Referenz
Elementare Makros Atmel AVR

Stand: 2.32 vom 26. 8. 2013

Das Registermodell

Adresse	Nutzung	Adresse	Nutzung
0	Multiplikationsergebnis Low	16	TEMP. Allgemeines Arbeitsregister
1	Multiplikationsergebnis High	17	
2		18	
3		19	
4		20	C: 3. Operandenregister oder Zählregister
5		21	CH
6		22	B: 2. Operandenregister
7		23	BH
8		24	A. Akkumulator
9		25	AH
10		26	X; XL 1. Adreßregister
11		27	XH
12		28	Y; YL 2. Adreßregister
13		29	YH
14		30	Z; ZL 3. Adreßregister
15		31	ZH

- TEMP ist das allgemeine Arbeits- und Hilfsregister in den Makros. Es darf außerhalb der Makros dann verwendet werden, wenn man weiß, was man tut. Sein Inhalt ist grundsätzlich als verloren anzusehen. TEMP darf kein Parameter eines Makros sein; auf TEMP dürfen nur die üblichen AVR-Befehle angewendet werden.
- A, B, C sind Operationsregister. Die typische Nutzung: A ist der Akkumulator, B nimmt den zweiten Operanden auf, C dient zu Zählzwecken.
- X, Y, Z sind die Adreßregister gemäß AVR-Architektur.
- Anwendungsspezifisch nutzbar (z. B. für globale Variable, Parameterübergabe, Unterbrechungsbehandlung oder Systemsteuerung) sind R2 bis R15 sowie R17, R18 und R19.
- Die Register r0 und r1 sind zusätzliche Arbeits- und Hilfsregister.

Registeradressen

Die Adressen einzelner Register (jeweils 8 Bits) werden als Parameter übergeben. Hierfür gelten die Bezeichnungen gemäß derAtmel-Konvention (r0...r31, xl, xh usw.). Die Makros gelten für alle Register (r0...r31 außer TEMP). Weitere zulässige Bezeichnungen gemäß Registermodell (mit .def-Anweisungen zu deklarieren): temp, a, ah, b, bh, c, ch, x, y, z.

Allgemeine Adressen (general_adrs)

Ist der Wert < 40H oder (bei Einzelbitzugriffen) < 20H, ist es eine E-A-Adresse, und der jeweilige Zugriff wird mit E-A-Befehlen ausgeführt. Ansonsten ist es eine SRAM-Adresse.

Verzweigung und Unterprogrammruf
Der Atmel-Assembler kann nicht automatisch zwischen JMP und RJMP bzw. CALL und RCALL wählen. Manche Schaltkreise unterstützen nur RJMP bzw. RCALL. Die Makrounterstützung erfordertd eshalb folgendes:

- Wenn der Schaltkreis nur RJMP/RCALL unterstützt, ist ganz am Anfang des Hauptprogramms folgende Definition zu setzen:

  #define reljump (0)

- Wenn der Schaltkreis JMP/CALL unterstützt, ist diese Definition wegzulassen.

Mnemonic	Parameter	Wirkung
1. Elementare Transporte		
gst	general_adrs, reg_adrs	Register auf allgemeiner Adresse speichern (Ausgabe)
gdst	general_adrs, reg_adrs, reg_adrs	Zwei Register als 16-Bit-Wort auf allgemeiner Adresse speichen. 1. Register = High Byte, 2. Register = Low Byte
gld	reg_adrs, general_adrs	Register von allgemeiner Adresse laden (Eingabe)
gdld	reg_adrs, reg_adrs, general_adrs	Zwei Register mit 16-Bit-Wort von allgemeiner Adresse laden. 1. Register = High Byte, 2. Register = Low Byte
glit	general_adrs, literal	Direktwert auf allgemeiner Adresse speichern (Ausgabe)
gdlit	general_adrs, 16-bit-literal	Langen Direktwert (16 Bits) auf allgemeiner Adresse und Folgeadresse speichern (Ausgabe)..
gsdlit	general_adrs, general_adrs, 16-bit-literal	Langen Direktwert (16 Bits) byteweise auf zwei allgemeine Adressen (High, Low) speichern (Scatter)
lit	register_adrs, literal	Direktwert in Register laden (betrifft alle 32 Register)
gmov	general_adrs, general_adrs	Von der zweiten allgemeinen Adresse auf die erste allgemeine Adresse transportieren
gdmov	general_adrs, general_adrs	16 Bits von der zweiten allgemeinen Adresse auf die erste allgemeine Adresse transportieren
gclear	general_adrs	Nullen auf allgemeiner Adresse speichern (Löschen)
gdclear	general_adrs	16 Bits auf allgemeiner Adresse löschen
2. Elementare Bitzugriffe		
gbit0	general_adrs, bit_adrs	Bit auf allgemeiner Adresse löschen
bit0	register_adrs, bit_adrs	Bit in Register löschen
gbit1	general_adrs, bit_adrs	Bit auf allgemeiner Adresse setzen
bit1	register_adrs, bit_adrs	Bit in Register setzen
gbitt	general_adrs, bit_adrs	Bit auf allgemeiner Adresse umschalten (toggle)

