, 빼기	6	`SMUSD{X} Rd, Rm, Rs`		Rd := Rm[15:0] * RsX[15:0] – Rm[31:16] * RsX[31:16]	Q
	및 누산	6	`SMLSD{X} Rd, Rm, Rs, Rn`		Rd := Rn + Rm[15:0] * RsX[15:0] – Rm[31:16] * RsX[31:16]	Q
	및 long 누산	6	`SMLSLD{X} RdLo, RdHi, Rm, Rs`		RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] – Rm[31:16] * RsX[31:16]	
	부호 있는 상위 워드 곱하기	6	`SMMUL{R} Rd, Rm, Rs`		Rd := (Rm * Rs)[63:32]	
	및 누산	6	`SMMLA{R} Rd, Rm, Rs, Rn`		Rd := Rn + (Rm * Rs)[63:32]	
	및 빼기	6	`SMMLS{R} Rd, Rm, Rs, Rn`		Rd := Rn – (Rm * Rs)[63:32]	
	내부 40비트 누산으로 곱하기	XS	`MIA Ac, Rm, Rs`		Ac := Ac + Rm * Rs	
	패킹된 하프워드	XS	`MIAPH Ac, Rm, Rs`		Ac := Ac + Rm[15:0] * Rs[15:0] + Rm[31:16] * Rs[31:16]	
	하프워드	XS	`MIAxy Ac, Rm, Rs`		Ac := Ac + Rm[x] * Rs[y]	
나누기	부호 있거나 부호 없음	RM	`<op> Rd, Rn, Rm`		Rd := Rn / Rm <op> is `SDIV`(부호 있음) 또는 `UDIV`(부호 없음)	T
데이터 이동	데이터		`MOV{S} Rd, <Operand2>`	N Z C	Rd := Operand2 Shift 명령어 참조	N
	이동 안 함		`MVN{S} Rd, <Operand2>`	N Z C	Rd := 0xFFFFFFFF EOR Operand2	N
	위쪽	T2	`MOVT Rd, #<imm16>`		Rd[31:16] := imm16, Rd[15:0] 영향 없음, imm16 범위 0 ~ 65535	
	확장	T2	`MOV Rd, #<imm16>`		Rd[15:0] := imm16, Rd[31:16] = 0, imm16 범위 0 ~ 65535	
	40비트 누산기에서 레지스터로	XS	`MRA RdLo, RdHi, Ac`		RdLo := Ac[31:0], RdHi := Ac[39:32]	
	레지스터에서 40비트 덧셈기로	XS	`MAR Ac, RdLo, RdHi`		Ac[31:0] := RdLo, Ac[39:32] := RdHi[7:0]	
시프트	오른쪽으로 산술 시프트		`ASR{S} Rd, Rm, <Rs|sh>`	N Z C	Rd := ASR(Rm, Rs|sh) `MOV{S} Rd, Rm, ASR <Rs|sh>`와 같음	N
	왼쪽으로 논리 시프트		`LSL{S} Rd, Rm, <Rs|sh>`	N Z C	Rd := LSL(Rm, Rs|sh) `MOV{S} Rd, Rm, LSL <Rs|sh>`와 같음	N
	오른쪽으로 논리 시프트		`LSR{S} Rd, Rm, <Rs|sh>`	N Z C	Rd := LSR(Rm, Rs|sh) `MOV{S} Rd, Rm, LSR <Rs|sh>`와 같음	N
	오른쪽으로 회전		`ROR{S} Rd, Rm, <Rs|sh>`	N Z C	Rd := ROR(Rm, Rs|sh) `MOV{S} Rd, Rm, ROR <Rs|sh>`와 같음	N
	확장을 포함하여 오른쪽으로 회전		`RRX{S} Rd, Rm`	N Z C	Rd := RRX(Rm) `MOV{S} Rd, Rm, RRX`와 같음	
선행 0 수 계산		5	`CLZ Rd, Rm`		Rd := Rm의 선행 0 수	
비교	비교		`CMP Rn, <Operand2>`	N Z C V	Rn – Operand2에서 CPSR 플래그 업데이트	N
	부정		`CMN Rn, <Operand2>`	N Z C V	Rn + Operand2에서 CPSR 플래그 업데이트	N
논리	테스트		`TST Rn, <Operand2>`	N Z C	Rn AND Operand2에서 CPSR 플래그 업데이트	N
	동등 테스트		`TEQ Rn, <Operand2>`	N Z C	Rn EOR Operand2에서 CPSR 플래그 업데이트	
	AND		`AND{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn AND Operand2	N
	EOR		`EOR{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn EOR Operand2	N
	ORR		`ORR{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn OR Operand2	N
	ORN	T2	`ORN{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn OR NOT Operand2	T
	비트 지우기		`BIC{S} Rd, Rn, <Operand2>`	N Z C	Rd := Rn AND NOT Operand2	N

