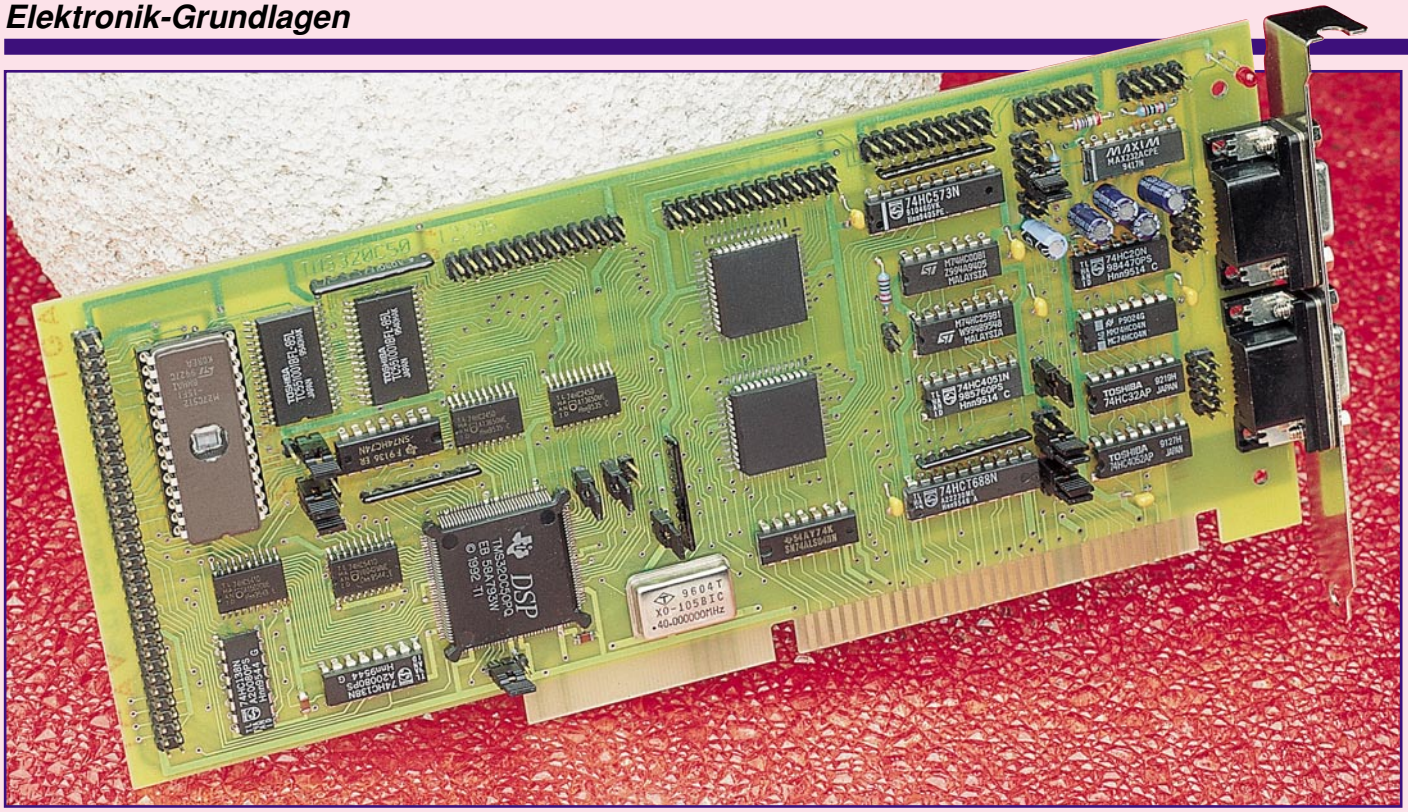

Digitale Signal-Prozessoren DSP Teil 4

Grundlagen und praktische Anwendungen von DSPs

Die Programmierung von DSPs ist das Thema des vorliegenden Artikels, gefolgt von praxisorientierten Anwendungsbeispielen.