Mnemonic	Parameter	Wirkung
bitt	register_adrs, bit_adrs	Bit in Register l umschalten (toggle)
gbitz	general_adrs, bit_adrs	ZF nach Bit auf allgemeiner Adresse. Wenn ZF=1, dann Bit=0
bitz	register_adrs, bit_adrs	ZF nach Bit in Register. Wenn ZF=1, dann Bit=0
gbitc	general_adrs, bit_adrs	CF nach Bit auf allgemeiner Adresse. Wenn CF=1, dann Bit=1
bitc	register_adrs, bit_adrs	CF nach Bit in Register. Wenn CF=1, dann Bit=1
3. Tests und Bitabfragen		
gzero	general_adrs	Allgemeine Adresse auf Null testen (ZF)
gdzero	general_adrs	16 Bits auf allgemeiner Adresse auf Null testen (ZF)
zero	register_adrs	Register auf Null testen (ZF). Synonym zu TST
gbit	general_adrs, bit_adrs	Bit auf allgemeiner Adresse testen (ZF). Wenn ZF=1, dann Bit=0
bit	register_adrs, bit_adrs	Bit in Register testen (ZF). Wenn ZF=1, dann Bit=0
gbittoc	general_adrs, bit_adrs	Bit auf allgemeiner Adresse ins Carry Flag transportieren (Bit => CF)
bittoc	register_adrs, bit_adrs	Bit in Register ins Carry Flag transportieren (Bit => CF)
4. Verzweigungen		
brz	branch_adrs	Kurze Verzweigung, wenn ZF=1. Synonym zu BREQ
brnz	branch_adrs	Kurze Verzweigung, wenn ZF=0. Synonym zu BRNE
jp, goto	jump_adrs	Unbedingte Verzweigung. Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpz, jpeq	jump_adrs	Verzweigen, wenn ZF=1 (Gleichheit). Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpnz, jpne	jump_adrs	Verzweigen, wenn ZF=0 (Ungleichheit). Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpb	jump_adrs	Verzweigen, wenn vorzeichenlos kleiner (below; A – B ergibt A < B). Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpbe	jump_adrs	Verzweigen, wenn vorzeichenlos kleiner oder gleich (below or equal; A – B ergibt A <= B). Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpa	jump_adrs	Verzweigen, wenn vorzeichenlos größer (above; A – B ergibt A > B). Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpae	jump_adrs	Verzweigen, wenn vorzeichenlos größer oder gleich (above or equal; A – B ergibt A >= B). Wahl zwischen zwischen JMP und RJMP
jpl	jump_adrs	Verzweigen, wenn vorzeichenbehaftet kleiner (less; A – B ergibt A < B). Wahl zwischen zwischen JMP und RJMP

Mnemonic	Parameter	Wirkung
jple	jump_adrs	Verzweigen, wenn vorzeichenbehaftet kleiner oder gleich (less or equal; A – B ergibt A <= B). Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpg	jump_adrs	Verzweigen, wenn vorzeichenbehaftet größer (greater; A – B ergibt A > B). Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpge	jump_adrs	Verzweigen, wenn vorzeichenbehaftet größer oder gleich (greater or equal; A – B ergibt A >= B). Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpg0	general_adrs, bit_adrs, jump_adrs	Verzweigen, wenn Bit auf allgemeiner Adresse = 0. Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jpg1	general_adrs, bit_adrs, jump_adrs	Verzweigen, wenn Bit auf allgemeiner Adresse = 1. Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jp0	register_adrs, bit_adrs, jump_adrs	Verzweigen, wenn Bit in Register = 0. Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
jp1	register_adrs, bit_adrs, jump_adrs	Verzweigen, wenn Bit in Register = 1. Wahl zwischen zwischen JMP und RJMP gemäß Definition reljump(0)
5. Ein Befehlswort überspringen (16 Bits)		
skipz, skipeq		Ein Befehlswort überspringen, wenn ZF=1 (Gleichheit)
skipnz, skipne		Ein Befehlswort überspringen, wenn ZF=0 (Ungleichheit)
skipb		Ein Befehlswort überspringen, wenn vorzeichenlos kleiner (below; A – B ergibt A < B)
skipbe		Ein Befehlswort überspringen, wenn vorzeichenlos kleiner oder gleich (below or equal; A – B ergibt A <= B)
skipa		Ein Befehlswort überspringen, wenn vorzeichenlos größer (above; A – B ergibt A > B)
skipae		Ein Befehlswort überspringen, wenn vorzeichenlos größer oder gleich (above or equal; A – B ergibt A >= B)
skipl		Ein Befehlswort überspringen, wenn vorzeichenbehaftet kleiner (less; A – B ergibt A < B)
skiple		Zwei Befehlswörter überspringen, wenn vorzeichenbehaftet kleiner oder gleich (less or equal; A – B ergibt A <= B)
skipg		Ein Befehlswort überspringen, wenn vorzeichenbehaftet größer (greater; A – B ergibt A > B)
skipge		Ein Befehlswort überspringen, wenn vorzeichenbehaftet größer oder gleich (greater or equal; A – B ergibt A >= B)
skipg0	general_adrs, bit_adrs	Ein Befehlswort überspringen, wenn Bit auf allgemeiner Adresse = 0

