Befehlsübersicht Arm® Cortex™-M0+

Die wichtigsten Assemblerbefehle auf einen Blick

Arithmetische Befehle und Transferbefehle

Befehl	Assembler	Flags	Operation	Bemerkungen
Addition	**adds** rd,rm,*imm*	NZCV	rd := rm+*imm*	0 ≤ *imm* ≤ 7
	adds rd,*imm*	NZCV	rd := rd+*imm*	0 ≤ *imm* ≤ 255
	adds rd,rm,rn	NZCV	rd := rm+rn	
	adds rd,rm	NZCV	rd := rd+rm	
	adcs rd,rm	NZCV	rd := rd+rm+C	
	add rd,rm		rd := rd+rm	auch Hi-Register verwendbar
	add rd,SP,*imm*		rd := SP+*imm*	0 ≤ *imm* ≤ 1020, durch 4 teilbar
	add SP,*imm*		SP := SP+*imm*	0 ≤ *imm* ≤ 508, durch 4 teilbar
Subtraktion	**subs** rd,rm,*imm*	NZCV	rd := rm-*imm*	0 ≤ *imm* ≤ 7
	subs rd,*imm*	NZCV	rd := rd-*imm*	0 ≤ *imm* ≤ 255
	subs rd,rm,rn	NZCV	rd := rm-rn	
	subs rd,rm	NZCV	rd := rd-rm	
	sbcs rd,rm	NZCV	rd := rd-rm-NOT(C)	
	sub SP,*imm*		SP := SP-*imm*	0 ≤ *imm* ≤ 508, durch 4 teilbar
Negation	**negs** rd,rm	NZCV	rd := -rm	
Multiplikation	**muls** rd,rm	NZ	rd := rd*rm	
Vergleich	**cmn** rm,rn	NZCV	Flags ← rm+rn	rm und rn unverändert
	cmp rm,rn	NZCV	Flags ← rm-rn	rm und rn unverändert
	cmp rm,*imm*	NZCV	Flags ← rm-*imm*	0 ≤ *imm* ≤ 255,
Erweiterung der Wortbreite	**sxth** rd,rm		rd(15:0) := rm(15:0) rd(31:16):= rm(15)	Vorzeichenerweiterung, Halbwort
	sxtb rd,rm		rd(7:0) := rm(7:0) rd(31:8) := rm(7)	Vorzeichenerweiterung, Byte
	uxth rd,rm		rd(15:0) := rm(15:0) rd(31:16):= 0	vz.-lose Erweiterung, Halbwort
	uxtb rd,rm		rd(7:0) := rm(7:0) rd(31:8) := 0	vz.-lose Erweiterung, Byte
Move	**movs** rd,*imm*	NZ	rd := *imm*	0 ≤ *imm* ≤ 255
	movs rd,rm	NZ	rd := rm	
	mov rd,rm		rd := rm	auch Hi-Register verwendbar
No Operation	**nop**		-	"Leerbefehl" ohne Wirkung, identisch mit: mov r8,r8

Anmerkungen:

- Unmittelbar angegebene Operanden können auch mit vorangestelltem Doppelkreuz # geschrieben werden. Beispiel: `adds r4,r3,#2`
- Das Zielregister kann bei vielen Befehlen auch explizit angegeben werden. Beispiel: `adcs r4,r4,r5` statt `adcs r4,r5`

Logische Befehle

Befehl	Assembler	Flags	Operation	Bemerkungen
Bitweise logische Operationen	**ands** rd,rm	NZ	rd := rd AND rm	"UND"
	eors rd,rm	NZ	rd := rd EXOR rm	"Exkl. ODER"
	orrs rd,rm	NZ	rd := rd OR rm	"ODER"
	bics rd,rm	NZ	rd := rd AND(NOT rm)	"Bit Clear"
	mvns rd,rm	NZ	rd := NOT rm	"Move Not"
	tst rm,rn	NZ	Flags ← rm AND rn	rm und rn unverändert