ARM 및 Thumb-2 명령어 세트
빠른 참조 카드

연산		§	어셈블러	동작	메모
비트 필드	비트 필드 지우기	T2	`BFC Rd, #<lsb>, #<width>`	Rd[(width+lsb–1):lsb] := 0, Rd의 다른 비트는 영향을 받지 않음	
	비트 필드 삽입	T2	`BFI Rd, Rn, #<lsb>, #<width>`	Rd[(width+lsb–1):lsb] := Rn[(width-1):0], Rd의 다른 비트는 영향을 받지 않음	
	부호 있는 비트 필드 추출	T2	`SBFX Rd, Rn, #<lsb>, #<width>`	Rd[(width–1):0] = Rn[(width+lsb–1):lsb], Rd[31:width] = Replicate(Rn[width+lsb–1])	
	부호 없는 비트 필드 추출	T2	`UBFX Rd, Rn, #<lsb>, #<width>`	Rd[(width–1):0] = Rn[(width+lsb–1):lsb], Rd[31:width] = Replicate(0)	
패킹	하프워드 맨 아래 + 맨 위 패킹	6	`PKHBT Rd, Rn, Rm{, LSL #<sh>}`	Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16]. sh 0-31	
	하프워드 맨 위 + 맨 아래 패킹	6	`PKHTB Rd, Rn, Rm{, ASR #<sh>}`	Rd[31:16] := Rn[31:16], Rd[15:0] := (Rm ASR sh)[15:0]. sh 1-32.	
부호 있는 확장	하프워드를 워드로	6	`SXTH Rd, Rm{, ROR #<sh>}`	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3	N
	2바이트를 하프워드로	6	`SXTB16 Rd, Rm{, ROR #<sh>}`	Rd[31:16] := SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3	
	바이트를 워드로	6	`SXTB Rd, Rm{, ROR #<sh>}`	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3	N
부호 없는 확장	하프워드를 워드로	6	`UXTH Rd, Rm{, ROR #<sh>}`	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3	N
	2바이트를 하프워드로	6	`UXTB16 Rd, Rm{, ROR #<sh>}`	Rd[31:16] := ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3	
	바이트를 워드로	6	`UXTB Rd, Rm{, ROR #<sh>}`	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3	N
더하기 포함 부호 있는 확장	하프워드를 워드로, 더하기	6	`SXTAH Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3	
	2바이트를 하프워드로, 더하기	6	`SXTAB16 Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:16] := Rn[31:16] + SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := Rn[15:0] + SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3	
	바이트를 워드로, 더하기	6	`SXTAB Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3	
더하기 포함 부호 없는 확장	하프워드를 워드로, 더하기	6	`UXTAH Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3	
	2바이트를 하프워드로, 더하기	6	`UXTAB16 Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:16] := Rn[31:16] + ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := Rn[15:0] + ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3	
	바이트를 워드로, 더하기	6	`UXTAB Rd, Rn, Rm{, ROR #<sh>}`	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3	
역방향	워드의 비트	T2	`RBIT Rd, Rm`	For (i = 0; i < 32; i++) : Rd[i] = Rm[31– i]	