Mnemonic	Parameter	Wirkung
skipg1	general_adrs, bit_adrs	Ein Befehlswort überspringen, wenn Bit auf allgemeiner Adresse = 1
6. Zwei Befehlswörter überspringen (32 Bits)		
skipz2, skipeq2		Zwei Befehlswörter überspringen, wenn ZF=1 (Gleichheit)
skipnz2, skipne2		Zwei Befehlswörter überspringen, wenn ZF=0 (Ungleichheit)
skipb2		Zwei Befehlswörter überspringen, wenn vorzeichenlos kleiner (below; A – B ergibt A < B)
skipbe2		Zwei Befehlswörter überspringen, wenn vorzeichenlos kleiner oder gleich (below or equal; A – B ergibt A <= B)
skipa2		Zwei Befehlswörter überspringen, wenn vorzeichenlos größer (above; A – B ergibt A > B)
skipae2		Zwei Befehlswörter überspringen, wenn vorzeichenlos größer oder gleich (above or equal; A – B ergibt A >= B)
skipl2		Zwei Befehlswörter überspringen, wenn vorzeichenbehaftet kleiner (less; A – B ergibt A < B)
skiple2		Zwei Befehlswörter überspringen, wenn vorzeichenbehaftet kleiner oder gleich (less or equal; A – B ergibt A <= B)
skipg2		Zwei Befehlswörter überspringen, wenn vorzeichenbehaftet größer (greater; A – B ergibt A > B)
skipge2		Zwei Befehlswörter überspringen, wenn vorzeichenbehaftet größer oder gleich (greater or equal; A – B ergibt A >= B)
skipg02	general_adrs, bit_adrs	Zwei Befehlswörter überspringen, wenn Bit auf allgemeiner Adresse = 0
skipg12	general_adrs, bit_adrs	Zwei Befehlswörter überspringen, wenn Bit auf allgemeiner Adresse = 1
7. Einen beliebig langen Folgebefehl überspringen (diese Makros nutzen das temp-Register)		
skipfz, skipfeq		Den Folgebefehl überspringen, wenn ZF=1 (Gleichheit)
skipfnz, skipfne		Den Folgebefehl überspringen, wenn ZF=0 (Ungleichheit)
skipfc		Den Folgebefehl überspringen, wenn CF=1 (Ausgangsübertrag)
skipfnc		Den Folgebefehl überspringen, wenn CF=0 (kein Ausgangsübertrag)
skipfn		Den Folgebefehl überspringen, wenn negativ (N=1)
skipfp		Den Folgebefehl überspringen, wenn positiv (N=0)
skipfo		Den Folgebefehl überspringen, wenn Überlauf (V=1)

Mnemonic	Parameter	Wirkung
skipfno		Den Folgebefehl überspringen, wenn kein Überlauf (V=0)
skipfs		Den Folgebefehl überspringen, wenn vorzeichenbehaftet kleiner (S=1)
skipfns		Den Folgebefehl überspringen, wenn vorzeichenbehaftet größer oder gleich (S=0) SF=0
skipfh		Den Folgebefehl überspringen, wenn Halbbyteübertrag (H=1)
skipfnh		Den Folgebefehl überspringen, wenn kein Halbbyteübertrag (H=0)
skipft		Den Folgebefehl überspringen, wenn T=1
skipfnt		Den Folgebefehl überspringen, wenn T=0
skipfe		Den Folgebefehl überspringen, wenn Interrupt erlaubt (IF=1)
skipfd		Den Folgebefehl überspringen, wenn Interrupt verhindert (IF=0)
skipfb		Den Folgebefehl überspringen, wenn vorzeichenlos kleiner (below; A – B ergibt A < B)
skipfbe		Den Folgebefehl überspringen, wenn vorzeichenlos kleiner oder gleich (below or equal; A – B ergibt A <= B)
skipfa		Den Folgebefehl überspringen, wenn vorzeichenlos größer (above; A – B ergibt A > B)
skipfae		Den Folgebefehl überspringen, wenn vorzeichenlos größer oder gleich (above or equal; A – B ergibt A >= B)
skipfl		Den Folgebefehl überspringen, wenn vorzeichenbehaftet kleiner (less; A – B ergibt A < B)
skipfle		Den Folgebefehl überspringen, wenn vorzeichenbehaftet kleiner oder gleich (less or equal; A – B ergibt A <= B)
skipfg		Den Folgebefehl überspringen, wenn vorzeichenbehaftet größer (greater; A – B ergibt A > B)
skipfge		Den Folgebefehl überspringen, wenn vorzeichenbehaftet größer oder gleich (greater or equal; A – B ergibt A >= B)
skipfg0	general_adrs, bit_adrs	Den Folgebefehl überspringen, wenn Bit auf allgemeiner Adresse = 0
skipfg1	general_adrs, bit_adrs	Den Folgebefehl überspringen, wenn Bit auf allgemeiner Adresse = 1
8. Unterprogrammruf und Rückkehr		
cal, gosub	subroutine_adrs	Unterprogrammruf. Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callz, calleq	subroutine_adrs	Unterprogrammruf, wenn ZF=1 (Gleichheit). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)

Mnemonic	Parameter	Wirkung
callnz, callne	subroutine_adrs	Unterprogrammruf, wenn ZF=0 (Ungleichheit). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callb	subroutine_adrs	Unterprogrammruf, wenn vorzeichenlos kleiner (below; A – B ergibt A < B). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callbe	subroutine_adrs	Unterprogrammruf, wenn vorzeichenlos kleiner oder gleich (below or equal; A – B ergibt A <= B). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
calla	subroutine_adrs	Unterprogrammruf, wenn vorzeichenlos größer (above; A – B ergibt A > B). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callae	subroutine_adrs	Unterprogrammruf, wenn vorzeichenlos größer oder gleich (above or equal; A – B ergibt A >= B). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
calll	subroutine_adrs	Unterprogrammruf, wenn vorzeichenbehaftet kleiner (less; A – B ergibt A < B). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callle	subroutine_adrs	Unterprogrammruf, wenn vorzeichenbehaftet kleiner oder gleich (less or equal; A – B ergibt A <= B). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callg	subroutine_adrs	Unterprogrammruf, wenn vorzeichenbehaftet größer (greater; A – B ergibt A > B). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callge	subroutine_adrs	Unterprogrammruf, wenn vorzeichenbehaftet größer oder gleich (greater or equal; A – B ergibt A >= B). Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callg0	general_adrs, bit_adrs, subroutine_adrs	Unterprogrammruf, wenn Bit auf allgemeiner Adresse = 0. Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
callg1	general_adrs, bit_adrs, subroutine_adrs	Unterprogrammruf, wenn Bit auf allgemeiner Adresse = 1. Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
call0	register_adrs, bit_adrs, subroutine_adrs	Unterprogrammruf, wenn Bit in Register = 0. Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
call1	register_adrs, bit_adrs, subroutine_adrs	Unterprogrammruf, wenn Bit in Register = 1. Wahl zwischen zwischen CALL und RCALL gemäß Definition reljump(0)
retz, reteq		Rückkehr, wenn ZF=1 (Gleichheit)
retnz, retne		Rückkehr, wenn ZF=0 (Ungleichheit)
retb		Rückkehr, wenn vorzeichenlos kleiner (below; A – B ergibt A < B)
retbe		Rückkehr, wenn vorzeichenlos kleiner oder gleich (below or equal; A – B ergibt A <= B)

