Einige wichtige Assemblerbefehle

Wolfgang Kippels

19. September 2014

Hier habe ich die (meiner Meinung nach) wichtigsten Assemblerbefehle für die X86-Prozessoren in alphabetischer Reihenfolge zusammengetragen. Sie gehört zu einem Assembler-Lehrgang, der hier zu finden ist:
http://dk4ek.de/lib/exe/fetch.php/as_lehrg.pdf

Die Befehle für die FPU sind **nicht** mit dabei.

Groß und fett gedruckt steht immer der Befehl. Dahinter stehen ***kursiv*** zwischen 0 und 3 Parametern, die durch Kommata voneinander getrennt sind. Hier müssen die entsprechenden Register oder Variablen eingetragen werden. Darunter steht die Beschreibung, was der Befehl macht. In der jeweils letzten Zeile ist angegeben, welche Flags der Befehl beeinflusst.

AAA

Korrektur nach ASCII- (bzw. BCD-ungepackt-) Addition.
Wenn das niederwertige Halbbyte von AL eine Pseudotetrade enthält oder das AF gesetzt ist, wird 6 zu AL addiert und 1 zu AH. Die höherwertige Tetrade von AL wird auf Null gesetzt.

Flags: beeinflusst: A, C undefiniert: O, S, Z, P

AAD

Korrektur vor einer ASCII- (bzw. BCD-ungepackt-) Division.
AH wird mit 10 multipliziert und zu AL addiert. AH wird dann auf Null gesetzt. AX enthält dann das binäre Äquivalent der ursprünglichen BCD-Zahl.

Flags: beeinflusst: S, Z, P undefiniert: O, A, C

AAM

Korrektur nach ASCII- (bzw. BCD-ungepackt-) Multiplikation.
Das Produkt in AL wird durch 10 geteilt. Das ganzzahlige Ergebnis erscheint in AH, der Rest in AL.

Flags: beeinflusst: S, Z, P undefiniert: O, A, C

AAS

Korrektur nach ASCII- (bzw. BCD-ungepackt-) Subtraktion.
Wenn das niederwertige Halbbyte von AL nach einer Subtraktion eine Pseudotetrade enthält oder AF gesetzt ist, wird AH um 1 heruntergezählt und von AL 6 subtrahiert. Die höherwertige Tetrade von AL wird auf Null gesetzt.

Flags: beeinflusst: A, C undefiniert: O, S, Z, P

ADC *Ziel,Quelle*

Addieren mit Übertrag.
Ziel := *Ziel* + *Quelle* + CF

Flags: beeinflusst: O, S, Z, A, P, C

ADD *Ziel,Quelle*

Addieren ohne Übertrag.
Ziel := *Ziel* + *Quelle*

Flags: beeinflusst: O, S, Z, A, P, C

AND *Ziel,Quelle*

Logisches UND.
Ziel bitweise UND-verknüpft mit *Quelle* , Ergebnis in *Ziel*

Flags: beeinflusst: S, Z, P undefiniert: A O=0, C=0

CALL *Adresse*

Aufruf eines Unterprogramms als **NEAR**-Prozedur. Das Unterprogramm muss mit **RET** enden.

Flags: - - -

CALLF *Adresse*

Aufruf eines Unterprogramms als **FAR**-Prozedur. Das Unterprogramm muss mit **RETF** enden.

Flags: - - -

CBW

Umwandlung von Byte in Word.
Wandelt das Byte in AL vorzeichenrichtig in ein Wort in AX.

Flags: - - -

CLC

Carry-Flag CF löschen.

Flags: C=0

CLD

Direction-Flag DF löschen. Bei String-Operationen bewegen sich die Zeiger nun vorwärts.

Flags: D=0

CLI

Interrupt-Flag IF löschen. Danach werden keine maskierbaren Interrupts mehr angenommen.

Flags: I=0

CMC

Carry-Flag CF komplementieren. Das CF wird auf 0 gesetzt, wenn es 1 war und umgekehrt.

Flags: beeinflusst: C

CMP *Operand1,Operand2*

Vergleiche *Operand2* mit *Operand1.*

Flags: beeinflusst: O, S, Z, A, P, C

CMPSB

Vergleiche Byte von Strings.