8. Allgemeines zur Programmierung

Die Programmierung von Prozessoren aller Art wird heute durch Hochsprachen beherrscht. Will man aber die Ressourcen eines DSP voll ausschöpfen, so erreicht man im allgemeinen die besseren Ergebnisse mit einer Assemblerprogrammierung. Dieser Weg wird noch dadurch gestützt, daß der Prozessor seine volle Leistung nur erreicht, wenn er für Programm- und Datenzugriffe nicht auf externe Speicherbereiche zugreifen muß.

Es stehen uns bei dem verwendeten TMS320C50, dem Festkommaprozessor mit dem größten internen Speicherbereich, lediglich 10k-Worte zur Verfügung. Die Praxis zeigt aber, daß mit 10k-Worten in Assembler bereits sehr komplexe Programme realisierbar sind. So wurde z. B. eine Stereo-Bildauswertung mit 1k-Worten programmiert. In Pascal hätte das gleiche Programm einen Umfang von über 10 Seiten.

Die Programmierung von Signalprozessoren unterscheidet sich in vielen Punkten von den verbreiteten Prozessoren. Aus diesem Grund wollen wir uns zuerst mit der Struktur des Prozessors befassen. Bei der Darstellung des Befehlssatzes geben wir einen Überblick über die wichtigsten Wirkprinzipien und Befehlsgruppen. Der Anspruch auf Vollständigkeit soll dabei nicht erhoben werden, da die recht komplex wirkenden Befehle eine intensive Arbeit mit dem Prozessorhandbuch erfordern.

9. Die Struktur des TMS320C50

Standardprozessoren besitzen üblicherweise eine Registerbank, mit der sich relativ freizügig arbeiten läßt. Das Ergebnis sind komplizierte und langsame Befehlsstrukturen. Im Gegensatz dazu orientieren sich DSPs an den Regeln der RISC-Architektur, was einfache Funktionsstrukturen verlangt. Es entsteht dabei eine feste Funktionszuteilung der einzelnen Register zu Befehlsgruppen oder Hardwarefunktionen.

9.1 Das Registermodell

In Bild 1 sind die wesentlichsten Register dargestellt. Außer diesen existieren noch Spezialregister, die an einzelne Befehle gebunden sind, wie z. B. die Register für die Schleifenbefehle. Diese werden teilweise auch aus dem Befehl heraus genutzt. Hinzu kommen diverse Steuer- und Statusregister für den Prozessor sowie einzelne Befehle. Den Kern des Prozessors bildet der Akkumulator, der eine Arbeitsbreite von 32 Bit besitzt. Über ihn werden alle arithmetischen und logischen Operationen ausgeführt. Durch einen Tausch-Befehl kann dieser Akku gegen den Akkumulator B getauscht werden. Mit dem im Hintergrund stehenden Akku läßt sich bis auf wenige Ausnahmen nicht direkt arbeiten - er dient als Hilfsregister. Die Registerbank AR0 bis AR7 hat die Funktion von Zeigerregistern. In ihr können Pointer abgelegt werden, die auf den Datenspeicher zeigen.

Zur Manipulation der Pointer besitzt dieser Block eine eigene Pointer-ALU, deren Funktion sich auf Add und Sub beschränkt. Dies erfolgt aber dafür parallel zur eigentlichen Befehlsausführung. Da der Prozessor aufgrund seiner geringen Befehlsbreite (16 Bit) keine Möglichkeit zur Auswahl eines Pointers hat, erfolgt dies über ein gesondertes Register, den ARP. Aus dem gleichen Grund kann in einem Befehl auch

keine direkte Speicheradresse verwendet werden. Eine im Befehl angegebene Speicheradresse beschränkt sich auf die unteren 7 Bit - die restlichen (oberen) 9 Bit stehen im Seitenregister (DPR), wodurch eine direkte Adressierung immer im Bereich von 128 Worten erfolgt. Die letzte Gruppe arbeitet mit dem Multiplikationsbefehl zusammen. In TREG0 wird der erste Faktor abgelegt, der zweite Faktor steht im MUL-Befehl, und das Ergebnis erscheint im PREG.

