Praktikum zu Technische Grundlagen der Informatik

Ausgabe Winter-Semester 1999/2000, Stand: 25. Juli 2000

Arndt Bode
mit
Andreas Bauer, Birgit Eßbaumer, Wolfgang Karl, Markus Leberecht,
Peter Luksch, Bruno Piochacz, Max Walter, Roland Wismüller

Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM)
Institut für Informatik der Technischen Universität München
D-80290 München

Tel: (089) 289-28240, Fax: (089) 289-28232
email: bode@informatik.tu-muenchen.de

Id: DasBuch.tex,v 3.10 2000/07/25 06:52:04 wismuell Exp

7.3 Beschreibung der Zielarchitektur

7.3.1 Vorbemerkungen

Die im Rahmen des Aufgabenbereichs II zu implementierende Zielarchitektur stellt in vieler Hinsicht einen typischen Prozessor dar, der viele Eigenschaften kommerzieller Mikroprozessoren aufweist. Der Befehlssatz ist im wesentlichen an die x86-Prozessoren von Intel sowie die 680XX-Prozessoren von Motorola angelehnt. Allerdings ist die Zielarchitektur nicht als Beispiel für einen optimalen Prozessorentwurf zu betrachten. Eine wesentliche Einschränkung gegenüber kommerziellen Prozessoren ist das Fehlen von Byte- und Langwort-Operationen sowie die fehlenden Funktionen zur Betriebssystem-Unterstützung. Die Organisation der mikroprogrammierten Maschine läßt zudem auch keine hohe Leistungsfähigkeit des Prozessors zu.

7.3.2 Aufbau der Zielarchitektur

Zur Vereinfachung wurde für die Zielarchitektur eine reine 16-Bit Struktur gewählt.

Das Leitwerk übernimmt die Aufgabe der Befehlshol– und Dekodierphase. Der als nächstes auszuführende Befehl wird über den Befehlszähler adressiert. Der Befehlszähler wird nach jedem Befehl automatisch auf den im Speicher unmittelbar folgenden Befehl fortgeschaltet. Eine Ausnahme sind Sprung- und Unterprogrammbefehle, die den Befehlszähler direkt setzen können.

Das Rechenwerk besteht aus einer arithmetischen und logischen Einheit (ALU) zur Verarbeitung von 16-Bit Integer-Daten. Es enthält acht Allzweckregister (r0 bis r7) sowie den Kellerzeiger (SP). Das Rechenwerk kann auch zur Berechnung komplexerer Adreßausdrücke verwendet werden; Adreßwerte, die ebenfalls 16 Bit groß sind, können in den Rechenwerksregistern gespeichert werden. Statusinformationen werden von der ALU im Maschinenstatusregister abgelegt. Abbildung 7.1 zeigt nochmals die dem Maschinenprogrammierer zugänglichen Register.

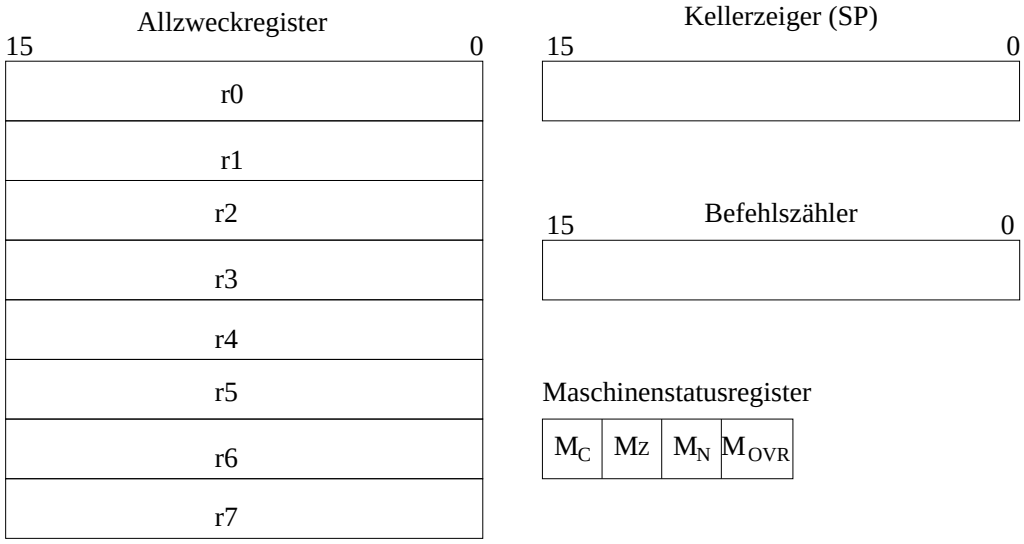


Abbildung 7.1: Maschinenregister

Der Hauptspeicher umfaßt einen Adreßraum von 64K Worten à 16 Bit. Der Zugriff auf den Hauptspeicher benötigt zwei Taktzyklen. Im ersten Taktzyklus wird die Speicherzelle adressiert und im zweiten Taktzyklus wird das Datum gelesen bzw. geschrieben. Der Zugriff erfolgt wortweise, d.h. in 16-Bit Einheiten.

Der Maschinenbefehlssatz für die Zielarchitektur umfaßt Maschinenbefehle aus folgenden Befehlsgruppen:

- Transportbefehle
- Arithmetische und logische Befehle
- Schiebe- und Rotationsbefehle
- Bit- und Bitfeldbefehle
- Vergleichsbefehle
- Verzweigungen und Sprünge
- Unterprogrammaufrufe
- Systembefehle
- Erweiterte Befehle

Der Befehlssatz der Zielarchitektur ist weitgehend orthogonal, d.h. im allgemeinen kann bei jedem Befehl jede beliebige, sinnvolle Kombination von Adressierungsarten verwendet werden. Ausnahmen sind die Kombination von Speicher- und unmittelbarer Adressierung, die durch das verwendete Befehlsformat ausgeschlossen wird, sowie einige spezielle Befehle.