Wirkung wie **CMP**, wobei *Operand1* adressiert wird durch DS:SI und *Operand2* durch ES:DI. Nach dem Vergleich werden SI und DI in Abhängigkeit von DF um 1 erhöht (DF=0) oder um 1 erniedrigt (DF=1).

Flags: beeinflusst: O, S, Z, A, P, C

CMPSW

Vergleiche Word von Strings

Wirkung wie **CMPSB**, jedoch **WORD**-Vergleich und anschließende Erhöhung bzw. Erniedrigung von DI und SI um 2.

Flags: beeinflusst: O, S, Z, A, P, C

CWD

Wort in Doppelwort umwandeln.

Wandelt ein Wort in AX vorzeichengerecht in ein Doppelwort in DX:AX um.

Flags: beeinflusst: O, S, Z, A, P, C

DAA

Dezimalkorrektur nach einer Addition (BCD-gepackt) in AL.

Wenn das niederwertige Halbbyte von AL größer als 9 ist oder das AF gesetzt ist, wird 6 zu AL addiert. Ist dann die höherwertige Tetrade größer als 9 oder ist CF gesetzt, wird 60h zu AL addiert und CF gesetzt.

Flags: beeinflusst: S, Z, A, P, C undefiniert: O

DAS

Dezimalkorrektur nach einer Subtraktion (BCD-gepackt) in AL.

Wenn das niederwertige Halbbyte von AL größer als 9 ist oder das AF gesetzt ist, wird 6 von AL subtrahiert. Ist dann die höherwertige Tetrade größer als 9 oder ist CF gesetzt, wird 60hex von AL subtrahiert und CF gesetzt.

Flags: beeinflusst: S, Z, A, P, C undefiniert: O

DEC *Operand*

Von *Operand* 1 subtrahieren.

Flags: beeinflusst: S, Z, A, P undefiniert: O

DIV *Operand*

Dividiere vorzeichenlos.

- **Wenn der Operand ein Byte ist**, wird der Inhalt von AX durch das Byte dividiert. Das ganzzahlige Ergebnis der Division steht ohne Vorzeichen in AL, der Divisions-Rest in AH.
- **Ist der Operand ein Wort**, wird der Inhalt von DX:AX durch das Wort dividiert. Das ganzzahlige Ergebnis der Division steht ohne Vorzeichen in AX, der Divisions-Rest in DX.

Paßt das Ergebnis nicht in das vorgesehene Register, wird ein Interrupt 0 (Überlauf nach Division) ausgelöst.

Flags: undefiniert: O, S, Z, A, P, C

HLT

Halt.

Der Prozessor geht in den Halte-Zustand. Dieser Zustand wird erst durch einen Interrupt oder Reset (Anlegen eines entsprechenden Signals an den Prozessor) verlassen.

Flags: - - -

IDIV *Operand*

Ganzzahl-Division mit Vorzeichen.

- **Wenn der Operand ein Byte ist**, wird der Inhalt von AX durch das Byte dividiert. Das ganzzahlige Ergebnis der Division steht vorzeichenrichtig in AL, der Divisions-Rest in AH.
- **Ist der Operand ein Wort**, wird der Inhalt von DX:AX durch das Wort dividiert. Das ganzzahlige Ergebnis der Division steht vorzeichenrichtig in AX, der Divisions-Rest in DX.

Paßt das Ergebnis nicht in das vorgesehene Register, wird ein Interrupt 0 (Überlauf nach Division) ausgelöst.

Flags: undefiniert: O, S, Z, A, P, C

IMUL *Quelle* oder:
IMUL *Ziel,Quelle* oder:
IMUL *Ziel,Faktor1,Faktor2*

- **Ist nur ein Operand vorhanden:**
  - **Ist der Operand ein Byte**, so wird er mit dem Inhalt von AL vorzeichenrichtig multipliziert. Das Ergebnis der Multiplikation steht in AX.
  - **Ist der Operand ein Wort**, so wird er mit dem Inhalt von AX vorzeichenrichtig multipliziert. Das Ergebnis der Multiplikation steht in DX:AX (32Bit).
- **Sind zwei Operanden vorhanden:** Der erste Operand wird mit dem zweiten vorzeichenrichtig multipliziert, das Ergebnis steht im ersten Operanden.
- **Sind drei Operanden vorhanden:** Faktor1 wird mit Faktor2 vorzeichenrichtig multipliziert, das Ergebnis steht in *Ziel*.