9.2 Das Speichermodell

Die Festkomma-DSPs von TI können einen Daten - und Programmspeicher von jeweils 64k adressieren. Dieser teilt sich in interne und externe Bereiche, wobei die internen unterschiedlich konfigurierbar sind. Wir haben uns für eine Konfiguration entschieden, welche die meisten Anforderungen erfüllt. Sie ist in Bild 2 dargestellt.

9.3 Der Programmspeicher

Als eigentlicher Programmspeicher dient der interne ROM. Der von uns eingesetzte DSP enthält ein Ladeprogramm, mit dem er das eigentliche Programm von einem externen EPROM holen kann. Zur Ausführung schreibt der DSP das Programm in den Bereich ab 800h. Eine externe Programmabarbeitung ist unüblich, da sie durch die Zugriffszeiten zu langsam würde.

9.4 Datenspeicher

Er besteht aus dem großen Bereich, der als SARAM gekennzeichet ist. Dies bedeutet, daß in diesem Bereich nur ein einfacher Zugriff möglich ist. Die DARAM-Bereiche dagegen erlauben einen Doppel-Zugriff (Dual-Access-RAM), so daß sie für Arbeitsvariable geeignet sind. In diesem Speicherbereich kann ein Lesen und Schreiben zeitgleich erfolgen. Eine Besonderheit bilden die Memory-Mapped-Register (MM-Reg.). Hier befinden sich die Arbeits- und Statusregister des Prozessors.

9.5 Der Speicherzugriff

Aus der Beschreibung der Registerstruktur ergeben sich zwei verschiedene Wege des Speicherzugriffes. Der erste Weg ist die direkte Adressierung im Befehl, wobei sich die Adresse aus dem Teil im Befehl und dem Page-Pointer ergibt.

Der zweite Weg ist die indirekte Adressierung. Hierbei wird die Adresse aus dem durch den ARP ausgewählten Pointerregister verwendet.

10. Das Befehlssystem des TMS320C50

Nachdem wir uns mit der Struktur des Prozessors befaßt haben, wenden wir uns nun dem Befehlssystem zu.