7.3.3 Befehlsformate und Adressierungsarten

Die Zielarchitektur benutzt durchgehend ein Zweiadreß-Befehlsformat. Je nachdem, welche Adressierungsarten im Befehl verwendet werden, ergeben sich Einwort- bzw. Zweiwort-Befehle, wie sie in Abbildung 7.2 dargestellt sind. Im Feld Opcode ist der 8-Bit Operationscode des Befehls enthalten, in dem auch die verwendeten Adressierungsarten codiert sind. Die Opcodes für die einzelnen Befehle sind in der Befehlsbeschreibungen angegeben. Die Bedeutung der Felder RA, RB und des Konstantenfelds ist der folgenden Beschreibung der Adressierungsarten zu entnehmen.

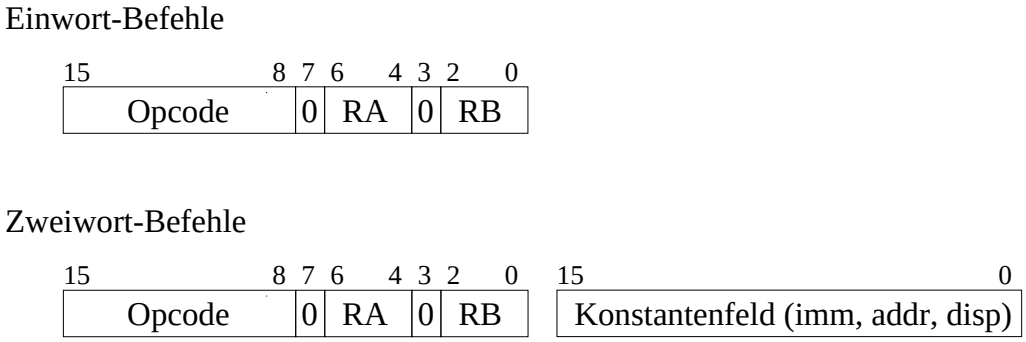


Abbildung 7.2: Befehlsformate der Zielarchitektur

Die Zielarchitektur unterstützt insgesamt fünf verschiedene Adressierungsarten für Operanden:

1. **Registeradressierung**

   Der Operand befindet sich hier in einem Prozessorregister. Die Nummer des Register ist im Feld RA (für einen Quelloperanden) bzw. RB (für einen Zieloperanden) enthalten.

2. **Unmittelbare Adressierung**

   Der Operand ist hier direkt in dem Konstantenfeld angegeben. Diese Adressierungsart ist nur für Quelloperanden zulässig.

3. **Direkte Adressierung**

   Der Operand ist hier ein Speicherwort, dessen Adresse im Konstantenfeld angegeben ist.

4. **Indirekte Adressierung**

   Der Operand ist ein Speicherwort, dessen Adresse in einem Register enthalten ist. Die Nummer dieses Registers ist im RA-Feld (für den Quelloperanden) bzw. RB-Feld (für den Zieloperanden) enthalten.

5. **Indizierte bzw. basisrelative Adressierung**

   Wie bei der direkten und indirekten Adressierung ist der Operand durch ein Speicherwort gegeben. Die Adresse des Speicherwortes ist hier jedoch durch die Summe aus dem Inhalt des Konstantenfeldes und dem Inhalt des Registers gegeben, das im RA-Feld (für den Quelloperanden) bzw. RB-Feld (für den Zieloperanden) definiert ist.

Je nach Anzahl der Operanden ergeben sich 3 Hauptvarianten von Befehlen der Zielarchitektur:

1. Befehle ohne explizite Angabe eines Operanden. Dies sind immer Einwort-Befehle.
2. Befehle mit einem Operanden (je nach Befehl entweder Quell- oder Zieloperand). Bei diesen Befehlen sind i.a. alle Adressierungsarten erlaubt, soweit diese sinnvoll sind. Für Register- und indirekte Adressierung ergeben sich Einwort-Befehle, für die anderen Adressierungsarten Zweiwort-Befehle.
3. Für Befehle mit zwei Operanden sind folgende Kombinationen von Adressierungsarten erlaubt:
   (a) Register, Register (Einwort-Befehl)
   (b) Register als Ziel, unmittelbare Adressierung für Quelle (Zweiwort-Befehl)
   (c) Register, direkte Adressierung (Zweiwort-Befehl)
   (d) Register, indirekte Adressierung (Einwort-Befehl)
   (e) Register, indizierte/basisrelative Adressierung (Zweiwort-Befehl)

Bei allen Befehlen, bei denen die Felder RA bzw. RB nicht zur Adressierung von Operanden benötigt werden, wird deren Inhalt ignoriert, kann also vom Maschinenprogrammierer beliebig gesetzt werden.

7.3.4 Der Befehlssatz der Zielarchitektur

Wichtig: In der nachfolgenden Beschreibung der einzelnen Befehle wird der Einfachheit halber von „Register ***RA***“ bzw. „Register ***RB***“ gesprochen, wobei dies jeweils „das durch das RA- (bzw. RB-) Feld des Maschinenbefehlswortes adressierte Register“ bedeutet. Einzelne Register mit fest vorgegebener Nummer werden dagegen als „r0“ ... „r7“ bezeichnet.

Darüberhinaus werden durchgehend folgende Bezeichnungen verwendet:

src	Quelloperand
dst	Zieloperand
[***RA***] / [***RB***]	Speicherwort, dessen Adresse im Register ***RA*** / ***RB*** steht
disp[***RA***] / disp[***RB***]	Speicherwort mit Adresse disp + ***RA*** / disp + ***RB***
[addr]	Speicherwort mit Adresse addr
imm	unmittelbar adressierter Quelloperand
x[i]	Bit i des Operanden x
PC	Programmzähler
SP	Kellerzeiger

Die Werte disp, addr und imm sind dabei im Konstantenfeld gespeichert.