Schiebebefehle

Befehl	Assembler	Flags	Operation	Bemerkungen
Schieben	**lsls** rd,rm	NZC	rd := rd << rm	Logisches Linksschieben 0 ≤ *imm* ≤ 31
	lsls rd,rm,*imm*	NZC	rd := rm << *imm*	
	lsrs rd,rm	NZC	rd := rd >> rm	Logisches Rechtsschieben 0 ≤ *imm* ≤ 31
	lsrs rd,rm,*imm*	NZC	rd := rm >> *imm*	
	asrs rd,rm	NZC	rd := rd ASR rm	Arithm. Rechtsschieben 0 ≤ *imm* ≤ 31
	asrs rd,rm,*imm*	NZC	rd := rm ASR *imm*	
Rotieren	**rors** rd,rm	NZC	rd := rd ROR rm	Rechtsrotieren

Speicherzugriffe

Befehl	Assembler	Operation	Bemerkungen
Register laden	**ldr** rd,[rm,rn] **ldr** rd,[rm,*imm*]	rd := Mem[…]	Lade Wort 0 ≤ *imm* ≤ 124, *imm* durch 4 teilbar
	ldrh rd,[rm,rn] **ldrh** rd,[rm,*imm*]	rd[15:0] := Mem[…]	Lade Halbwort vorzeichenlos 0 ≤ *imm* ≤ 62, *imm* durch 2 teilbar
	ldrb rd,[rm,rn] **ldrb** rd,[rm,*imm*]	rd[7:0] := Mem[…]	Lade Byte, vorzeichenlos 0 ≤ *imm* ≤ 31
	ldrsh rd,[rm,rn]	rd[15:0] := Mem[…]	Lade Halbwort, vorzeichenbehaftet
	ldrsb rd,[rm,rn]	rd[7:0] := Mem[…]	Lade Byte, vorzeichenbehaftet
	ldr rd,=*expr*	Lade Konstante *expr*	*Pseudobefehl*
	ldr rd,*label*	rd := Mem[*label*]	*label* im Bereich PC … PC+1020
Register speichern	**str** rd,[rm,rn] **str** rd,[rm,*imm*]	Mem[…] := rd	Speichere Wort 0 ≤ *imm* ≤ 124, *imm* durch 4 teilbar
	strh rd,[rm,rn] **strh** rd,[rm,*imm*]	Mem[…] := rd[15:0]	Speichere Halbwort 0 ≤ *imm* ≤ 62, *imm* durch 2 teilbar
	strb rd,[rm,rn] **strb** rd,[rm,*imm*]	Mem[…] := rd[7:0]	Speichere Byte 0 ≤ *imm* ≤ 31
Stackzugriff	**push** {*reglist*}	Speichern der Register in der Registerliste auf Stack	
	push {*reglist*,LR}	Speichern der Register in Registerliste (incl. LR) auf Stack	
	pop {*reglist*}	Laden der Register in der Registerliste vom Stack	
	pop {*reglist*,PC}	Laden der Register in der Registerliste (incl. PC) vom Stack	
	str rd,[SP,*imm*]	Mem[SP+*imm*] := rd	0 ≤ *imm* ≤ 1020, *imm* durch 4 teilbar
	ldr rd,[SP,*imm*]	rd := Mem[SP+*imm*]	0 ≤ *imm* ≤ 1020, *imm* durch 4 teilbar

Sprungbefehle

Befehl	Assembler	Operation	Bemerkungen
Bedingter Sprung	**bcc** *label*	if *cc* then PC := *label*	Condition Code *cc:* siehe Tabelle "Sprungbedingungen" *label* muss im Bereich -252 bis +258 relativ zum aktuellen PC liegen
Unbedingter Sprung	**b** *label*	PC := *label*	
Unbedingter Sprung in Unterprogramm	**bl** *label*	PC := *label* LR := Rücksprungadresse	
Indirekter unbedingter Sprung	**bx** rm	PC := [rm]	Sprung, bei der die Sprungadresse aus einem Register stammt (auch Hi-Register), rm(0) muss 1 sein
Indirekter unbedingter Sprung in Unterprogramm	**blx** rm	PC := [rm] LR := Rücksprungadresse	Sprung in ein Unterprogramm. Die Sprungadresse stammt aus einem Register (auch Hi-Register), rm(0) muss 1 sein