	워드의 바이트	6	`REV Rd, Rm`	Rd[31:24] := Rm[7:0], Rd[23:16] := Rm[15:8], Rd[15:8] := Rm[23:16], Rd[7:0] := Rm[31:24]	N
	두 하프워드의 바이트	6	`REV16 Rd, Rm`	Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:24] := Rm[23:16], Rd[23:16] := Rm[31:24]	N
	하위 하프워드의 바이트, 부호 확장	6	`REVSH Rd, Rm`	Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:16] := Rm[7] * &FFFF	N
선택	바이트 선택	6	`SEL Rd, Rn, Rm`	Rd[7:0] := Rn[7:0] if GE[0] = 1, else Rd[7:0] := Rm[7:0] GE[1], GE[2], GE[3]일 경우 마찬가지로 비트[15:8], [23:16], [31:24]가 선택됨	
If-Then	If-Then	T2	`IT{pattern} {cond}`	pattern에 따라 다음과 같은 최대 네 개의 명령어 조건을 만듦. pattern은 최대 세 개의 문자로 구성된 문자열임. 각 문자는 T(Then)이거나 E(Else)일 수 있음 IT 다음의 첫 번째 명령어에는 cond 조건이 적용됨. 다음 명령어는 해당 문자가 T인 경우 cond 조건이 적용되고, 해당 문자가 E인 경우 cond의 반대 조건이 적용됨 사용 가능한 조건 코드는 **조건 필드** 표 참조	T, U
분기	분기		`B <label>`	PC := label. label은 이 명령어 ±32MB (T2: 16MB, T: –252 - +256B)	N, B
	링크 포함		`BL <label>`	LR := 다음 명령어의 주소, PC := label. label은 이 명령어 32MB (T2: 16MB)	
	및 전환	4T	`BX Rm`	PC := Rm. Rm[0]이 1이면 타겟은 Thumb이고, Rm[0]이 0이면 타겟은 ARM	N
	링크 및 전환 포함(1)	5T	`BLX <label>`	LR := 다음 명령어의 주소, PC := label, 변경 명령어 세트. label은 이 명령어 ±32MB(T2: 16MB)	C
	링크 및 전환 포함(2)	5	`BLX Rm`	LR := 다음 명령어의 주소, PC := Rm[31:1] Rm[0]이 1일 경우 Thumb으로 변경, Rm[0]이 0일 경우 ARM으로 변경	N
	및 Jazelle 상태로 변경	5J	`BXJ Rm`	가능한 경우 Jazelle 상태로 변경	
	비교, 0인 경우(0이 아닌 경우) 분기	T2	`CB{N}Z Rn,<label>`	If Rn {== or !=} 0 then PC := label. label은 이 명령어 + 4-130	N,T,U
	테이블 분기 바이트	T2	`TBB [Rn, Rm]`	PC = PC + ZeroExtend(Memory(Rn + Rm, 1) << 1). 분기 범위 4 ~ 512. Rn은 PC일 수 있음	T, U
	하프워드 테이블 분기	T2	`TBH [Rn, Rm, LSL #1]`	PC = PC + ZeroExtend(Memory(Rn + Rm << 1, 2) << 1). 분기 범위 4 ~ 131072. Rn은 PC일 수 있음	T, U
PSR 간 이동	PSR에서 레지스터로		`MRS Rd, <PSR>`	Rd := PSR	
	레지스터 플래그에서 APSR 플래그로		`MSR APSR_<flags>, Rm`	APSR_<flags> := Rm	
	즉치 플래그에서 APSR 플래그로		`MSR APSR_<flags>, #<imm8m>`	APSR_<flags> := immed_8r	
	레지스터에서 PSR로		`MSR <PSR>_<fields>, Rm`	PSR := Rm(선택한 바이트만)	
	즉치값에서 PSR로		`MSR <PSR>_<fields>, #<imm8m>`	PSR := immed_8r(선택한 바이트만)	
프로세서 상태 변경	프로세서 상태 변경	6	`CPSID <iflags> {, #<p_mode>}`	지정된 인터럽트를 비활성화하고 선택적으로 모드 변경	U, N
		6	`CPSIE <iflags> {, #<p_mode>}`	지정된 인터럽트를 활성화하고 선택적으로 모드 변경	U, N
	프로세서 모드 변경	6	`CPS #<p_mode>`		U
	엔디안 설정	6	`SETEND <endianness>`	로드 및 엔디안 설정 및 저장. `<endianness>`는 `BE`(빅엔디안)거나 `LE`(리틀엔디안)일 수 있음	U, N