7.3.4.1 Befehlssatz nach Verwendung geordnet (nur Kurzbeschreibungen)

Transportbefehle:

- `move` – Datentransport
- `xchg` – Datenaustausch
- `lea` – Lade effektive Adresse

Arithmetische und logische Befehle:

- `add` – Ganzzahlige Addition
- `addc` – Ganzzahlige Addition mit Übertrag
- `sub` – Ganzzahlige Subtraktion
- `subc` – Ganzzahlige Subtraktion mit Übertrag
- `umul` – Ganzzahlige vorzeichenlose Multiplikation
- `mul` – Ganzzahlige vorzeichenbehaftete Multiplikation
- `udiv` – Ganzzahlige vorzeichenlose Division
- `div` – Ganzzahlige vorzeichenbehaftete Division
- `inc` – Hochzählen
- `dec` – Herunterzählen
- `neg` – Vorzeichenwechsel
- `and` – Bitweise UND-Verknüpfung
- `or` – Bitweise ODER-Verknüpfung
- `xor` – Bitweise Exklusiv-ODER-Verknüpfung
- `not` – Bitweise Negation

Schiebe- und Rotationsbefehle:

- `asr` – Arithmetisches Schieben nach rechts
- `lsr` – Logisches Schieben nach rechts
- `lsl` – Logisches Schieben nach links
- `rr` – Rotieren nach rechts
- `rl` – Rotieren nach links
- `rrc` – Rotieren nach rechts über C-Flag
- `rlc` – Rotieren nach links über C-Flag

Bit- und Bitfeldbefehle:

- `bset` – Bit setzen
- `bclr` – Bit löschen
- `binv` – Bit invertieren
- `btst` – Bit testen
- `bscanf` – Bit-Suche vorwärts
- `bscanr` – Bit-Suche rückwärts

Vergleichsbefehle:

- `cmp` – Arithmetischer Vergleich
- `tst` – Arithmetischer Vergleich auf 0

Verzweigungs- und Sprungbefehle:

- `jmp` – Unbedingter Sprung

- `jmp`*cc* – Bedingte Sprünge
- `loop` – Schleifenbefehl
- `loopz` – Bedingter Schleifenbefehl
- `loopnz` – Bedingter Schleifenbefehl

Unterprogrammaufrufe:

- `call` – Unterprogrammsprung
- `ret` – Unterprogrammrücksprung
- `push` – Retten auf den Keller
- `pop` – Restaurieren vom Keller
- `ldsp` – Laden des Kellerzeigers
- `stsp` – Speichern des Kellerzeigers

Systembefehle:

- `halt` – Anhalten des Prozessors
- `clc` – Löschen des C-Flags
- `cmc` – Invertieren des C-Flags

Erweiterte Befehle:

- `xlat` – Tabellengesteuerte Übersetzung
- `bmove` – Blocktransport
- `bscan` – Blocksuche
- `bcmp` – Blockvergleich
- `binit` – Blockinitialisierung

7.3.4.2 Befehlssatz alphabetisch geordnet

`add` ..... **Ganzzahlige Addition**

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	08
[addr]	***RB***	09
[***RA***]	***RB***	0A
disp[***RA***]	***RB***	0B
RA	***RB***	0C
RA	[addr]	0D
RA	[***RB***]	0E
RA	disp[***RB***]	0F

Funktion:

$dst = dst + src$

Beschreibung:

Der Befehl `add` addiert die 16-Bit Zahlen in *src* und *dst* und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation gesetzt.

`addc` ..... **Ganzzahlige Addition mit Übertrag**

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	D0

Funktion:

$dst = dst + src$ + C-Flag

Beschreibung:

Der Befehl `addc` addiert die 16-Bit Zahlen in *src* und *dst* sowie den Inhalt des C-Flags im Maschinenstatusregister und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

`and` ..... **Bitweise UND-Verknüpfung**

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	18
[addr]	***RB***	19
[***RA***]	***RB***	1A
disp[***RA***]	***RB***	1B
RA	***RB***	1C
RA	[addr]	1D
RA	[***RB***]	1E
RA	disp[***RB***]	1F

Funktion:

$dst = dst \ \& \ src$

Beschreibung:

Der Befehl `and` berechnet die bitweise UND-Verknüpfung von *src* und *dst* und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

`asr` ..... Arithmetisches Schieben nach rechts

Adressierungsarten:

src	*dst*	Opcode
	RB	80
	[addr]	81
	[***RB***]	82
	disp[***RB***]	83
RA	***RB***	C0
imm	***RB***	C1

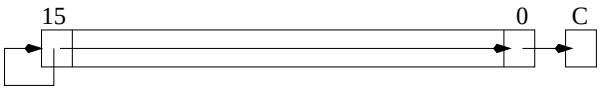
Funktion:

$dst = dst \ / \ 2$

$dst = dst \ / \ 2^{(src \ \mathbf{mod} \ 16)}$

Beschreibung:

Der Befehl `asr` schiebt den Inhalt von *dst* um eine oder mehrere Bitpositionen nach rechts, wobei das Vorzeichenbit unverändert bleibt. Die Anzahl n der Bitpositionen ist für die Formen ohne expliziten Quelloperand 1, ansonsten gilt $n = src \ \mathbf{mod} \ 16$. Der Befehl schreibt den Inhalt von Bit $n-1$ in das C-Flag, den Inhalt von Bit i (für $i \in [n, 14]$) an die Bitposition $i-n$ und den Inhalt von Bit 15 an die Bitpositionen $15...15-n$. Das folgende Bild veranschaulicht die Operation graphisch.