Ist das Ergebnis zu groß, um in das angegebene Register zu passen, wird das Overflow- und das Carry-Flag gesetzt.

Flags: beeinflusst: O, C undefiniert: S, Z, A, P

IN *Ziel,Kanal-Nr.*

Input.
Liest ein Byte, Wort oder Doppelwort von einem Eingabe-Kanal (Port-Baustein) ein und überträgt es in das *Ziel* AL, AX oder EAX. Die Kanal-Nummer ist entweder eine 1-Byte-Konstante (0 ... 255), oder der Inhalt von DX, womit 64K verschiedene Kanäle adressierbar sind.

Flags: - - -

INC *Operand*

Eins hochzählen.
Der *Operand* wird um 1 erhöht.

Flags: beeinflusst: O, S, Z, A, P

INSB

Byte-String einlesen.
Liest ein Byte über den durch DX adressierten Kanal ein und überträgt es in das durch ES:DI adressierte Byte. Wenn DF=0 ist, wird anschließend DI um 1 erhöht, ansonsten um 1 erniedrigt.

Flags: - - -

INSW

Wort-String einlesen.
Liest ein Wort über den durch DX adressierten Kanal ein und überträgt es in das durch ES:DI adressierte Wort. Wenn DF=0 ist, wird anschließend DI um 2 erhöht, ansonsten um 2 erniedrigt.

Flags: - - -

INT *Nummer*

Interrupt aufrufen.
Der Interrupt mit der angegebenen *Nummer* wird aufgerufen.

Flags: - - -

INTO

Unterbrechung bei Overflow.
Wenn das OF gesetzt ist, wird ein Interrupt 04 ausgelöst, anderenfalls normal weitergearbeitet.

Flags: I=0, T=0

IRET

Rückkehr von einem Interrupt.
Steht am Ende einer Interrupt-Routine. Auch die gesicherten Flags werden zurückgeholt.

Flags: beeinflusst: Alle

JA *Ziel*

Sprung zu *Ziel*, wenn vorzeichenlos größer. Identisch mit **JNBE**.

Flags: - - -

JAE *Ziel*

Sprung zu *Ziel*, wenn vorzeichenlos größer oder gleich. Identisch mit **JNB**, **JNC**.

Flags: - - -

JB *Ziel*

Sprung zu *Ziel*, wenn vorzeichenlos kleiner. Identisch mit **JNAE**, **JC**.

Flags: - - -

JBE *Ziel*

Sprung zu *Ziel*, wenn vorzeichenlos kleiner oder gleich. Identisch mit **JNA**.

Flags: - - -

JC *Ziel*

Sprung zu *Ziel*, wenn CF gesetzt. Identisch mit **JNAE**, **JB**.

Flags: - - -

JCXZ *Ziel*
Sprung zu *Ziel*, wenn der Inhalt von CX = 0 ist.
Flags: - - -

JE *Ziel*
Sprung zu *Ziel*, wenn gleich. Identisch mit **JZ**.
Flags: - - -

JECXZ *Ziel*
Sprung zu *Ziel*, wenn der Inhalt von ECX = 0 ist.
Flags: - - -

JG *Ziel*
Sprung zu *Ziel*, wenn vorzeichenrichtig größer. Identisch mit **JNLE**.
Flags: - - -

JGE *Ziel*
Sprung zu *Ziel*, wenn vorzeichenrichtig größer oder gleich. Identisch mit **JNL**.
Flags: - - -

JL *Ziel*
Sprung zu *Ziel*, wenn vorzeichenrichtig kleiner. Identisch mit **JNGE**.
Flags: - - -