ARM 및 Thumb-2 명령어 세트
빠른 참조 카드

단일 데이터 항목 로드 및 저장			어셈블러	<op>가 LDR일 경우의 동작	<op>가 STR일 경우의 동작	메모
단어 바이트 또는 하프워드 로드 또는 저장	즉치 오프셋		`<op>{size}{T} Rd, [Rn {, #<offset>}]{!}`	Rd := [address, size]	[address, size] := Rd	1, N
	사후 인덱싱, 즉치값		`<op>{size}{T} Rd, [Rn], #<offset>`	Rd := [address, size]	[address, size] := Rd	2
	레지스터 오프셋		`<op>{size} Rd, [Rn, +/-Rm {, <opsh>}]{!}`	Rd := [address, size]	[address, size] := Rd	3, N
	사후 인덱싱, 레지스터		`<op>{size}{T} Rd, [Rn], +/-Rm {, <opsh>}`	Rd := [address, size]	[address, size] := Rd	4
	PC 기준		`<op>{size} Rd, <label>`	Rd := [label, size]	사용할 수 없음	5, N
더블워드 로드 또는 저장	즉치 오프셋	5E	`<op>D Rd1, Rd2, [Rn {, #<offset>}]{!}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	6, 9
	사후 인덱싱, 즉치값	5E	`<op>D Rd1, Rd2, [Rn], #<offset>`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	6, 9
	레지스터 오프셋	5E	`<op>D Rd1, Rd2, [Rn, +/-Rm {, <opsh>}]{!}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	7, 9
	사후 인덱싱, 레지스터	5E	`<op>D Rd1, Rd2, [Rn], +/-Rm {, <opsh>}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	7, 9
	PC 기준	5E	`<op>D Rd1, Rd2, <label>`	Rd1 := [label], Rd2 := [label + 4]	사용할 수 없음	8, 9

데이터 또는 명령어 사전 로드		§(PLD)	§(PLI)	§(PLDW)	어셈블러	<op>가 PLD일 경우의 동작	<op>가 PLI일 경우의 동작	<op>가 PLDW일 경우의 동작	메모
	즉치 오프셋	5E	7	7MP	`<op> [Rn {, #<offset>}]`	[address, 32] 사전 로드 (데이터)	[address, 32] 사전 로드 (명령어)	쓰기를 위해 사전 로드[address, 32](데이터)	1, C
	레지스터 오프셋	5E	7	7MP	`<op> [Rn, +/-Rm {, <opsh>}]`	[address, 32] 사전 로드 (데이터)	[address, 32] 사전 로드 (명령어)	쓰기를 위해 사전 로드[address, 32](데이터)	3, C
	PC 기준	5E	7		`<op> <label>`	[label, 32] 사전 로드 (데이터)	[label, 32] 사전 로드 (명령어)		5, C