`bclr` ..... Bit löschen

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	6C
RA	[addr]	6D
RA	[***RB***]	6E
RA	disp[***RB***]	6F

Funktion:

$dst = dst \ \& \ \sim 2^{src \ \mathbf{mod} \ 16}$

Beschreibung:

Der Befehl `bclr` löscht das durch *src* **mod** 16 gegebene Bit in *dst*.

bcmp ..... Blockvergleich

Adressierungsarten:

src	*dst*	Opcode
	RB	F3

Funktion:

do
  if ([r0] != [r1])
    break
  r0 = r0 + 1
  r1 = r1 + 1
  dst = *dst* - 1
while (*dst* != 0)

Beschreibung:

Der Befehl `bcmp` vergleicht zwei Speicherbereiche wortweise. Die Anfangsadressen der Speicherblöcke müssen beim Aufruf in den Registern r0 bzw. r1 gespeichert sein, die (identische!) Länge der beiden Blöcke im Register ***RB***. Stellt der Befehl einen Unterschied in den Speicherblöcken fest, wird er mit gelöschtem Z-Flag beendet; r0 und r1 zeigen dann auf die ersten Worte der beiden Blöcke, die verschieden sind. Sind die Blöcke identisch, ist nach Beendigung des Befehls das Z-Flag gesetzt.

binit ..... Blockinitialisierung

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	F3

Funktion:

do
  [r0] = *src*
  r0 = r0 + 1
  dst = *dst* - 1
while (*dst* != 0)

Beschreibung:

Der Befehl `binit` initialisiert einen Speicherbereich mit dem Wert des Operanden *src*. Die Anfangsadresse des Speicherblocks muß beim Aufruf im Register r0 gespeichert sein, die Länge des Blocks wird im Operanden *dst* spezifiziert.

binv ..... Bit invertieren

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	70
RA	[addr]	71
RA	[***RB***]	72
RA	disp[***RB***]	73

Funktion:

$dst = dst \hat{} 2^{src}$ **mod** 16

Beschreibung:

Der Befehl `binv` invertiert das durch *src* **mod** 16 gegebene Bit in *dst*.

`bmove` ..... **Blocktransport**

Adressierungsarten:

src	*dst*	Opcode
	RB	F1

Funktion:

do
[r1] = [r0]
r0 = r0 + 1
r1 = r1 + 1
dst = *dst* - 1
while (*dst* != 0)

Beschreibung:

Der Befehl `bmove` dient zum Kopieren größerer Speicherblöcke. Beim Aufruf muß das Register r0 die Adresse des zu kopierenden Speicherblocks enthalten, Register ***RB*** dessen Länge und Register r1 die Zieladresse.

`bscan` ..... **Blocksuche**

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	F2

Funktion:

do
if ([r0] == *src*)
break
r0 = r0 + 1
dst = *dst* - 1
while (*dst* != 0)

Beschreibung:

Der Befehl `bscan` durchsucht einen Speicherblock nach dem ersten Auftreten des Wortes, das im Register ***RA*** gespeichert ist. Der Speicherblock wird definiert durch seine Anfangsadresse, die im Register r0 gespeichert ist und seine Länge, die im Register ***RB***enthalten ist. Findet `bscan` das gesuchte Wort, enthält r0 die zugehörige Adresse und das Z-Flag ist gesetzt. Wird das Wort nicht gefunden, wird das Z-Flag gelöscht.

`bscanf` ..... Bit-Suche vorwärts

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	78
[addr]	***RB***	79
[***RA***]	***RB***	7A
disp[***RA***]	***RB***	7B

Funktion:

Falls *src* != 0
    $dst = \max \{i \in [0,15] \mid (src \;\&\; 2^i) != 0 \}$
sonst
    dst = -1

Beschreibung:

Der Befehl `bscanf` sucht in *src* nach dem am weitesten links stehenden (höchstwertigen) 1-Bit und schreibt dessen Bitposition nach *dst*. Enthält *src* kein 1-Bit, gilt also *src* = 0, so wird *dst* auf den Wert -1 gesetzt.

`bscanr` ..... Bit-Suche rückwärts

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	7C
[addr]	***RB***	7D
[***RA***]	***RB***	7E
disp[***RA***]	***RB***	7F

Funktion:

Falls *src* != 0
    $dst = \min \{i \in [0,15] \mid (src \;\&\; 2^i) != 0 \}$
sonst
    dst = 16

Beschreibung:

Der Befehl `bscanr` sucht in *src* nach dem am weitesten rechts stehenden (niedrigstwertigen) 1-Bit und schreibt dessen Bitposition nach *dst*. Enthält *src* kein 1-Bit, gilt also *src* = 0, so wird *dst* auf den Wert 16 gesetzt.

`bset` ..... Bit setzen

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	68
RA	[addr]	69
RA	[***RB***]	6A
RA	disp[***RB***]	6B

Funktion:

$dst = dst \mid 2^{src} \textbf{ mod } 16$

Beschreibung:

Der Befehl `bset` setzt das durch *src* **mod** 16 gegebene Bit in *dst*.

`btst` ..... **Bit testen**

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	74
RA	[addr]	75
RA	[***RB***]	76
RA	disp[***RB***]	77

Funktion:

$dst \ \& \ 2^{src} \textbf{ mod } 16$

Beschreibung:

Der Befehl `btst` prüft, ob das durch *src* **mod** 16 gegebene Bit in *dst* gesetzt ist und setzt das Z-Flag entsprechend.

`call` ..... **Unterprogrammsprung**

Adressierungsarten:

src	*dst*	Opcode
imm		B9
RA		BA

Funktion:

SP = SP - 1
[SP] = PC
PC = *src*

Beschreibung:

Der Befehl `call` rettet den aktuellen Stand des Programmzählers (d.h. die Adresse des unmittelbar folgenden Befehls; Rückkehradresse) auf den Keller und führt anschließend einen unbedingten Sprung aus. Die Befehlsausführung wird an dem Speicherwort weitergeführt, dessen Adresse durch *src* gegeben ist. Der Befehlszähler wird also mit dem Inhalt des Operanden *src* geladen.

