Thumb® 16 비트 명령어 세트
빠른 참조 카드

이 카드에서는 ARM 버전 6T2 이전의 Thumb 사용 가능 프로세서에서 사용할 수 있는 모든 Thumb 명령어의 목록을 제공합니다. 또한 Thumb-2 16비트 명령어 목록도 제공합니다.
이 카드에 표시된 명령어는 다른 설명이 없는 한 Thumb-2에서 모두 16비트입니다.
레지스터는 달리 지정된 경우를 제외하고 모두 Lo(R0-R7)입니다. Hi 레지스터는 R8-R15입니다.

표기 규칙 풀이				
§	**ARM 아키텍처 버전** 표 참조		<loreglist+LR>	쉼표로 구분되고 중괄호 { 및 }로 묶인 Lo 레지스터 목록 + LR
<loreglist>	쉼표로 구분되고 중괄호 { 및 }로 묶인 Lo 레지스터 목록		<loreglist+PC>	쉼표로 구분되고 중괄호 { 및 }로 묶인 Lo 레지스터 목록 + PC

연산		§	어셈블러	업데이트	동작	메모
이동	즉치값		`MOVS Rd, #<imm>`	N Z	Rd := imm	imm 범위 0 ~ 255
	Lo를 Lo로 이동		`MOVS Rd, Rm`	N Z	Rd := Rm	`LSLS Rd, Rm, #0`의 동의어
	Hi를 Lo에, Lo를 Hi에, Hi를 Hi에 더하기		`MOV Rd, Rm`		Rd := Rm	Lo를 Lo에 더하지 않음
	임의 값을 임의 값에 더하기	6	`MOV Rd, Rm`		Rd := Rm	임의 레지스터를 임의 레지스터에 더하기
더하기	즉치값 3		`ADDS Rd, Rn, #<imm>`	N Z C V	Rd := Rn + imm	imm 범위 0 ~ 7
	모든 레지스터의 Lo		`ADDS Rd, Rn, Rm`	N Z C V	Rd := Rn + Rm	
	Hi를 Lo에, Lo를 Hi에, Hi를 Hi에 더하기		`ADD Rd, Rd, Rm`		Rd := Rd + Rm	Lo를 Lo에 더하지는 않음
	임의 값을 임의 값에 더하기	T2	`ADD Rd, Rd, Rm`		Rd := Rd + Rm	임의 레지스터를 임의 레지스터에 더하기
	즉치값 8		`ADDS Rd, Rd, #<imm>`	N Z C V	Rd := Rd + imm	imm 범위 0 ~ 255
	carry 포함		`ADCS Rd, Rd, Rm`	N Z C V	Rd := Rd + Rm + C-bit	
	값을 SP에 더하기		`ADD SP, SP, #<imm>`		SP := SP + imm	imm 범위 0 ~ 508(워드로 정렬)
	SP의 폼 주소		`ADD Rd, SP, #<imm>`		Rd := SP + imm	imm 범위 0 ~ 1020(워드로 정렬)
	PC의 폼 주소		`ADR Rd, <label>`		Rd := label	label 범위 PC ~ PC+1020(워드로 정렬)
빼기	Lo 및 Lo		`SUBS Rd, Rn, Rm`	N Z C V	Rd := Rn – Rm	
	즉치값 3		`SUBS Rd, Rn, #<imm>`	N Z C V	Rd := Rn – imm	imm 범위 0 ~ 7