기타 메모리 연산			어셈블러	동작	메모
다중 로드	블록 데이터 로드		`LDM{IA|IB|DA|DB} Rn{!}, <reglist-PC>`	[Rn]에서 레지스터 목록 로드	N, I
	반환(및 전환)		`LDM{IA|IB|DA|DB} Rn{!}, <reglist+PC>`	레지스터에 로드, PC := [address][31:1] (§ 5T: [address][0]이 1일 경우 Thumb으로 변경)	I
	및 CPSR 복원		`LDM{IA|IB|DA|DB} Rn{!}, <reglist+PC>^`	레지스터 로드, 분기(§ 5T: 및 전환), CPSR := SPSR. 예외 모드 전용	I
	사용자 모드 레지스터		`LDM{IA|IB|DA|DB} Rn, <reglist-PC>^`	[Rn]에서 사용자 모드 레지스터 목록 로드. 권한 모드 전용	I
팝			`POP <reglist>`	`LDM SP!, <reglist>`의 표준 형식	N
단독 로드	세마포 연산	6	`LDREX Rd, [Rn]`	Rd := [Rn], 주소를 단독 액세스로 태그 설정함. 공유 주소가 아닌 경우 미결정 태그가 설정됨 Rd, Rn은 PC가 아님	
	하프워드 또는 바이트	6K	`LDREX{H|B} Rd, [Rn]`	Rd[15:0] := [Rn] 또는 Rd[7:0] := [Rn], 주소를 단독 액세스로 태그 설정함 공유 주소가 아닌 경우 미결정 태그가 설정됨. Rd, Rn은 PC가 아님	
	더블워드	6K	`LDREXD Rd1, Rd2, [Rn]`	Rd1 := [Rn], Rd2 := [Rn+4], 주소를 단독 액세스로 태그 설정함 공유 주소가 아닌 경우 미결정 태그가 설정됨. Rd1, Rd2, Rn은 PC가 아님	9
다중 저장	푸시, 또는 블록 데이터 저장		`STM{IA|IB|DA|DB} Rn{!}, <reglist>`	[Rn]에 레지스터 목록 저장	N, I
	사용자 모드 레지스터		`STM{IA|IB|DA|DB} Rn{!}, <reglist>^`	[Rn]에 사용자 모드 레지스터 목록 저장. 권한 모드 전용	I
푸시			`PUSH <reglist>`	`STMDB SP!, <reglist>`의 표준 형식	N
단독 저장	세마포 연산	6	`STREX Rd, Rm, [Rn]`	허용되는 경우 [Rn] := Rm, 단독 태그 지움, Rd := 0. 그렇지 않으면 Rd := 1. Rd, Rm, Rn은 PC가 아님	
	하프워드 또는 바이트	6K	`STREX{H|B} Rd, Rm, [Rn]`	허용되는 경우 [Rn] := Rm[15:0] 또는 [Rn] := Rm[7:0], 단독 태그 지움, Rd := 0. 그렇지 않으면 Rd := 0. Else Rd := 1 Rd. Rd, Rm, Rn은 PC가 아님	
	더블워드	6K	`STREXD Rd, Rm1, Rm2, [Rn]`	허용되는 경우 [Rn] := Rm1, [Rn+4] := Rm2, 단독 태그 지움, Rd := 0. 그렇지 않으면 Rd := 1 Rd, Rm1, Rm2, Rn은 PC가 아님	10
단독 지우기		6K	CLREX	지역 프로세서 단독 태그 지우기	C

메모: 로드, 저장 및 사전 로드 연산의 옵션 사용 가능성 및 범위

참고	ARM 워드, B, D	ARM SB, H, SH	ARM T, BT	Thumb-2 워드, B, SB, H, SH, D	Thumb-2 T, BT, SBT, HT, SHT
1	오프셋: – 4095 ~ +4095	오프셋: –255 ~ +255	사용할 수 없음	오프셋: 쓰기 되돌림의 경우 –255 ~ +255, 그 밖의 경우 –255 ~ +4095	오프셋: 0 ~ +255, 쓰기 되돌림 허용 안 됨
2	오프셋: – 4095 ~ +4095	오프셋: –255 ~ +255	오프셋: – 4095 ~ +4095	오프셋: –255 ~ +255	사용할 수 없음
3	`{, <opsh>}`의 전체 범위	`{, <opsh>}` 허용 안 됨	사용할 수 없음	`<opsh>`가 `LSL #<sh>`로 제한됨. `<sh>` 범위 0 ~ 3	사용할 수 없음
4	`{, <opsh>}`의 전체 범위	`{, <opsh>}` 허용 안 됨	`{, <opsh>}`의 전체 범위	사용할 수 없음	사용할 수 없음
5	현재 명령어의 +/– 4092 내에 있는 레이블	사용할 수 없음	사용할 수 없음	현재 명령어의 +/– 4092 내에 있는 레이블	사용할 수 없음
6	오프셋: –255 ~ +255	-	-	오프셋: –1020 ~ +1020, 4의 배수여야 함	-
7	`{, <opsh>}` 허용 안 됨	-	-	사용할 수 없음	-
8	현재 명령어의 +/– 252 내에 있는 레이블	-	-	사용할 수 없음	-
9	Rd1은 짝수이며 r14가 아님. Rd2 == Rd1 + 1	-	-	Rd1 != PC, Rd2 != PC	-
10	Rm1은 짝수이며 r14가 아님. Rm2 == Rm1 + 1	-	-	Rm1 != PC, Rm2 != PC	-