`clc` ..... **Löschen des C-Flags**

Adressierungsarten:

src	*dst*	Opcode
		DE

Funktion:

C-Flag = 0

Beschreibung:

Der Befehl `clc` löscht das C-Flag im Maschinenstatusregister.

`cmc` ..... Invertieren des C-Flags

Adressierungsarten:

src	*dst*	Opcode
		DF

Funktion:

C-Flag = ˜ C-Flag

Beschreibung:

Der Befehl `cmc` invertiert das C-Flag im Maschinenstatusregister.

`cmp` ..... Arithmetischer Vergleich

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	30
[addr]	***RB***	31
[***RA***]	***RB***	32
disp[***RA***]	***RB***	33
RA	***RB***	34
RA	[addr]	35
RA	[***RB***]	36
RA	disp[***RB***]	37

Funktion:

dst - *src*

Beschreibung:

Der Befehl `cmp` subtrahiert die 16-Bit Zahl in *src* von der in *dst* und setzt die Statusflags im Maschinenstatusregister wie folgt:

C = 1 $\Leftrightarrow$ *dst* < *src* (bei Interpretation als vorzeichenlose Zahlen)

Z = 1 $\Leftrightarrow$ *dst* = *src*

N = Bit 15 des Ergebnisses

OVR = 1 $\Leftrightarrow$ (Übertrag in Bit 15 $\neq$ Übertrag aus Bit 15)

Das eigentliche Ergebnis der Subtraktion wird verworfen, *dst* also nicht verändert.

`dec` ..... **Herunterzählen**

Adressierungsarten:

src	*dst*	Opcode
	RB	44
	[addr]	45
	[***RB***]	46
	disp[***RB***]	47

Funktion:

dst = *dst* - 1

Beschreibung:

Der Befehl `dec` zählt die 16-Bit Zahl in *dst* um 1 herunter. Die Statusflags im Maschinenstatusregister werden von diesem Befehl nicht verändert.

`div` ..... **Ganzzahlige vorzeichenbehaftete Division**

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	D5

Funktion:

dst = *dst* / *src*

Beschreibung:

Der Befehl `div` dividiert die vorzeichenbehaftete 16-Bit Zahl in *dst* durch die ebenfalls vorzeichenbehaftete 16-Bit Zahl in *dst* und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

`halt` ..... **Anhalten des Prozessors**

Adressierungsarten:

src	*dst*	Opcode
		DD

Funktion:

Anhalten des Prozessors

Beschreibung:

Der Befehl `halt` hält den Prozessor an, d.h. die normale Befehlsausführung wird gestoppt. Aus dem Halt-Zustand kann der Prozessor nur durch ein externes Reset-Signal befreit werden.

`inc` ..... Hochzählen

Adressierungsarten:

src	*dst*	Opcode
	RB	40
	[addr]	41
	[***RB***]	42
	disp[***RB***]	43

Funktion:

$dst = dst + 1$

Beschreibung:

Der Befehl `inc` zählt die 16-Bit Zahl in *dst* um 1 hoch. Die Statusflags im Maschinenstatusregister werden von diesem Befehl nicht verändert.

`jmp` ..... Unbedingter Sprung

Adressierungsarten:

src	*dst*	Opcode
imm		B0
RA		B4
[addr]		B1
[***RA***]		B2
disp[***RA***]		B3

Funktion:

PC = *src*

Beschreibung:

Der Befehl `jmp` führt einen unbedingten Sprung aus. Die Befehlsausführung wird an dem Speicherwort weitergeführt, dessen Adresse durch *src* gegeben ist. Der Befehlszähler wird also mit dem Inhalt des Operanden *src* geladen.

`jmp cc` ..... Bedingte Sprünge

Adressierungsarten:

src	*dst*	Opcode (binär!)
imm		100ccc01
RA		100ccc10

Die Codierung für die drei mit ccc bezeichneten Bits ist in nachstehender Tabelle definiert.

Funktion:

Falls *cc*:
PC = *src*

Beschreibung:

Die Befehlsbezeichnung `jmp`*cc* steht stellvertretend für die einzelnen Befehle `jmpz`, `jmpnz`, `jmpc`, `jmpnc`, `jmpovr`, `jmpnovr`, `jmplt` und `jmpgt`. Diese Befehle prüfen, ob die Statusflags (Z, C, N, OVR) im Maschinenstatusregister die nachfolgend definierte Bedingung erfüllen:

cc	Codierung	Bedingung
z	000	Z
nz	001	$\overline{Z}$
c	010	C
nc	011	$\overline{C}$
ovr	100	OVR
novr	101	$\overline{OVR}$
lt	110	$N \oplus OVR$
gt	111	$\overline{N \oplus OVR} \wedge \overline{Z}$

Ist die Bedingung erfüllt, wird die Befehlsausführung an dem Speicherwort weitergeführt, dessen Adresse durch *src* gegeben ist. Der Befehlszähler wird also mit dem Inhalt des Operanden *src* geladen. Anderenfalls wird die Befehlsausführung mit dem nachfolgenden Befehl fortgesetzt.

`ldsp` ..... Laden des Kellerzeigers

Adressierungsarten:

src	*dst*	Opcode
RA		5C
imm		58
[addr]		5D
[***RA***]		5E
disp[***RA***]		5F

Funktion:

SP = *src*

Beschreibung:

Der Befehl `ldsp` lädt den Kellerzeiger mit dem Inhalt des Operanden *src*.

`lea` ..... Lade effektive Adresse

Adressierungsarten:

src	*dst*	Opcode
[addr]	***RB***	A1
[***RA***]	***RB***	A2
disp[***RA***]	***RB***	A3

Funktion:

dst = &*src*

Beschreibung:

Der Befehl `lea` lädt die effektive Adresse des Quelloperanden *src* nach *dst*.