	즉치값 8		`SUBS Rd, Rd, #<imm>`	N Z C V	Rd := Rd – imm	imm 범위 0 ~ 255
	carry 포함		`SBCS Rd, Rd, Rm`	N Z C V	Rd := Rd – Rm – NOT C-bit	
	SP에서 값 빼기		`SUB SP, SP, #<imm>`		SP := SP – imm	imm 범위 0 ~ 508(워드로 정렬)
	부정		`RSBS Rd, Rn, #0`	N Z C V	Rd := – Rn	동의어: `NEGS Rd, Rn`
곱하기	곱하기		`MULS Rd, Rm, Rd`	N Z * *	Rd := Rm * Rd	* C 및 V 플래그는 4T에서 예상할 수 없으며, §5T 이상에서는 변경되지 않음
비교			`CMP Rn, Rm`	N Z C V	Rn – Rm에서 APSR 플래그 업데이트	Lo와 Lo, Lo와 Hi, Hi와 Lo, 또는 Hi와 Hi를 비교할 수 있음
	음수		`CMN Rn, Rm`	N Z C V	Rn + Rm에서 APSR 플래그 업데이트	
	즉치값		`CMP Rn, #<imm>`	N Z C V	Rn – imm에서 APSR 플래그 업데이트	imm 범위 0 ~ 255
논리	AND		`ANDS Rd, Rd, Rm`	N Z	Rd := Rd AND Rm	
	배타적 OR		`EORS Rd, Rd, Rm`	N Z	Rd := Rd EOR Rm	
	OR		`ORRS Rd, Rd, Rm`	N Z	Rd := Rd OR Rm	
	비트 지우기		`BICS Rd, Rd, Rm`	N Z	Rd := Rd AND NOT Rm	
	이동하지 않음		`MVNS Rd, Rd, Rm`	N Z	Rd := NOT Rm	
	비트 테스트		`TST Rn, Rm`	N Z	Rn AND Rm에서 APSR 플래그 업데이트	
시프트/회전	왼쪽으로 논리 시프트		`LSLS Rd, Rm, #<shift>`	N Z C*	Rd := Rm << shift	허용되는 시프트는 0에서 31 사이. 시프트가 0인 경우 * C 플래그는 영향 없음
			`LSLS Rd, Rd, Rs`	N Z C*	Rd := Rd << Rs[7:0]	Rs[7:0]이 0인 경우 * C 플래그는 영향 없음
	오른쪽으로 논리 시프트		`LSRS Rd, Rm, #<shift>`	N Z C	Rd := Rm >> shift	허용되는 시프트는 1에서 32 사이
			`LSRS Rd, Rd, Rs`	N Z C*	Rd := Rd >> Rs[7:0]	Rs[7:0]이 0인 경우 * C 플래그는 영향 없음
	오른쪽으로 산술 시프트		`ASRS Rd, Rm, #<shift>`	N Z C	Rd := Rm ASR shift	허용되는 시프트는 1에서 32 사이
			`ASRS Rd, Rd, Rs`	N Z C*	Rd := Rd ASR Rs[7:0]	Rs[7:0]이 0인 경우 * C 플래그는 영향 없음
	오른쪽으로 회전		`RORS Rd, Rd, Rs`	N Z C*	Rd := Rd ROR Rs[7:0]	Rs[7:0]이 0인 경우 * C 플래그는 영향 없음