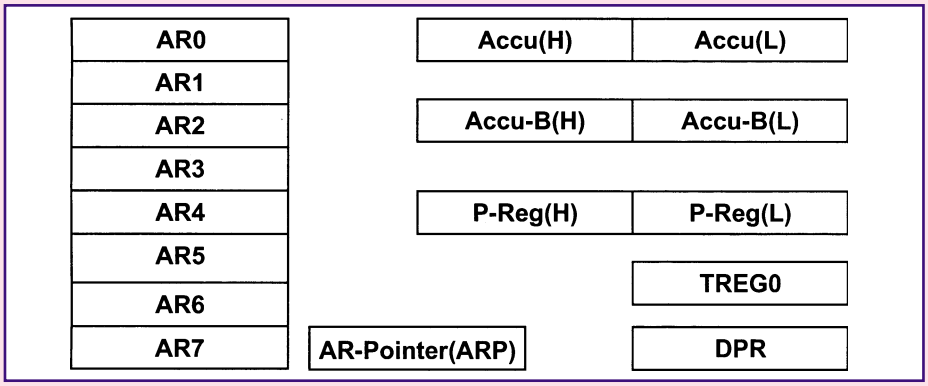

Bild 1: Hauptregister des TMS 320C50

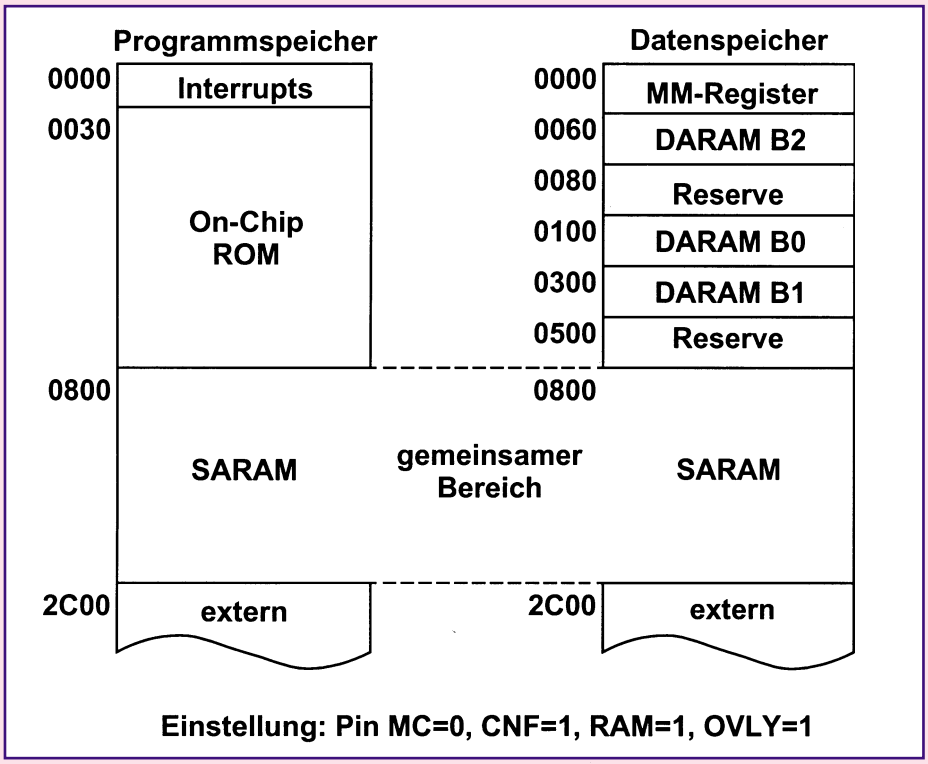

Bild 2: Speichermodell des TMS 320C50

Tabelle 1: Befehlskonstruktionen, die für den größten Teil des Befehlssatzes gelten

	Befehl	Beispiel	
kein Operand		ABS	
ein Operand	dma	ADD	temp1
	#k	MPY	#10
zwei Operanden	ind [,nextARP]	MPYA	*+, AR3
	dma [,shift]	LACC	temp1,5
	#k [,shift]	AND	#0ffh,16
drei Operanden	ind [,shift [,nextARP]]	ADD	*-, 0, AR2
	pma [,ind [,nextARP]]	CALL	m1, *, AR0
		BANZ	anf, *-
	pma, cond1 [,cond2]	BCND	m2, NZ

10.1 Aufbau der Befehlssyntax

Der Aufbau der Assemblerzeile entspricht dem der üblichen Assembler. Es wird keine Groß-/Kleinschreibung unterschieden, die einzelnen Felder sind mit Leerzeichen oder Tabulator zu trennen, und die Marken werden mit einem Doppelpunkt abgeschlossen.

Für die ausführbaren Befehle gilt die Schreibweise im Prozessorhandbuch, die Pseudobefehle sind in der Anleitung zum Assembler zu finden. Da bei letzterem mehrere Varianten gebräuchlich sind, wurden teilweise mehrere Schreibweisen für die gleiche Funktion aufgenommen. Als Beispiel sei dafür genannt die Definition eines Speicherbereiches im Wortformat

durch .BYTE oder DW, wie es beim Microsoft-Assembler üblich ist. Der größte Teil der Befehle läßt sich in einem Grundschema darstellen. Dabei werden zuvor die Abkürzungen geklärt:

dma	direkte Speicheradresse (die unteren 7 Bit)
pma	eine volle Programmadresse
ind	Funktionssysmbol für die indirekte Adressierung (z. B. Post-Increment) des aktuellen Pointerregisters
shift	Anzahl von Bits, die der Operand rechts oder links geschoben wird
#k	eine numerische Konstante
nextARP	Name des Pointerregisters, welches im nächsten Befehl aktiv sein soll
cond	eine Sprungbedingung

Es ergeben sich nun die Befehlskonstruktionen in Tabelle 1, die für den größten Teil des Befehlssatzes gelten.