`loop` ..... **Schleifenbefehl**

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	A5
RA	***RB***	A6

Funktion:

dst = *dst* - 1
Falls *dst* != 0
    PC = *src*

Beschreibung:

Der Befehl `loop` dient zur einfachen Programmierung von Schleifen. Er zählt zunächst *dst* um eins herunter. Ist *dst* nun nicht Null, wird die Befehlsausführung an dem Speicherwort weitergeführt, dessen Adresse durch *src* gegeben ist. Der Befehlszähler wird also mit dem Inhalt des Operanden *src* geladen. Anderenfalls wird die Befehlsausführung mit dem nachfolgenden Befehl fortgesetzt.

`loopnz` ..... **Bedingter Schleifenbefehl**

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	AD
RA	***RB***	AE

Funktion:

Falls Z-Flag gelöscht
    dst = *dst* - 1
    Falls *dst* != 0
        PC = *src*

Beschreibung:

Der Befehl `loopnz` arbeitet bei gelöschtem Z-Flag wie der Befehl `loop`. Anderenfalls wird die Befehlsausführung mit dem nachfolgenden Befehl fortgesetzt.

`loopz` ..... **Bedingter Schleifenbefehl**

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	A9
RA	***RB***	AA

Funktion:

Falls Z-Flag gesetzt
    dst = *dst* - 1
    Falls *dst* != 0
        PC = *src*

Beschreibung:

Der Befehl `loopz` arbeitet bei gesetztem Z-Flag wie der Befehl `loop`. Anderenfalls wird die Befehlsausführung mit dem nachfolgenden Befehl fortgesetzt.

`lsl` ..... Logisches Schieben nach links

Adressierungsarten:

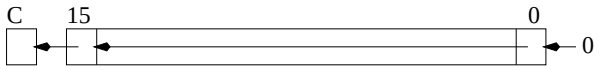
src	*dst*	Opcode
	RB	88
	[addr]	89
	[***RB***]	8A
	disp[***RB***]	8B
RA	***RB***	C4
imm	***RB***	C5

Funktion:

$dst = dst << 1$
$dst = dst << src$

Beschreibung:

Der Befehl `lsl` schiebt den Inhalt von *dst* um eine oder mehrere Bitpositionen nach links, wobei von rechts Nullen in die freiwerdenden (niederwertigen) Bits geschoben werden. Die Anzahl n der Bitpositionen ist für die Formen ohne expliziten Quelloperand 1, ansonsten gilt $n = src$ **mod** 16. Der Befehl schreibt den Inhalt von Bit $16-n$ in das C-Flag und den Inhalt von Bit i (für $i \in [0, 15-n]$) an die Bitposition $i+n$. Die Bits $0...n-1$ werden mit Nullen gefüllt. Das folgende Bild veranschaulicht die Operation graphisch.



`lsr` ..... Logisches Schieben nach rechts

Adressierungsarten:

src	*dst*	Opcode
	RB	84
	[addr]	85
	[***RB***]	86
	disp[***RB***]	87
RA	***RB***	C2
imm	***RB***	C3

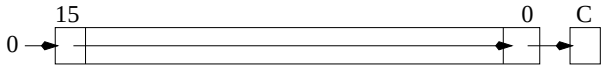
Funktion:

$dst = dst >> 1$
$dst = dst >> (src$ **mod** $16)$

Beschreibung:

Der Befehl `lsr` schiebt den Inhalt von *dst* um eine oder mehrere Bitpositionen nach rechts, wobei von links Nullen in die freiwerdenden (höherwertigen) Bits geschoben werden. Die Anzahl n der

Bitpositionen ist für die Formen ohne expliziten Quelloperand 1, ansonsten gilt $n = src$ **mod** 16. Der Befehl schreibt den Inhalt von Bit $n-1$ in das C-Flag und den Inhalt von Bit i (für $i \in [n, 15]$) an die Bitposition $i - n$. Die Bits $15...16 - n$ werden mit Nullen gefüllt. Das folgende Bild veranschaulicht die Operation graphisch.



`move` ..... Datentransport

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	38
[addr]	***RB***	01
[***RA***]	***RB***	02
disp[***RA***]	***RB***	03
RA	***RB***	04
RA	[addr]	05
RA	[***RB***]	06
RA	disp[***RB***]	07

Funktion:

$dst = src$

Beschreibung:

Der Befehl `move` kopiert ein 16-Bit Wort von *src* nach *dst*.

`mul` ..... Ganzzahlige vorzeichenbehaftete Multiplikation

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	D3

Funktion:

$dst = dst * src$

Beschreibung:

Der Befehl `mul` multipliziert die vorzeichenbehafteten 16-Bit Zahlen in *src* und *dst* und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

`neg` ..... **Vorzeichenwechsel**

Adressierungsarten:

src	*dst*	Opcode
	RB	48
	[addr]	49
	[***RB***]	4A
	disp[***RB***]	4B

Funktion:

dst = 0 - *dst*

Beschreibung:

Der Befehl `neg` berechnet das Zweierkomplement der 16-Bit Zahl in *dst* und speichert das Resultat wieder in *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

`not` ..... **Bitweise Negation**

Adressierungsarten:

src	*dst*	Opcode
	RB	4C
	[addr]	4D
	[***RB***]	4E
	disp[***RB***]	4F

Funktion:

dst = ~*dst*

Beschreibung:

Der Befehl `not` berechnet das bitweise Komplement von *dst* und speichert das Resultat wieder in *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

`or` ..... **Bitweise ODER-Verknüpfung**

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	20
[addr]	***RB***	21
[***RA***]	***RB***	22
disp[***RA***]	***RB***	23
RA	***RB***	24
RA	[addr]	25
RA	[***RB***]	26
RA	disp[***RB***]	27

Funktion:

$dst = dst \mid src$

Beschreibung:

Der Befehl `or` berechnet die bitweise ODER-Verknüpfung von *src* und *dst* und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

`pop` ..... Restaurieren vom Keller

Adressierungsarten:

src	*dst*	Opcode
	RB	DA

Funktion:

dst = [SP]
SP = SP + 1

Beschreibung:

Der Befehl `pop` liest und entfernt den obersten Eintrag vom Keller und speichert in in *dst*.