Mnemonic	Parameter	Wirkung
reta		Rückkehr, wenn vorzeichenlos größer (above; A – B ergibt A > B)
retae		Rückkehr, wenn vorzeichenlos größer oder gleich (above or equal; A – B ergibt A >= B)
retl		Rückkehr, wenn vorzeichenbehaftet kleiner (less; A – B ergibt A < B)
retle		Rückkehr, wenn vorzeichenbehaftet kleiner oder gleich (less or equal; A – B ergibt A <= B)
retg		Rückkehr, wenn vorzeichenbehaftet größer (greater; A – B ergibt A > B)
retge		Rückkehr, wenn vorzeichenbehaftet größer oder gleich (greater or equal; A – B ergibt A >= B)
retg0	general_adrs, bit_adrs	Rückkehr, wenn Bit auf allgemeiner Adresse = 0
retg1	general_adrs, bit_adrs	Rückkehr, wenn Bit auf allgemeiner Adresse =
ret0	register_adrs, bit_adrs	Rückkehr, wenn Bit in Register = 0
ret1	register_adrs, bit_adrs	Rückkehr, wenn Bit in Register = 1
9. Zugriffe im Speicheradreßraum mit Offsetadressen (relativ zum Y-Register)		
litof	offset, literal	Direktwert speichern oder ausgeben
litdof	offset, literal	Langen Direktwert speichern oder ausgeben
ldof	register_adrs, offset	Register laden
stof	offset, register_adrs	Register speichern
ldaof	offset	A-Register laden (16 Bits)
ldbof	offset	B-Register laden (16 Bits)
ldcof	offset	C-Register laden (16 Bits)
ldxof	offset	X-Register laden (16 Bits)
ldzof	offset	Z-Register laden (16 Bits)
staof	offset	A-Register speichern (16 Bits)
stbof	offset	B-Register speichern (16 Bits)
stcof	offset	C-Register speichern (16 Bits)
clearof	offset	Byte löschen
zerof	offset	Inhalt auf Null testen (ZF)
bitof0	offset, bit_adrs	Bit setzen
bitof1	offset, bit_adrs	Bit löschen
bitof	offset, bit_adrs	Bit testen (ZF)

Mnemonic	Parameter	Wirkung
skipof0	offset, bit_adrs	Ein Befehlswort überspringen, wenn Bit=0
skipof1	offset, bit_adrs	Ein Befehlswort überspringen, wenn Bit=1
skipof20	offset, bit_adrs	Zwei Befehlswörter überspringen, wenn Bit=0
skipof21	offset, bit_adrs	Zwei Befehlswörter überspringen, wenn Bit=1
skipoff0	offset, bit_adrs	Den Folgebefehl überspringen, wenn Bit=0
skipoff1	offset, bit_adrs	Den Folgebefehl überspringen, wenn Bit=1
10. Bytezugriffe im Speicheradreßraum mit indirekter Adressierung (alle Register r0...r31)		
ldr	register_adrs,SRAM_adrs	Register indirekt laden
ldincr	register_adrs,SRAM_adrs	Register indirekt laden. Danach Adresse erhöhen
lddecr	register_adrs,SRAM_adrs	Adresse vermindern. Danach Register indirekt laden
str	SRAM_adrs,register_adrs	Register indirekt speichern
stincr	SRAM_adrs,register_adrs	Register indirekt speichern. Danach Adresse erhöhen
stdecr	SRAM_adrs,register_adrs	Adresse vermindern. Danach Register indirekt speichern
ldrof	register_adrs,SRAM_adrs, offset_literal	Register indirekt laden. 16-Bit-Offset (Direktwert) wird zur Adresse addiert.
ldrofs	register_adrs,SRAM_adrs, offset_adrs	Register indirekt laden. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert.
strof	SRAM_adrs,offset_literal, register_adrs	Register indirekt speichern. 16-Bit-Offset (Direktwert) wird zur Adresse addiert.
strofs	SRAM_adrs,offset_adrs, register_adrs	Register indirekt speichern. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
11. Wortzugriffe im Speicheradreßraum mit indirekter Adressierung (nur Register A, B, C, X, Y, Z)		
ldda	SRAM_adrs	A-Register indirekt laden
lddinca	SRAM_adrs	A-Register indirekt laden. Danach Adresse erhöhen
ldddeca	SRAM_adrs	Adresse vermindern. Danach A-Register indirekt laden
stda	SRAM_adrs	A-Register indirekt speichern
stdinca	SRAM_adrs	A-Register indirekt speichern. Danach Adresse erhöhen
stddeca	SRAM_adrs	Adresse vermindern. Danach A-Register indirekt speichern
lddofa	SRAM_adrs,offset_literal	A-Register indirekt laden. 16-Bit-Offset (Direktwert) wird zur Adresse addiert
lddofsa	SRAM_adrs,offset_adrs	A-Register indirekt laden. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
stdofa	SRAM_adrs,offset_literal	A-Register indirekt speichern. 16-Bit-Offset (Direktwert) wird zur Adresse addiert