Bei den Indirekt-Operanden sind 3 Varianten der möglichen 8 besonders häufig in der Anwendung:

*	keine Veränderung des aktiven Pointers,
*+	erhöhe den aktiven Pointer um 1,
*-	erniedrige den aktiven Pointer um 1.

Nachdem die Befehlskonstruktionen geklärt sind und somit ein erster Eindruck über die Schreibweise vorhanden ist, wollen wir einige Befehlsgruppen schwerpunktmäßig betrachten.

10.2 Ladebefehle (Tabelle 2)

Bis auf wenige Ausnahmen bewegen alle Lade- (LOAD) oder Schreibbefehle (STORE) ein Datum zwischen dem Speicher und einem Register. Im Gegensatz zu anderen Prozessoren ist es nicht an der Stellung der Operanden ersichtlich, von wo nach wo transportiert wird. Hier ist die Stellung der Operanden immer gleich, und der Befehl definiert die Richtung. Auch kommt man schnell zu dem Schluß, daß ein Transport zwischen zwei Registern nicht ohne Probleme möglich ist.

10.3 Arithmetische Befehle (Tabelle 3)

Die arithmetischen Befehle beschränken sich auf Addition, Subtraktion, Multiplikation und Quadrierung. Für diese Operationen ist das Ergebnis immer 32 Bit groß. Leider gibt es in dieser Prozessorfamilie keine Division. Ergänzt wird diese Gruppe durch die zwei einzigen direkten arithmetischen Operationen mit den Pointerregistern.

In der Tabelle 2 gelten dabei alle aufgeführten Operanden-Versionen für die davorstehenden Befehle.

Tabelle 2: die häufigsten Ladebefehle

Befehl	Operanden	Beispiel	Beschreibung
LACC	dma [,shift]	LACC temp1	Laden des Akkumulators vom Speicher
	ind [,shift [,nextARP]]	LACC *+,0,AR2	indirekt vom Speicher
	#k [,shift]	LACC #5	mit 5 durch Shift wird der High-Teil des Accu erreicht
LACL	dma	LACL temp1	Laden des Akkumulators vom Speicher
	ind [,nextARP]	LACL *+ ,AR2	indirekt vom Speicher
	#k	LACL #5	mit 5
SACL	dma [,shift]	SACL temp1	Speichern des Akkumulators nach MEM
	ind [,shift [,nextARP]	SACL *+,0,AR2	indirekt zum Speicher
SACH	.. wie SACL	SACH temp2	Speichern des H-Teil des Accu nach MEM
LAR	ARx, dma	LAR AR1, temp1	Laden des Pointer AR1 vom Speicher
	ARx, ind [,nextARP]	LAR AR3, *+, AR2	indirekt vom Speicher
	ARx, #k	LACL #5	mit 5
SAR	ARx, dma	SAR AR1, temp1	Laden des Pointer AR1 zum Speicher
SAR	ARx, ind [,nextARP]	SAR AR3, *+, AR2	indirekt zum Speicher
LDP	#k oder pma	LDP Block1	Laden des Seitenzeigers
LARP	ARx	LARP AR1	Setzen des Zeigers auf den aktuellen Pointer (ist identisch mit MAR * , ARx)

Tabelle 3: die häufigsten Arithmetik-Befehle

Befehl	Operanden	Beispiel	Beschreibung
ADD	dma [,shift]	ADD temp1	AC := AC + (dma)
SUB	ind [,shift [,nextARP]]	SUB *+,0,AR2	AC := AC - (mem(aktiven Pointer))
MPY	#k [,shift]	MPY #5	PREG := TREG0 * Konstante
ADDC	dma [,shift]	ADDC temp1	AC := AC + (dma) + CY
SUBB	ind [,shift [,nextARP]]	SUBB *+,0,AR2	AC := AC - (mem(aktiven Pointer) -CY
ADRK	#k	ADRK #16	aktives Pointerregister + 16
SBRK	#k	SBRK #2	aktives Pointerregister - 2