ARM 및 Thumb-2 명령어 세트
빠른 참조 카드

보조 프로세서 연산	§	어셈블러		동작	메모
데이터 연산		CDP{2} <copr>, <op1>, CRd, CRn, CRm{, <op2>}		보조 프로세서가 정의됨	C2
보조 프로세서에서 ARM 레지스터로 이동		MRC{2} <copr>, <op1>, Rd, CRn, CRm{, <op2>}		보조 프로세서가 정의됨	C2
두 개의 ARM 레지스터 이동	5E	MRRC <copr>, <op1>, Rd, Rn, CRm		보조 프로세서가 정의됨	
두 개의 대체 ARM 레지스터 이동	6	MRRC2 <copr>, <op1>, Rd, Rn, CRm		보조 프로세서가 정의됨	C
ARM 레지스터에서 보조 프로세서로 이동		MCR{2} <copr>, <op1>, Rd, CRn, CRm{, <op2>}		보조 프로세서가 정의됨	C2
두 개의 ARM 레지스터 이동	5E	MCRR <copr>, <op1>, Rd, Rn, CRm		보조 프로세서가 정의됨	
두 개의 대체 ARM 레지스터 이동	6	MCRR2 <copr>, <op1>, Rd, Rn, CRm		보조 프로세서가 정의됨	C
로드 및 저장, 사전 인덱싱		<op>{2} <copr>, CRd, [Rn, #+/-<offset8*4>]{!}	op: LDC 또는 STC 오프셋: 0 ~ 1020 범위에서 4의 배수	보조 프로세서가 정의됨	C2
로드 및 저장, 0 오프셋		<op>{2} <copr>, CRd, [Rn] {, 8-bit copro. option}	op: LDC 또는 STC	보조 프로세서가 정의됨	C2
로드 및 저장, 사후 인덱싱		<op>{2} <copr>, CRd, [Rn], #+/-<offset8*4>	op: LDC 또는 STC 오프셋: 0 ~ 1020 범위에서 4의 배수	보조 프로세서가 정의됨	C2

기타 연산		§	어셈블러	동작	메모
워드 스왑			SWP Rd, Rm, [Rn]	temp := [Rn], [Rn] := Rm, Rd := temp	A, D
바이트 스왑			SWPB Rd, Rm, [Rn]	temp := ZeroExtend([Rn][7:0]), [Rn][7:0] := Rm[7:0], Rd := temp	A, D
반환 상태 저장		6	SRS{IA|IB|DA|DB} SP{!}, #<p_mode>	[SPm] := LR, [SPm + 4] := CPSR	C, I
예외에서 반환		6	RFE{IA|IB|DA|DB} Rn{!}	PC := [Rn], CPSR := [Rn + 4]	C, I
브레이크포인트		5	BKPT <imm16>	프리페치 중단 *또는* 디버그 상태 시작. 16비트 비트 필드가 명령어에 인코딩됨	C, N
보안 모니터 호출		Z	SMC <imm4>	보안 모니터 호출 예외. 4비트 비트 필드가 명령어에 인코딩됨. 이전 SMI	
관리자 호출			SVC <imm24>	관리자 호출 예외. 24비트 비트 필드가 명령어에 인코딩됨. 이전 SWI.	N
연산 없음		6K	NOP	없음. 시간이 전혀 소요되지 않을 수 있음	N, V
힌트	디버그 힌트	7	DBG	디버그 힌트는 디버그 및 관련 시스템에 대한 힌트를 제공	
	데이터 메모리 장벽	7	DMB	메모리 액세스의 점유 순서를 확인	C
	데이터 동기화 장벽	7	DSB	메모리 액세스의 완료를 확인	C
	명령어 동기화 장벽	7	ISB	프로세서 파이프라인 플러시 및 예상 논리 분기	C
	이벤트 설정	6K	SEV	다중 프로세서 시스템에서 이벤트 알림 구현되지 않은 경우 NOP	N
	이벤트 대기	6K	WFE	이벤트, IRQ, FIQ, 부정확한 중단 또는 디버그 시작 요청 대기. 구현되지 않은 경우 NOP	N
	인터럽트 대기	6K	WFI	IRQ, FIQ, 부정확한 중단 또는 디버그 시작 요청 대기. 구현되지 않은 경우 NOP	N
	양도	6K	YIELD	대체 스레드에 제어권 양도. 구현되지 않은 경우 NOP	N