Mnemonic	Parameter	Wirkung
stdofsa	SRAM_adrs,offset_adrs	A-Register indirekt speichern. 16-Bit-Offset wird zur Adresse addiert
lddb	SRAM_adrs	B-Register indirekt laden
lddincb	SRAM_adrs	B-Register indirekt laden. Danach Adresse erhöhen
ldddecb	SRAM_adrs	Adresse vermindern. Danach B-Register indirekt laden
stdb	SRAM_adrs	B-Register indirekt speichern
stdincb	SRAM_adrs	B-Register indirekt speichern. Danach Adresse erhöhen
stddecb	SRAM_adrs	Adresse vermindern. Danach B-Register indirekt speichern
lddofb	SRAM_adrs,offset_literal	B-Register indirekt laden. 16-Bit-Offset (Direktwert) wird zur Adresse addiert
lddofsb	SRAM_adrs,offset_adrs	B-Register indirekt laden. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
stdofb	SRAM_adrs,offset_literal	B-Register indirekt speichern. 16-Bit-Offset (Direktwert) wird zur Adresse addiert
stdofsb	SRAM_adrs,offset_adrs	B-Register indirekt speichern. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
lddc	SRAM_adrs	C-Register indirekt laden
lddincc	SRAM_adrs	C-Register indirekt laden. Danach Adresse erhöhen
ldddecc	SRAM_adrs	Adresse vermindern. Danach A-Register indirekt laden
stdc	SRAM_adrs	C-Register indirekt speichern
stdincc	SRAM_adrs	C-Register indirekt speichern. Danach Adresse erhöhen
stddec	SRAM_adrs	Adresse vermindern. Danach C-Register indirekt speichern
lddofc	SRAM_adrs,offset_literal	C-Register indirekt laden. 16-Bit-Offset (Direktwert) wird zur Adresse addie
lddofsc	SRAM_adrs,offset_adrs	C-Register indirekt laden. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
stdofc	SRAM_adrs,offset_literal	C-Register indirekt speichern. 16-Bit-Offset (Direktwert) wird zur Adresse addiert
stdofsc	SRAM_adrs,offset_adrs	C-Register indirekt speichern. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
lddx	SRAM_adrs	X-Register indirekt laden
lddincx	SRAM_adrs	X-Register indirekt laden. Danach Adresse erhöhen
ldddecx	SRAM_adrs	Adresse vermindern. Danach X-Register indirekt laden
stdx	SRAM_adrs	X-Register indirekt speichern
stddincx	SRAM_adrs	X-Register indirekt speichern. Danach Adresse erhöhen

Mnemonic	Parameter	Wirkung
stddecx	SRAM_adrs	Adresse vermindern. Danach X-Register indirekt speichern
lddofx	SRAM_adrs,offset_literal	A-Register indirekt laden. 16-Bit-Offset (Direktwert) wird zur Adresse addiert.
lddofsx	SRAM_adrs,offset_adrs	X-Register indirekt laden. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert.
stdofx	SRAM_adrs,offset_literal	X-Register indirekt speichern. 16-Bit-Offset (Direktwert) wird zur Adresse addiert.
stdofsx	SRAM_adrs,offset_adrs	X-Register indirekt speichern. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert.
lddy	SRAM_adrs	Y-Register indirekt laden
lddincy	SRAM_adrs	Y-Register indirekt laden. Danach Adresse erhöhen
ldddecy	SRAM_adrs	Adresse vermindern. Danach Y-Register indirekt laden
stdy	SRAM_adrs	Y-Register indirekt speichern
stdincy	SRAM_adrs	Y-Register indirekt speichern. Danach Adresse erhöhen
stddecy	SRAM_adrs	Adresse vermindern. Danach Y-Register indirekt speichern
lddofy	SRAM_adrs,offset_literal	Y-Register indirekt laden. 16-Bit-Offset (Direktwert) wird zur Adresse addiert
lddofsy	SRAM_adrs,offset_adrs	Y-Register indirekt laden. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
stdofy	SRAM_adrs,offset_literal	Y-Register indirekt speichern. 16-Bit-Offset (Direktwert) wird zur Adresse addiert
stdofsy	SRAM_adrs,offset_adrs	Y-Register indirekt speichern. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
lddz	SRAM_adrs	Z-Register indirekt laden
lddincz	SRAM_adrs	Z-Register indirekt laden. Danach Adresse erhöhen
ldddecz	SRAM_adrs	Adresse vermindern. Danach Z-Register indirekt laden
stdz	SRAM_adrs	Z-Register indirekt speichern
stdincz	SRAM_adrs	Z-Register indirekt speichern. Danach Adresse erhöhen
stddecz	SRAM_adrs	Adresse vermindern. Danach Z-Register indirekt speichern
lddofz	SRAM_adrs,offset_literal	Z-Register indirekt laden. 16-Bit-Offset (Direktwert) wird zur Adresse addiert
lddofsz	SRAM_adrs,offset_adrs	Z-Register indirekt laden. 16-Bit-Offset (aus SRAM) wird zur Adresse addiert
stdofz	SRAM_adrs,offset_literal	Z-Register indirekt speichern. 16-Bit-Offset (Direktwert) wird zur Adresse addiert
stdofsz	SRAM_adrs,offset_adrs	Z-Register indirekt speichern.16-Bit-Offset (aus SRAM) wird zur Adresse addiert