Sprungbedingungen

Abkürzung (*cc*)	Bedeutung	Bedingung für Sprung
eq	equal	Z=1
ne	not equal	Z=0
cs / hs	"carry set", unsigned higher or same	C=1
cc / lo	"carry cleared", unsigned lower	C=0
mi	"minus", negative	N=1
pl	"plus", positive or zero	N=0
vs	"v set", signed overflow	V=1
vc	"v cleared", no signed overflow	V=0
hi	unsigned higher	(C=1) and (Z=0)
ls	unsigned lower or same	(C=0) or (Z=1)
ge	signed greater than or equal	N=V
lt	signed less than	N!=V
gt	signed greater than	(Z=0) and (N=V)
le	signed less than or equal	(Z=1) or (N!=V)

Ausgewählte Spezialbefehle

Befehl	Assembler	Operation	Bemerkungen
Zugriffe auf Spezialregister	**mrs** rd,*specreg*	rd := *specreg*	Lese- und Schreibzugriff auf Spezialregister, z.B. CONTROL, PSR und PRIMASK
	msr *specreg*,rm	*specreg* := rm	
Interruptverarbeitung	**cpsie** i	PRIMASK(0) := 1	Freigabe der (maskierbaren) Interrupts in der CPU
	cpsid i	PRIMASK(0) := 0	Sperren der (maskierbaren) Interrupts in der CPU
	wfi		CPU wechselt in einen Low-Power-Modus und die Ausführung des Programms wird bis zum Auftreten eines Interrupts angehalten
Supervisor Call	**svc** imm		Auslösen einer Exception per Software. Kann zum Beispiel für Betriebssystemaufrufe verwendet werden.

Assemblerdirektiven (Auswahl)

Direktive	Assembler	Funktion	
Wahl der Syntax	**.syntax** unified	Wahl der neuen Syntax "Unified Assembly Language"	
Codegenerierung	**.thumb**	Erzeugen von Thumb-Code	
	.thumb_func	LSB des nachfolgenden Labels wird auf 1 gesetzt. Damit kann das Label als Sprungziel zum Aufruf von Thumb-Code verwendet werden.	
Datei inkludieren	**.include** "*fname*"	Datei *fname* inkludieren	
Label exportieren	**.global** *label*	*label* global bekanntmachen, z.B. für Einsprung als Unterprogramm	
Konstanten	**.word** *constlist*	32-Bit-Konstanten im Speicher ablegen	*constlist* ist eine durch Kommas getrennte Liste der Konstanten
	.hword *constlist* **.short** *constlist*	16-Bit-Konstanten im Speicher ablegen	
	.byte *constlist*	8-Bit-Konstanten im Speicher ablegen	
Speicherbereich reservieren	**.space** *nb*	*nb* Bytes reservieren und mit Nullen initialisieren	
	.space *nb,val*	*nb* Bytes reservieren und mit *val* initialisieren	
Zeichenketten	**.ascii** "*string*"	Zeichenkette *string* im Speicher ablegen	
	.asciz "*string*"	Zeichenkette *string* im Speicher ablegen, Null-Byte am Ende	
Wahl eines Speichersegments	**.text** *subsegment*	Umschalten auf das Programmsegment (= Programmcode) Die Angabe eines Unterbereichs (*subsegment*) ist optional	
	.data *subsegment*	Umschalten auf das Datensegment .data (= initialisierte Werte) Die Angabe eines Unterbereichs (*subsegment*) ist optional	
	.bss	Umschalten auf das Datensegment .bss (= nicht-initialis. Werte)	
Alignment	**.align**	Übersetzung mit der nächsten durch 4 teilbaren Adresse fortsetzen	
	.align *ma*	Übersetzung mit der nächsten durch 2^{ma} teilbaren Adresse fortsetzen	
Wiederholungen	**.rept** *nrpt*	Befehle oder Direktiven zwischen .rept und .endr werden *nrpt* mal wiederholt	
	.endr		
Definition von Symbolen	**.equ** *sym,expr*	Das Symbol *sym* wird auf den Wert *expr* gesetzt *expr* kann auch auf anderen Symbolen basieren, zum Beispiel .set symbol1,(symbol2+symbol3)*4	
	.set *sym,expr*		