메모

A	Thumb 상태에서 사용할 수 없음	**P**	이 명령어에서 Rn은 Thumb 상태의 PC일 수 있습니다.
B	IT 블록에 있지 않아도 Thumb 상태에서 조건부로 실행될 수 있음	**Q**	포화(더하기 또는 빼기)나 오버플로(곱하기)가 발생할 경우 Q 플래그를 설정함. Q 플래그를 읽고 재설정하는 데는 MRS 및 MSR을 사용함
C	ARM 상태에서 조건 코드를 사용할 수 없음	**R**	<sh> 범위는 ARM 명령어에서 1 ~ 32 사이임
C2	선택 사항 2는 ARMv5에서 사용할 수 있음. 대체 연산을 제공하며 ARM 상태에서 대체 형식에 조건 코드를 사용할 수 없음	**S**	S 한정자는 Thumb-2 명령어에서는 사용할 수 없음
D	향후 제공되지 않음. LDREX 및 STREX를 대신 사용하십시오.	**T**	ARM 상태에서 사용할 수 없음
G	개별 연산의 결과에 따라 CPSR의 GE 플래그 네 개를 업데이트	**U**	IT 블록에서 사용할 수 없음. ARM 또는 Thumb 상태에서 조건 코드 사용할 수 없음
I	IA는 기본값으로 대개 생략	**V**	NOP 명령어를 사용할 수 없는 경우 어셈블러가 적절한 명령어를 삽입
L	ARM: <imm8m>. 16비트 Thumb: 0 ~ 1020 범위에서 4의 배수. 32비트 Thumb: 0-4095.		
N	이 명령어의 일부 또는 모든 형식이 Thumb-2 코드에서는 16비트(Narrow) 명령어임. 자세한 내용은 *Thumb 16 비트 명령어 세트(UAL) 빠른 참조 카드*를 참조하십시오.		

ARM 및 Thumb-2 명령어 세트
빠른 참조 카드

ARM 아키텍처 버전	
n	ARM 아키텍처 버전 *n* 이상
*n*T, *n*J	ARM 아키텍처 버전 *n* 이상의 T 또는 J 변형
*5*E	ARM 버전 5E 및 6 이상
T2	ARM 버전 6 이상의 모든 Thumb-2 버전
6K	ARM 명령어의 ARMv6K 이상, Thumb의 ARMv7
7MP	다중 처리 확장을 구현하는 ARMv7 아키텍처
Z	ARM 버전 6 이상의 모든 보안 확장 버전
RM	ARMv7-R 및 ARMv7-M 전용
XS	XScale 보조 프로세서 명령어

유연한 피연산자 2	
즉치값	`#<imm8m>`
레지스터, 선택적으로 상수에 의해 시프트됨(아래 참조)	`Rm {, <opsh>}`
레지스터, 레지스터에 의해 왼쪽으로 논리 시프트	`Rm, LSL Rs`
레지스터, 레지스터에 의해 오른쪽으로 논리 시프트	`Rm, LSR Rs`
레지스터, 레지스터에 의해 오른쪽으로 산술 시프트	`Rm, ASR Rs`
레지스터, 레지스터에 의해 오른쪽으로 회전	`Rm, ROR Rs`