Mnemonic	Parameter	Wirkung
12. Wortregister löschen (nur Register A, B, C, X, Y, Z)		
cleara		A-Register löschen (16 Bits)
clearb		B-Register löschen (16 Bits)
clearc		C-Register löschen (16 Bits)
clearx		X-Register löschen (16 Bits)
cleary		Y-Register löschen (16 Bits)
clearz		Z-Register löschen (16 Bits)
13. Worttransporte (nur Register A, B, C, X, Y, Z)		
lita	16-bit-literal	Langen Direktwert ins A-Register laden
litb	16-bit-literal	Langen Direktwert ins B-Register laden
litc	16-bit-literal	Langen Direktwert ins C-Register laden
litx	16-bit-literal	Langen Direktwert ins X-Register laden
lity	16-bit-literal	Langen Direktwert ins Y-Register laden
litz	16-bit-literal	Langen Direktwert ins Z-Register laden
textadrs	16-bit-literal	Zeichenadresse (Flash) ins Z-Register laden
lda	general_adrs	A-Register laden (16 Bits)
ldb	general_adrs	B-Register laden (16 Bits)
ldc	general_adrs	C-Register laden (16 Bits)
ldx	general_adrs	X-Register laden (16 Bits)
ldy	general_adrs	Y-Register laden (16 Bits)
ldz	general_adrs	Z-Register laden (16 Bits)
sta	general_adrs	A-Register speichern (16 Bits)
stb	general_adrs	B-Register speichern (16 Bits)
stc	general_adrs	C-Register speichern (16 Bits)
stx	general_adrs	X-Register speichern (16 Bits)
sty	general_adrs	Y-Register speichern (16 Bits)
stz	general_adrs	Z-Register speichern (16 Bits)
14. Direktwertoperationen (16 Bits)		
addlita	16-bit-literal	Addieren Direktwert zum A-Register
sublita	16-bit-literal	Subtrahieren Direktwert vom A-Register
complita	16-bit-literal	Vergleichen Direktwert mit A-Register
andlita	16-bit-literal	UND-Verknüpfung Direktwert mit A-Register

Mnemonic	Parameter	Wirkung
orlita	16-bit-literal	ODER-Verknüpfung Direktwert mit A-Register
xorlita	16-bit-literal	XOR-Verknüpfung Direktwert mit A-Register
zeroa		A-Register auf Null testen (ZF)
addlitb	16-bit-literal	Addieren Direktwert zum B-Register
sublitb	16-bit-literal	Subtrahieren Direktwert vom B-Register
complitb	16-bit-literal	Vergleichen Direktwert mit B-Register
andlitb	16-bit-literal	UND-Verknüpfung Direktwert mit B-Register
orlitb	16-bit-literal	ODER-Verknüpfung Direktwert mit B-Register
xorlitb	16-bit-literal	XOR-Verknüpfung Direktwert mit B-Register
zerob		B-Register auf Null testen (ZF)
adlitc	16-bit-literal	Addieren Direktwert zum C-Register
sublitc	16-bit-literal	Subtrahieren Direktwert vom C-Register
complitc	16-bit-literal	Vergleichen Direktwert mit C-Register
andlitc	16-bit-literal	UND-Verknüpfung Direktwert mit C-Register
orlitc	16-bit-literal	ODER-Verknüpfung Direktwert mit C-Register
xorlitc	16-bit-literal	XOR-Verknüpfung Direktwert mit C-Register
zeroc		C-Register auf Null testen (ZF)
adlitx	16-bit-literal	Addieren Direktwert zum X-Register
sublitx	16-bit-literal	Subtrahieren Direktwert vom X-Register
complitx	16-bit-literal	Vergleichen Direktwert mit X-Register
zerox		X-Register auf Null testen (ZF)
addlity	16_bit_literal	Addieren Direktwert zum Y-Register
sublity	16_bit_literal	Subtrahieren Direktwert vom Y-Register
complity	16_bit_literal	Vergleichen Direktwert mit Y-Register
zeroy		Y-Register auf Null testen (ZF).
addlitz	16_bit_literal	Addieren Direktwert zum Z-Register.
sublitz	16_bit_literal	Subtrahieren Direktwert vom Z-Register
complitz	16_bit_literal	Vergleichen Direktwert mit Z-Register
zeroz		Z-Register auf Null testen (ZF).
15. Allgemeine Direktwertoperationen (8 Bits) mit allen Registern		
addlit	register_adrs,literal	<Register> := <Register> + Literal
adclit	register_adrs,literal	<Register> := <Register> + Literal + CF

Mnemonic	Parameter	Wirkung
sublit	register_adrs,literal	<Register> := <Register> – Literal
sbclit	register_adrs,literal	<Register> := <Register> – Literal – CF
cplit	register_adrs,literal	Vergleich <Register> – Literal
cpclit	register_adrs,literal	Vergleich <Register> – Literal – CF
andlit	register_adrs,literal	<Register> := <Register> AND Literal
orlit	register_adrs,literal	<Register> := <Register> OR Literal
xorlit	register_adrs,literal	<Register> := <Register> XOR Literal
16. Spezielle Rechenoperationen		
absreg	register_adrs	<Register> := ABS <Register> (Absolutwert (Zweierkomplement))
17. Allgemeine Rechenoperationen (16 Bits)		
addab		<A> := <A> + <B>
subab		<A> := <A> – <B>
compab		Vergleich <A> – <B>
andab		<A> := <A> AND <B>
orab		<A> := <A> OR <B>
xorab		<A> := <A> XOR <B>
addac		<A> := <A> + <C>
subac		<A> := <A> – <C>
compac		Vergleich <A> – <C>
andac		<A> := <A> AND <C>
orac		<A> := <A> OR <C>
xorac		<A> := <A> XOR <C>
addxa		<X> := <X> + <A>
subxa		<X> := <X> – <A>
compxa		Vergleich <X> – <A>
addya		<Y> := <Y> + <A>
subya		<Y> := <Y> – <A>
compya		Vergleich <Y> – <A>
addza		<Z> := <Z> + <A>
subza		<Z> := <Z> – <A>
compza		Vergleich <Z> – <A>