`push` ..... Retten auf den Keller

Adressierungsarten:

src	*dst*	Opcode
RA		D8
imm		D9

Funktion:

SP = SP - 1
[SP] = *src*

Beschreibung:

Der Befehl `push` schreibt den Inhalt des Quelloperanden *src* in den Keller.

`ret` ..... Unterprogrammrücksprung

Adressierungsarten:

src	*dst*	Opcode
		D6
imm		D7

Funktion:

PC = [SP]
SP = SP + 1 bzw. SP = SP + 1 + *src*

Beschreibung:

Der Befehl `ret` holt die oberste auf dem Keller gesicherte Rückkehradresse wieder in den Befehlszähler zurück, bewirkt also einen Rücksprung aus einem Unterprogramm unmittelbar hinter den aufrufenden `call`-Befehl. Die Befehlsform mit Operand addiert zusätzlich *src* zum Kellerzeiger, um gekellerte Argumente des Unterprogramms vom Keller entfernen zu können.

`rl` ..... Rotieren nach links

Adressierungsarten:

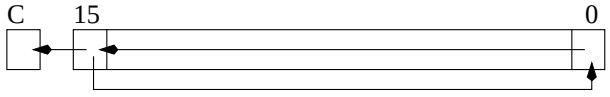
src	*dst*	Opcode
	RB	90
	[addr]	91
	[***RB***]	92
	disp[***RB***]	93
RA	***RB***	C8
imm	***RB***	C9

Funktion:

dst = *dst* um ein Bit nach links rotiert
dst = *dst* um *src* **mod** 16 Bits nach links rotiert

Beschreibung:

Der Befehl `rl` rotiert den Inhalt von *dst* um eine oder mehrere Bitpositionen nach links. Die Anzahl n der Bitpositionen ist für die Formen ohne expliziten Quelloperand 1, ansonsten gilt $n = src$ **mod** 16. Der Befehl schreibt den Inhalt von Bit $16 - n$ in das C-Flag; der ursprüngliche Inhalt von Bit i (für $i \in [0, 15]$) steht nach der Ausführung des Befehls an der Bitposition $i + n$ **mod** 16. Das folgende Bild veranschaulicht die Operation graphisch.



`rlc` ..... Rotieren nach links über C-Flag

Adressierungsarten:

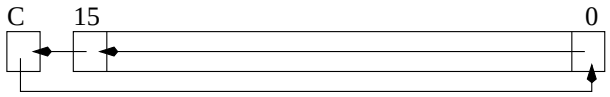
src	*dst*	Opcode
	RB	64
	[addr]	65
	[***RB***]	66
	disp[***RB***]	67

Funktion:

dst = *dst* um ein Bit nach links über C-Flag rotiert

Beschreibung:

Der Befehl `rlc` rotiert den Inhalt von *dst* um eine Bitpositionen nach links, wobei das C-Flag in die Rotation mit einbezogen wird. Der Befehl schreibt den Inhalt von Bit 15 in das C-Flag. den Inhalt von Bit i (für $i \in [0, 14]$) an die Bitposition $i + 1$ und den ursprünglichen Inhalt des C-Flags an die Bitposition 0. Das folgende Bild veranschaulicht die Operation graphisch.



`rr` ..... **Rotieren nach rechts**

Adressierungsarten:

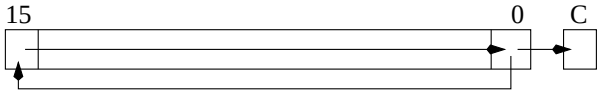
src	*dst*	Opcode
	RB	8C
	[addr]	8D
	[***RB***]	8E
	disp[***RB***]	8F
RA	***RB***	C6
imm	***RB***	C7

Funktion:

dst = *dst* um ein Bit nach rechts rotiert
dst = *dst* um *src* **mod** 16 Bits nach rechts rotiert

Beschreibung:

Der Befehl `rr` rotiert den Inhalt von *dst* um eine oder mehrere Bitpositionen nach rechts. Die Anzahl n der Bitpositionen ist für die Formen ohne expliziten Quelloperand 1, ansonsten gilt $n = src$ **mod** 16. Der Befehl schreibt den Inhalt von Bit $n-1$ in das C-Flag; der ursprüngliche Inhalt von Bit i (für $i \in [0, 15]$) steht nach der Ausführung des Befehls an der Bitposition $(i - n)$ **mod** 16. Das folgende Bild veranschaulicht die Operation graphisch.



`rrc` ..... **Rotieren nach rechts über C-Flag**

Adressierungsarten:

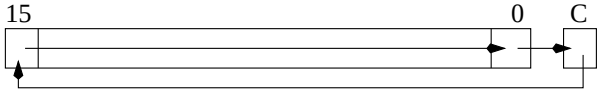
src	*dst*	Opcode
	RB	60
	[addr]	61
	[***RB***]	62
	disp[***RB***]	63

Funktion:

dst = *dst* um ein Bit nach rechts über C-Flag rotiert

Beschreibung:

Der Befehl `rrc` rotiert den Inhalt von *dst* um eine Bitpositionen nach rechts, wobei das C-Flag in die Rotation mit einbezogen wird. Der Befehl schreibt den Inhalt von Bit 0 in das C-Flag, den Inhalt von Bit i (für $i \in [1, 15]$) an die Bitposition $i - 1$ und den ursprünglichen Inhalt des C-Flags an die Bitposition 15. Das folgende Bild veranschaulicht die Operation graphisch.