레지스터, 선택적으로 상수에 의해 시프트됨		
(시프트 안 함)	`Rm`	`Rm, LSL #0`과 같음
왼쪽으로 논리 시프트	`Rm, LSL #<shift>`	허용되는 시프트 0-31
오른쪽으로 논리 시프트	`Rm, LSR #<shift>`	허용되는 시프트 1-32
오른쪽으로 산술 시프트	`Rm, ASR #<shift>`	허용되는 시프트 1-32
오른쪽으로 회전	`Rm, ROR #<shift>`	허용되는 시프트 1-31
확장을 포함하여 오른쪽으로 회전	`Rm, RRX`	

PSR 필드	(하나 이상의 접미사 사용)	
접미사	**의미**	
c	제어 필드 마스크 바이트	PSR[7:0]
f	플래그 필드 마스크 바이트	PSR[31:24]
s	상태 필드 마스크 바이트	PSR[23:16]
x	확장 필드 마스크 바이트	PSR[15:8]

조건 필드		
니모닉	**설명**	**설명(VFP)**
EQ	같음	같음
NE	같지 않음	같지 않음 또는 순서가 지정되지 않음
CS / HS	carry 설정/부호 없는 높거나 같음	크거나 같음 또는 순서가 지정되지 않음
CC / LO	carry 지우기/부호 없는 낮음	보다 작음
MI	음수	보다 작음
PL	양수 또는 0	크거나 같음 또는 순서가 지정되지 않음
VS	오버플로	순서가 지정되지 않음(하나 이상의 NaN 피연산자)
VC	오버플로 없음	순서가 지정됨
HI	부호 없는 높음	보다 큼 또는 순서가 지정되지 않음
LS	부호 없는 낮거나 같음	작거나 같음
GE	부호 있으면서 크거나 같음	크거나 같음
LT	부호 있으면서 보다 작음	보다 작음 또는 순서가 지정되지 않음
GT	부호 있으면서 보다 큼	보다 큼
LE	부호 있으면서 작거나 같음	작거나 같음 또는 순서가 지정되지 않음
AL	항상(대개 생략됨)	항상(대개 생략됨)

모든 ARM 명령어(참고 C 또는 참고 U가 있는 명령어 제외)는 명령 니모닉 다음에, 즉 이 카드에 표시된 명령어의 첫 번째 공백 앞에 이러한 조건 코드 중 하나가 있을 수 있습니다. 이 조건은 명령어에 인코딩됩니다.
모든 Thumb-2 명령어(참고 U가 있는 명령어 제외)는 명령 니모닉 다음에 이러한 조건 코드 중 하나가 있을 수 있습니다. 이 조건은 위의 IT 명령어(조건 분기 명령어 제외)에 인코딩됩니다. 명령어의 조건 코드는 위의 IT 명령어에 있는 조건 코드와 일치해야 합니다.
Thumb-2가 없는 프로세서에서 조건 코드가 있을 수 있는 유일한 Thumb 명령어는 `B <label>`뿐입니다.

프로세서 모드	
16	사용자
17	FIQ 고속 인터럽트
18	IRQ 인터럽트
19	관리자
23	중단
27	정의되지 않음
31	시스템

병렬 명령어의 접두사	
S	부호 있는 산술 모듈로 2^8 또는 2^{16}, CPSR GE 비트 설정
Q	부호 있는 포화 산술
SH	부호 있는 산술, 양분 결과
U	부호 없는 산술 모듈로 2^8 또는 2^{16}, CPSR GE 비트 설정
UQ	부호 없는 포화 산술
UH	부호 없는 산술, 양분 결과

소유권 고지 사항

설명서 번호

ARM QRC 0001M

변경 내역

발행판	날짜	변경된 내용	발행판	날짜	변경된 내용
A	1995년 6월	첫 번째 릴리스	B	1996년 9월	두 번째 릴리스
C	1998년 11월	세 번째 릴리스	D	1999년 10월	네 번째 릴리스
E	2000년 10월	다섯 번째 릴리스	F	2001년 9월	여섯 번째 릴리스
G	2003년 1월	일곱 번째 릴리스	H	2003년 10월	여덟 번째 릴리스
I	2004년 12월	아홉 번째 릴리스	J	2005년 5월	RVCT 2.2 SP1
K	2006년 3월	RVCT 3.0	L	2007년 3월	RVCT 3.1
M	2008년 9월	RVCT 4.0			