Mnemonic	Parameter	Wirkung
lsla		A-Registerinhalt um 1 Bitposition nach links verschieben
lslaf		A-Registerinhalt um 1 Bitposition nach links verschieben. Mit Eins auffüllen.
lsra		A-Registerinhalt um 1 Bitposition nach rechts verschieben
lsraf		A-Registerinhalt um 1 Bitposition nach rechts verschieben. Mit Eins auffüllen.
asra		A-Registerinhalt um 1 Bitposition arithmetisch nach rechts verschieben
rola		A-Registerinhalt um 1 Bitposition linksherum rotieren (16 Bits)
rora		A-Registerinhalt um 1 Bitposition rechtsherum rotieren (16 Bits)
nega		Das Vorzeichen des A-Registerinhaltes wechseln (Zweierkomplement)
absa		<A> := ABS <A> (Absolutwert (Zweierkomplement))
18. Rechenoperationen 16 Bits mit 8 Bits (Wort mit Byte)		
addaby	register_adrs	<A> := <A> + <Register>
subaby	register_adrs	<A> := <A> – <Register>
addxby	register_adrs	<X> := <X> + <Register>
subxby	register_adrs	<X> := <XA> – <Register>
addyby	register_adrs	<Y> := <Y> + <Register>
subyby	register_adrs	<Y> := <Y> – <Register>
addzby	register_adrs	<Z> := <Z> + <Register>
subzby	register_adrs	<Z> := <Z> – <Register>
19. Spezialitäten		
memlitz	literal	Direktwert speichern gemäß Adresse in Z. Adresse wird um 1 erhöht.
memdlitz	16_bit_literal	Langen Direktwert speichern gemäß Adresse in Z. Adresse wird um 2 erhöht.
incmem	SRAM_adrs	Das adressierte Byte im SRAM um 1 erhöhen
decmem	SRAM_adrs	Das adressierte Byte im SRAM um 1 vermindern
incdmem	SRAM_adrs, 16_bit_literal	Addieren langen Direktwert zu Speicherinhalt. Speicher + Direktwert. Ergebnis auch im A-Register
decdmem	SRAM_adrs, 16_bit_literal	Subtrahieren langen Direktwert von Speicherinhalt. Speicher – Direktwert. Ergebnis auch im A-Register
compmem	SRAM_adrs, literal	Byte im SRAM mit Festwert vergleichen (Byte – Festwert)

Mnemonic	Parameter	Wirkung
compdmem	SRAM_adrs, literal	Wort im SRAM mit langem Festwert vergleichen (Wort – Festwert)
compa	SRAM_adrs	Speicherinhalt mit A-Register vergleichen (8 Bits; Speicher – A).
compda	SRAM_adrs	Speicherinhalt mit A-Register vergleichen (16 Bits; Speicher – A).
pointx	SRAM_adrs, SRAM_adrs	Adreßzeiger im X-Register aufbauen. Wert der ersten Adresse (Anfangsadresse) + Inhalt der zweiten (aktueller Zeiger)
pointy	SRAM_adrs, SRAM_adrs	Adreßzeiger im Y-Register aufbauen. Wert der ersten Adresse (Anfangsadresse) + Inhalt der zweiten (aktueller Zeiger)
pointz	SRAM_adrs, SRAM_adrs	Adreßzeiger im Z-Register aufbauen. Wert der ersten Adresse (Anfangsadresse) + Inhalt der zweiten (aktueller Zeiger)
20. Makros zur Zeitverzögerung		
microsecs	16_bit_literal	Zeitverzögerung um n Mikrosekunden
millisecs	16_bit_literal	Zeitverzögerung um n Millisekunden
21. Makros zum Retten und Wiedereinstellen		
pushf		Das Zustandsregister (Flagregister) auf den Stack legen
popf		Das Zustandsregister (Flagregister) vom Stack holen
pushft		Das Zustandsregister (Flagregister) und das Register temp auf den Stack legen
popft		Das Zustandsregister (Flagregister) und das Register temp vom Stack holen
pushft10		Das Zustandsregister (Flagregister), das Register temp und die Register r1, r0 auf den Stack legen
popft10		Das Zustandsregister (Flagregister), das Register temp und die Register r1, r0 vom Stack holen
pushregs		Alle zum Registermodell gehörenden Register auf den Stack legen
popregs		Alle zum Registermodell gehörenden Register vom Stack holen
22. Makros zur Stackmaschinenemulation		
pusha		Das A-Register auf den Stack legen (16 Bits)
pushb		Das B-Register auf den Stack legen (16 Bits)
pushc		Das C-Register auf den Stack legen (16 Bits)
pushx		Das X-Register auf den Stack legen (16 Bits)
pushy		Das Y-Register auf den Stack legen (16 Bits)
pushz		Das Z-Register auf den Stack legen (16 Bits)
popa		Das A-Register vom Stack holen (16 Bits)