`stsp` ..... Speichern des Kellerzeigers

Adressierungsarten:

src	*dst*	Opcode
	RB	54
	[addr]	55
	[***RB***]	56
	disp[***RB***]	57

Funktion:

dst = SP

Beschreibung:

Der Befehl `stsp` schreibt den Inhalt des Kellerzeigers nach *dst*.

`sub` ..... Ganzzahlige Subtraktion

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	10
[addr]	***RB***	11
[***RA***]	***RB***	12
disp[***RA***]	***RB***	13
RA	***RB***	14
RA	[addr]	15
RA	[***RB***]	16
RA	disp[***RB***]	17

Funktion:

dst = *dst* - *src*

Beschreibung:

Der Befehl `sub` subtrahiert die 16-Bit Zahl in *src* von der in *dst*. Dabei werden die Statusflags im Maschinenstatusregister wie folgt gesetzt:

C = 1 $\Leftrightarrow$ *dst* < *src* (bei Interpretation als vorzeichenlose Zahlen)

Z = 1 $\Leftrightarrow$ *dst* = *src*

N = Bit 15 des Ergebnisses

OVR = 1 $\Leftrightarrow$ (Übertrag in Bit 15 $\neq$ Übertrag aus Bit 15)

Das Ergebnis der Subtraktion wird nach *dst* geschrieben.

`subc` ..... Ganzzahlige Subtraktion mit Übertrag

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	D1

Funktion:

dst = *dst* - *src* - C-Flag

Beschreibung:

Der Befehl `subc` subtrahiert die 16-Bit Zahl in *src* sowie das C-Flag von der 16-Bit Zahl in *dst*. Die Statusflags im Maschinenstatusregister werden dabei wie folgt gesetzt:

C = 1 $\Leftrightarrow$ *dst* < *src* + C-Flag (bei Interpretation als vorzeichenlose Zahlen)

Z = 1 $\Leftrightarrow$ *dst* = *src*

N = Bit 15 des Ergebnisses

OVR = 1 $\Leftrightarrow$ (Übertrag in Bit 15 $\neq$ Übertrag aus Bit 15)

Das Ergebnis der Subtraktion wird nach *dst* geschrieben.

`tst` ..... Arithmetischer Vergleich auf 0

Adressierungsarten:

src	*dst*	Opcode
	RB	50
	[addr]	51
	[***RB***]	52
	disp[***RB***]	53

Funktion:

dst - 0

Beschreibung:

Der Befehl `tst` setzt die Statusflags im Maschinenstatusregister, so, daß sie dem Wert von *dst* entsprechen. Das Z Flag wird genau dann gesetzt, wenn *dst* = 0 gilt, das S-Flag genau dann, wenn *dst* < 0. Die Flags C und OVR werden gelöscht.

`udiv` ..... Ganzzahlige vorzeichenlose Division

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	D4

Funktion:

dst = *dst* / *src*

Beschreibung:

Der Befehl `udiv` dividiert die vorzeichenlose 16-Bit Zahl in *dst* durch die ebenfalls vorzeichenlose 16-Bit Zahl in *dst* und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

umul ..... Ganzzahlige vorzeichenlose Multiplikation

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	D2

Funktion:

dst = *dst* * *src*

Beschreibung:

Der Befehl `umul` multipliziert die vorzeichenlosen 16-Bit Zahlen in *src* und *dst* und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.

xchg ..... Datenaustausch

Adressierungsarten:

src	*dst*	Opcode
[addr]	***RB***	39
[***RA***]	***RB***	3A
disp[***RA***]	***RB***	3B

Funktion:

dst $\leftrightarrow$ *src*

Beschreibung:

Der Befehl `xchg` tauscht die Inhalte von *src* und *dst* aus.

xlat ..... Tabellengesteuerte Übersetzung

Adressierungsarten:

src	*dst*	Opcode
RA	***RB***	F0

Funktion:

do
  if (([r0] >= [*src*]) && ([r0] <= [*src*+1]))
    [r0] = [*src*+2 + [r0]-[*src*]]
  r0 = r0 + 1
  dst = *dst* - 1
while (*dst* != 0)

Beschreibung:

Der Befehl `xlat` nimmt eine tabellengesteuerte Übersetzung eines Speicherbereichs vor. Die Anfangsadresse des zu übersetzenden Speicherblocks steht im Register r0, die Länge des Blocks im Register ***RB***. Das Register ***RA*** enthält die Adresse einer Übersetzungstabelle mit folgendem Aufbau:

Wort 0: Kleinster zu übersetzender Wert (min)

Wort 1: Größter zu übersetzender Wert (max)

Worte 2 ... 2+max-min: Übersetzungen der Worte min ... max

Der `xlat`-Befehle überschreibt jedes in dem spezifizierten Speicherblock liegende Speicherwort, dessen Inhalt im Intervall [min,max] liegt, mit dem durch den zugehörigen Tabelleneintrag definierten neuen Wert. Worte, deren Inhalt nicht im Intervall [min,max] liegt, bleiben unverändert.

`xor` ..... Bitweise Exklusiv-ODER-Verknüpfung

Adressierungsarten:

src	*dst*	Opcode
imm	***RB***	28
[addr]	***RB***	29
[***RA***]	***RB***	2A
disp[***RA***]	***RB***	2B
RA	***RB***	2C
RA	[addr]	2D
RA	[***RB***]	2E
RA	disp[***RB***]	2F

Funktion:

$dst = dst \hat{} src$

Beschreibung:

Der Befehl `xor` berechnet die bitweise Exklusiv-ODER-Verknüpfung von *src* und *dst* und schreibt das Ergebnis nach *dst*. Die Statusflags im Maschinenstatusregister werden entsprechend dem Ergebnis der Operation besetzt.