Tabelle 4: die wichtigsten Logik-Befehle

Befehl	Operanden	Beispiel	Beschreibung
AND	dma [,shift]	AND temp1	AC := AC .AND. (dma)
OR	ind [,nextARP]	OR *+,AR2	AC := AC .OR. (mem(aktiven Pointer))
XOR	#k [,shift]	XOR #5	AC := AC .XOR. 5
APL	#k, dma [,shift]		
OPL	#k, ind [,nextARP]	OPL #10h,flag1	(flag1) := (flag1) OR 10h
XPL			
BSAR	#k	BSAR #8	AC := AC shift right 8 (arithm. /logisch)
SFL	keine	SFL	AC := AC shift left 1
SFR		SFR	AC := AC shift right 1

10.4 Logische Befehle (Tabelle 4)

Die logische Verarbeitung der Daten beschränkt sich auf die Grundfunktionen AND, OR, XOR. Diese drei Befehle können außer mit dem Akkumulator auch direkt mit einer Speicherzelle ausgeführt werden.Das ist aber zeitlich nur sinnvoll mit dem internen DARAM. In diese Gruppe fallen außerdem die Verschiebebefehle.

10.5 Programmverzweigungen und Wiederholungen (Tabelle 5,6)

Die Gruppe der Programmverzweigungen und Unterprogrammaufrufe ist bei diesem Prozessor sehr komfortabel. Außer den üblichen Sprüngen besteht auch die Möglichkeit des bedingten Unterprogrammrufes, der bedingten Rückkehr vom Unterprogramm und der bedingten Befehls-

ausführung. Die Befehle lassen meist die Angabe zweier Bedingungen zu, wobei sich von den möglichen 13 Sprungbedingungen sechs auf den Akkumulator, sechs auf Flags und eine Bedingung auf einen Prozessorpin beziehen. Für die Unterprogrammaufrufe steht ein interner Stack mit 8 Ebenen zur Verfügung. Durch diesen relativ geringen Umfang ist bei der Programmierung eine Datenübergabe über den Stack bzw. ein Halten temporärer Variablen im Stack nicht sinnvoll.

Zu dieser Gruppe zählen ebenfalls die Schleifenbefehle. Einer steht bereits bei den bedingten Sprüngen. Der BANZ hat den Vorteil, das er mehrfach geschachtelt werden kann, da er sich auf die Pointerregister bezieht. Sein Nachteil ist allerdings der Zeitbedarf für die Schleifenorganisation. Aus diesem Grund existiert noch ein Repeat-Befehl und ein Befehl für die Wiederholung eines Blockes, die hardwaremäßig organisiert werden.

10.6 Die Status- und Steuerregister des DSP (Tabelle 7)

Für die Funktion des DSP und einiger Befehle ist eine Anzahl von Spezialregistern erforderlich. Diese befinden sich auf den unteren Adressen des Datenspeichers auf der Zero-Page. Die hier gebotene Auswahl beschränkt sich dabei auf einige Grundfunktionen, die für die Programmierung später benötigt werden.

10.7 Das Befehlspipelining

Zur Erreichung des hohen Befehlsdurchsatzes wird in den DSPs eine verschachtelte Ausführung der Befehle realisiert. Das bedeutet, daß während der Ausführungsphase eines Befehls bereits der nächste geholt und decodiert werden kann.

Ein einfaches Pipelining besteht meist aus den Stufen Befehl holen, Befehl decodieren, Operanden holen, Operation ausführen und Ergebnis ablegen. Jeder Befehl muß diese Stufen durchlaufen. Die Folge ist aber, daß ein Befehl, der ein Ergebnis im Speicher ablegen soll, in der Stufe „Ausführen" ist und der Folgebefehl in der Stufe „Operanden holen" bereits auf das Ergebnis zugreifen möchte.

Um diesen Konflikt zu lösen, nutzt man zwei Wege. Ein Weg überträgt dem Programmierer den Ausschluß dieses Konfliktes, das bedeutet, daß der verwendetete Compiler solche Fälle erkennen muß und durch Umsortieren von Befehlsfolgen oder Einfügen von NOPs dies behebt. Bei dem anderen Weg werden diese Fälle von der Hardware erkannt und die Bearbeitung der folgenden Befehle verzögert.