Thumb 16 비트 명령어 세트
빠른 참조 카드

연산		§	어셈블러	동작	메모
로드	즉치 오프셋 포함, 워드		`LDR Rd, [Rn, #<imm>]`	Rd := [Rn + imm]	imm 범위 0 ~ 124, 4의 배수
	하프워드		`LDRH Rd, [Rn, #<imm>]`	Rd := ZeroExtend([Rn + imm][15:0])	31:16비트 지우기. imm 범위 0 ~ 62, 짝수
	바이트		`LDRB Rd, [Rn, #<imm>]`	Rd := ZeroExtend([Rn + imm][7:0])	31:8비트 지우기. imm 범위 0 ~ 31
	레지스터 오프셋 포함, 워드		`LDR Rd, [Rn, Rm]`	Rd := [Rn + Rm]	
	하프워드		`LDRH Rd, [Rn, Rm]`	Rd := ZeroExtend([Rn + Rm][15:0])	31:16비트 지우기
	부호 있는 하프워드		`LDRSH Rd, [Rn, Rm]`	Rd := SignExtend([Rn + Rm][15:0])	31:16비트를 15비트로 설정
	바이트		`LDRB Rd, [Rn, Rm]`	Rd := ZeroExtend([Rn + Rm][7:0])	31:8비트 지우기
	부호 있는 바이트		`LDRSB Rd, [Rn, Rm]`	Rd := SignExtend([Rn + Rm][7:0])	31:8비트를 7비트로 설정
	PC 기준		`LDR Rd, <label>`	Rd := [label]	label 범위 PC ~ PC+1020(워드로 정렬)
	SP 기준		`LDR Rd, [SP, #<imm>]`	Rd := [SP + imm]	imm 범위 0 ~ 1020, 4의 배수
	다중 레지스터, 기본 레지스터 포함하지 않음		`LDM Rn!, <loreglist>`	레지스터 목록 로드(Rn 포함 안 함)	항상 기본 레지스터 업데이트, 이후 증가
	다중 레지스터, 기본 레지스터 포함		`LDM Rn, <loreglist>`	레지스터 목록 로드(Rn 포함)	기본 레지스터 업데이트 안 함, 이후 증가
저장	즉치 오프셋 포함, 워드		`STR Rd, [Rn, #<imm>]`	[Rn + imm] := Rd	imm 범위 0 ~ 124, 4의 배수
	하프워드		`STRH Rd, [Rn, #<imm>]`	[Rn + imm][15:0] := Rd[15:0]	Rd[31:16] 무시. imm 범위 0 ~ 62, 짝수
	바이트		`STRB Rd, [Rn, #<imm>]`	[Rn + imm][7:0] := Rd[7:0]	Rd[31:8] 무시. imm 범위 0 ~ 31
	레지스터 오프셋 포함, 워드		`STR Rd, [Rn, Rm]`	[Rn + Rm] := Rd	
	하프워드		`STRH Rd, [Rn, Rm]`	[Rn + Rm][15:0] := Rd[15:0]	Rd[31:16] 무시
	바이트		`STRB Rd, [Rn, Rm]`	[Rn + Rm][7:0] := Rd[7:0]	Rd[31:8] 무시
	SP 기준, 워드		`STR Rd, [SP, #<imm>]`	[SP + imm] := Rd	imm 범위 0 ~ 1020, 4의 배수
	다중 레지스터		`STM Rn!, <loreglist>`	레지스터 목록 저장	항상 기본 레지스터 업데이트, 이후 증가
푸시	푸시		`PUSH <loreglist>`	전체 내림차순 스택에 레지스터 푸시	
	링크 포함 푸시		`PUSH <loreglist+LR>`	전체 내림차순 스택에 LR 및 레지스터 푸시	
팝	팝		`POP <loreglist>`	전체 내림차순 스택에서 레지스터 팝	
	팝 및 반환	4T	`POP <loreglist+PC>`	레지스터 팝, PC에 로드된 주소로 분기	
	전환 포함 팝 및 반환	5T	`POP <loreglist+PC>`	address[0] = 0인 경우 ARM에 대해 삭제, 분기 및 변경	
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 **조건 필드** 표 참조
분기	조건부 분기		`B{cond} <label>`	If {cond} then PC := label	label은 현재 명령어의 –252 ~ +258바이트 내에 있어야 함 **조건 필드** 표 참조
	비교, 0인 경우(0이 아닌 경우) 분기	T2	`CB{N}Z Rn,<label>`	If Rn {==|!=} 0 then PC := label	label은 현재 명령어의 +4 ~ +130바이트 내에 있어야 함
	무조건 분기		`B <label>`	PC := label	label은 현재 명령어의 2KB 내에 있어야 함
	링크 포함 긴 분기		`BL <label>`	LR := 다음 명령어의 주소, PC := label	32비트 명령어 label은 현재 명령어의 4MB(T2: 16MB) 내에 있어야 함(T2: 16MB)
	분기 및 전환		`BX Rm`	PC := Rm AND 0xFFFFFFFE	Rm[0] = 0인 경우 ARM 상태로 변경
	링크 포함 분기 및 전환	5T	`BLX <label>`	LR := 다음 명령어의 주소, PC := label ARM으로 변경	32비트 명령어 label은 현재 명령어의 4MB(T2: 16MB) 내에 있어야 함(T2: 16MB)
	링크 포함 분기 및 전환	5T	`BLX Rm`	LR := 다음 명령어의 주소, PC := Rm AND 0xFFFFFFFE	Rm[0] = 0인 경우 ARM 상태로 변경
확장	부호 있음, 하프워드를 워드로 확장	6	`SXTH Rd, Rm`	Rd[31:0] := SignExtend(Rm[15:0])	
	부호 있음, 바이트를 워드로 확장	6	`SXTB Rd, Rm`	Rd[31:0] := SignExtend(Rm[7:0])	
	부호 없음, 하프워드를 워드로 확장	6	`UXTH Rd, Rm`	Rd[31:0] := ZeroExtend(Rm[15:0])	
	부호 없음, 바이트를 워드로 확장	6	`UXTB Rd, Rm`	Rd[31:0] := ZeroExtend(Rm[7:0])	
반전	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]	
	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]	
	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	