Mnemonic	Parameter	Wirkung
popb		Das B-Register vom Stack holen (16 Bits)
popc		Das C-Register vom Stack holen (16 Bits)
popx		Das X-Register vom Stack holen (16 Bits)
popy		Das Y-Register vom Stack holen (16 Bits)
popz		Das Z-Register vom Stack holen (16 Bits)
pushlit		Einen Direktwert (8 Bits) auf den Stack legen
pushdlit		Einen Direktwert (16 Bits) auf den Stack legen
fetch		Adresse vom Stack holen. Damit ein Byte lesen und auf den Stack legen. Das X-Register enthält die um 1 erhöhte Adresse (Folgeadresse)
fetchd		Adresse vom Stack holen. Damit 2 Bytes lesen und auf den Stack legen. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
store		Auf dem Stack liegt zuoberst (TOS) das Datenbyte, darunter die Adresse. Beides vom Stack nehmen und das Datenbyte auf der Adresse speichern. Das X-Register enthält die um 1 erhöhte Adresse (Folgeadresse)
stored		Auf dem Stack liegt zuoberst (TOS) das Datenwort (16 Bits), darunter die Adresse. Beides vom Stack nehmen und das Datenwort auf der Adresse speichern. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
drop		Ein Byte vom Stack entfernen
dropd		Zwei Bytes vom Stack entfernen
swaps		Die beiden obersten Bytes auf dem Stack tauschen ihre Plätze. Das neue oberste Byte (TOS) wird zudem ins A-Register geladen
swapsd		Die beiden obersten 16-Bit-Wörterauf dem Stack tauschen ihre Plätze. Das neue oberste Wort (TOS) wird zudem ins A-Register geladen
dup		Das Byte auf der zweiten Stackposition wird als Kopie auf den Stack gelegt. Das neue oberste Byte (TOS) wird zudem ins A-Register geladen
dupd		Das 16-Bit-Wort auf der zweiten Stackposition wird als Kopie auf den Stack gelegt. Das neue oberste Wort (TOS) wird zudem ins A-Register geladen
fetcha		Adresse vom Stack holen. Damit ein Byte ins A-Register laden. Das X-Register enthält die um 1 erhöhte Adresse (Folgeadresse)
fetchad		Adresse vom Stack holen. Damit ein Wort (16 Bits) ins A-Register laden. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)

Mnemonic	Parameter	Wirkung
fetchb		Adresse vom Stack holen. Damit ein Byte ins B-Register laden. Das X-Register enthält die um 1 erhöhte Adresse (Folgeadresse)
fetchbd		Adresse vom Stack holen. Damit ein Wort (16 Bits) ins B-Register laden. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
fetchc		Adresse vom Stack holen. Damit ein Byte ins C-Register laden. legen. Das X-Register enthält die um 1 erhöhte Adresse (Folgeadresse)
fetchcd		Adresse vom Stack holen. Damit ein Wort (16 Bits) ins C-Register laden. legen. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
fetchy		Adresse vom Stack holen. Damit ein Wort (16 Bits) ins Y-Register laden.Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
fetchz		Adresse vom Stack holen. Damit ein Wort (16 Bits) ins Z-Register laden. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
stora		Adresse vom Stack holen. Damit ein Byte aus dem A-Register speichern. legen. Das X-Register enthält die um 1 erhöhte Adresse (Folgeadresse)
storad		Adresse vom Stack holen. Damit ein Wort (16 Bits) aus dem A-Register speichern. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
storb		Adresse vom Stack holen. Damit ein Byte aus dem B-Register speichern. Das X-Register enthält die um 1 erhöhte Adresse (Folgeadresse)
storbd		Adresse vom Stack holen. Damit ein Wort (16 Bits) aus dem B-Register speichern. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
storc		Adresse vom Stack holen. Damit ein Byte aus dem C-Register speichern. Das X-Register enthält die um 1 erhöhte Adresse (Folgeadresse)
storcd		Adresse vom Stack holen. Damit ein Wort (16 Bits) aus dem C-Register speichern. Das X-Register enthält die um 2 erhöhte Adresse (Folgeadresse)
23. Makros zum Unterprogrammruf und zur Parameteradressierung		
enter	No_of_bytes (needed for local variables)	Funktionseintritt gemäß Programmiersprache C. Das Y-Register dient als Frame Pointer (FP). FP kommt auf den Stack. SP wird neuer FP. SP wird um die Anzahl der benötigten Bytes vermindert
leave		Verlassen einer Funktion gemäß Programmiersprache C. Das Y-Register dient als Frame Pointer (FP). FP wird neuer SP. Alter FP wird aus Stack wiederhergestellt. Abschließend Unterprogrammrückkehr.

Mnemonic	Parameter	Wirkung
stackindexx		Im X-Register wird ein Adreßzeiger bereitgestellt, der auf das erste Byte unterhalb des geretteten Befehlszählers zeigt . Damit kann auf Parameter im Stack zugegriffen werden.
stackindexy		Im Y-Register wird ein Adreßzeiger bereitgestellt, der auf das erste Byte unterhalb des geretteten Befehlszählers zeigt . Damit kann auf Parameter im Stack zugegriffen werden.
clrstackretx	No_of_bytes (to be cleared from stack)	Unterprogrammrückkehr, wobei die Anzahl an Bytes unterhalb des Befehlszählers vom Stack entfernt wird. SP in X, Rückkehradresse in Z
clrstackrety	No_of_bytes (to be cleared from stack)	Unterprogrammrückkehr, wobei die Anzahl an Bytes unterhalb des Befehlszählers vom Stack entfernt wird. SP in Y, Rückkehradresse in Z