Beim TMS320C50 werden beide Wege benutzt. Bei der normalen Befehlsausführung überwacht der Prozessor diese Fälle, während bei Zugriffen auf die Zero-Page (Steuerregister) zu beachten ist, daß die Wirkung der Operation erst ein oder zwei Befehle verspätet eintritt.

Für den Fall der Sprünge, die zwar keinen Datenkonflikt hervorrufen aber schon ein Stück auf dem „falschen Weg" arbeiten, wird entweder das Ende des Sprungbefehles abgewartet oder diese Eigenschaft gezielt in das Programm mit einbezogen.

Für letzteren Fall existiert für alle Verzweigungsbefehle ein gesonderter Befehlsaufruf. Das in Tabelle 8 dargestellte kurze Programm soll dies verdeutlichen. Es handelt sich dabei um eine Division durch eine Subtraktionsschleife.

Nachdem wir die wesentlichen Aspekte der Programmierung von DSPs beleuchtet haben, folgt in Kapitel 11 die Vertiefung anhand praktischer Programmierbeispiele. ELV

Tabelle 5 : die wichtigsten Verzweigungs-Befehle

Befehl	Operanden	Beispiel	Beschreibung
B	pma, [ind [,nextARP]]	B marke1	unbedingter Sprung zur Marke1
BANZ	pma, [ind [,nextARP]]	BANZ marke1, *-	Sprung zu Marke1, wenn aktives Pointerregister <>0 , aktiver Pointer - 1
BCND	pma, [cond1 [,cond2]]	BCND marke1, LT	Sprung zu Marke1, wenn AC < 0
CALL	pma, [ind [,nextARP]]	CALL m2	Unterprogrammruf ab m2
CC	pma, [cond1 [,cond2]]	CC m2, NEQ	Unterprogrammruf ab m2, wenn AC<>0
RET		RET	Rückkehr vom Unterprogramm
RETC	cond1 [,cond2]	RETC EQ	Rückkehr vom Unterprogramm, wenn AC=0
XC	k, cond1 [,cond2]	XC 1, NEQ	Führt den nächsten Befehl aus (1 Wort - k=1), wenn die Bedingung wahr ist

Tabelle 6 : die Wiederholungs-Befehle

Befehl	Operanden	Beispiel	Beschreibung
RPT	dma	RPT temp2	Der Folgebefehl wird entrprechend dem direkt
	ind, [,nextARP]	RPT *	oder indirekt angebenen Wert+1-mal
	#k	RPT #127	wiederholt
RPTB	pma	RPTB marke3	Es werden die im Bereich zwischen dem Befehl und der durch pma angegebenen Endadresse stehenden Befehle wiederholt. Die Anzahl steht in einem speziellen Register (BRCR)

Tabelle 7 : einige Status- und Steuerregister

Name	Adresse	Funktion
IMR	04h	Interruptmasken-Register zur Freigabe der Interrupts
PMST	07h	Prozessor-Mode-Status-Register
BRCR	09h	Zähler für den Block-Repeat-Befehl
TIM	24h	Timer-Register
PRD	25h	Periodenregister des Timer
TCR	26h	Steuerregister des Timer
PDWSR	28h	Wait-Zyklenregister für Programm- und Datenspeicher (extern)
IOWSR	29h	Wait-Zyklenregister für I/O-Kanäle
CWSR	2ah	Steuerregister für Waitstates

Tabelle 8

```
; Division   AR0 := ACC / tmp0
      larp    ar0                ; aktives Pointerregister = AR0
      lar     ar0,#0             ; AR0 als Zähler mit 0 laden
kfkt3:        bcndd   kfkt3,gt; Sprung zu kfkt3, wenn AC>0
; diese zwei Befehle werden vor Ausführung des Sprunges abgearbeitet !!
      sub     tmp0               ; AC := AC - tmp0
      adrk    #1                 ; erhöhe Zähler in AR0
;
      sbrk    #1                 ; Befehl nach dem Sprung! (Korrektur des Zählers)
```