Thumb 16 비트 명령어 세트
빠른 참조 카드

연산		§	어셈블러	동작	메모
프로세서 상태 변경	관리자 호출		SVC <immed_8>	관리자 호출 프로세서 예외	8비트 즉치값은 명령어에 인코딩됩니다. 이전 SWI
	프로세서 상태 변경	6	CPSID <iflags>	지정된 인터럽트 사용 안 함	
		6	CPSIE <iflags>	지정된 인터럽트 사용	
	엔디안 설정	6	SETEND <endianness>	로드 및 저장에 대한 엔디안 설정	<endianness>는 BE(빅엔디안)거나 LE(리틀엔디안)일 수 있음
	브레이크포인트	5T	BKPT <immed_8>	프리페치 중단 또는 디버그 상태 시작	8비트 즉치값은 명령어에 인코딩됨
연산 없음	연산 없음		NOP	없음. 시간이 전혀 소요되지 않을 수 있음	ARM v6K 이상에서 Real NOP를 사용할 수 있음
힌트	이벤트 설정	T2	SEV	다중 프로세서 시스템에서 이벤트 알림	Thumb-2에서 NOP 실행. ARM v7에서 기능을 사용할 수 있음
	이벤트 대기	T2	WFE	이벤트, IRQ, FIQ, 부정확한 중단 또는 디버그 시작 요청 대기	Thumb-2에서 NOP 실행. ARM v7에서 기능을 사용할 수 있음
	인터럽트 대기	T2	WFI	IRQ, FIQ, 부정확한 중단 또는 디버그 시작 요청 대기	Thumb-2에서 NOP 실행. ARM v7에서 기능을 사용할 수 있음
	양도	T2	YIELD	대체 스레드에 제어권 양도	Thumb-2에서 NOP 실행. ARM v7에서 기능을 사용할 수 있음

조건 필드	
니모닉	**설명**
EQ	같음
NE	같지 않음
CS / HS	carry 설정/부호 없는 높거나 같음
CC / LO	carry 지우기/부호 없는 낮음
MI	음수
PL	양수 또는 0
VS	오버플로
VC	오버플로 없음
HI	부호 없는 높음
LS	부호 없는 낮거나 같음
GE	부호 있으면서 크거나 같음
LT	부호 있으면서 보다 작음
GT	부호 있으면서 보다 큼
LE	부호 있으면서 작거나 같음
AL	항상. B{cond}에서는 사용하지 않음

ARMv6T2 이전의 프로세서를 위한 Thumb 코드에서 cond는 조건부 분기(B{cond}) 명령어에만 나타나고 다른 곳에는 나타나지 않아야 합니다.

Thumb-2 코드에서 cond는 이러한 명령어의 어느 곳에나(CBZ, CBNZ, CPSID, CPSIE, IT 및 SETEND 제외)에 나타날 수 있습니다.
이 조건은 위의 IT 명령어(B{cond} 명령어 제외)에 인코딩됩니다.
IT 명령어가 어셈블리 언어 소스 파일에 명시적으로 지정된 경우명령어의 조건은 해당 IT 명령어와 일치해야 합니다.

ARM 아키텍처 버전	
4T	ARM 버전 4 이상의 모든 Thumb 버전
5T	ARM 버전 5 이상의 모든 Thumb 버전
6	ARM 버전 6 이상의 모든 Thumb 버전
T2	ARM 버전 6 이상의 모든 Thumb-2 버전

소유권 고지 사항

설명서 번호

ARM QRC 0006E

변경 내역

발행판	날짜	변경된 내용
A	2004년 11월	첫 번째 릴리스
B	2005년 5월	RVCT 2.2 SP1
C	2006년 3월	RVCT 3.0
D	2007년 3월	RVCT 3.1
M	2008년 9월	RVCT 4.0