JLE *Ziel*
Sprung zu *Ziel*, wenn vorzeichenrichtig kleiner oder gleich. Identisch mit **JNG**.
Flags: - - -

JMP *Ziel*
Unbedingter Sprung.
Wenn das *Ziel* ein Doppelwort ist, erfolgt ein **FAR JUMP**, sonst ein **SHORT JUMP** zu *Ziel*.
Flags: - - -

JNA *Ziel*
Sprung zu *Ziel*, wenn vorzeichenlos nicht größer. Identisch mit **JBE**.
Flags: - - -

JNAE *Ziel*
Sprung zu *Ziel*, wenn vorzeichenlos nicht größer oder gleich. Identisch mit **JB** und **JC**.
Flags: - - -

JNB *Ziel*
Sprung zu *Ziel*, wenn vorzeichenlos nicht kleiner. Identisch mit **JAE**.
Flags: - - -

JNBE *Ziel*
Sprung zu *Ziel*, wenn vorzeichenlos nicht kleiner oder gleich. Identisch mit **JA**.
Flags: - - -

JNC *Ziel*
Sprung zu *Ziel*, wenn kein Carry-Flag gesetzt. Identisch mit **JAE**.
Flags: - - -

JNE *Ziel*
Sprung zu *Ziel*, wenn nicht gleich. Identisch mit **JNZ**.
Flags: - - -

JNG *Ziel*
Sprung zu *Ziel*, wenn vorzeichenrichtig nicht größer. Identisch mit **JLE**.
Flags: - - -

JNGE *Ziel*
Sprung zu *Ziel*, wenn vorzeichenrichtig nicht größer oder gleich. Identisch mit **JL**.
Flags: - - -

JNL *Ziel*
Sprung zu *Ziel*, wenn vorzeichenrichtig nicht kleiner. Identisch mit **JGE**.
Flags: - - -

JNLE *Ziel*
Sprung zu *Ziel*, wenn vorzeichenrichtig nicht kleiner oder gleich. Identisch mit **JG**.
Flags: - - -

JNO *Ziel*
Sprung zu *Ziel*, wenn Überlauf-Flag OF nicht gesetzt.
Flags: - - -

JNP *Ziel*
Sprung zu *Ziel*, wenn Parity-Flag PF nicht gesetzt. Identisch mit **JPO**.
Flags: - - -

JNS *Ziel*
Sprung zu *Ziel*, wenn Vorzeichen-Flag SF nicht gesetzt (Vorzeichen positiv).
Flags: - - -

JNZ *Ziel*

Sprung zu *Ziel*, wenn nicht Null (ZF nicht gesetzt). Identisch mit **JNE**.

Flags: - - -

JO *Ziel*

Sprung zu *Ziel*, wenn Überlauf-Flag OF gesetzt.

Flags: - - -

JP *Ziel*

Sprung zu *Ziel*, wenn Parity-Flag PF gesetzt. Identisch mit **JPE**.

Flags: - - -

JPE *Ziel*

Sprung zu *Ziel*, wenn Parity gerade. Identisch mit **JP**.

Flags: - - -

JPO *Ziel*

Sprung zu *Ziel*, wenn Parity ungerade. Identisch mit **JNP**.

Flags: - - -

JS *Ziel*

Sprung zu *Ziel*, wenn Vorzeichen-Flag SF gesetzt (Vorzeichen negativ).

Flags: - - -

JZ *Ziel*

Sprung zu *Ziel*, wenn Null (ZF gesetzt). Identisch mit **JE**.

Flags: - - -

LAHF

Lade AH mit dem niederwertigen Byte des Flagwortes.
Zuordnung der AH-Bits: 7=SF 6=ZF 4=AF 2=PF 0=CF

Flags: - - -

LDS *Ziel,Quelle*

Lade Zeiger nach DS und Register.
Lädt einen Zeiger (Doppelwort) aus dem Speicher (*Quelle*) in DS und das angegebene Wortregister (*Ziel*). Das mit *Quelle* adressierte zuerst gelesene Speicherwort geht zu dem angegebenen Wortregister, das nächste nach DS.

Flags: - - -

LEA *Ziel,Quelle*

Lade effektive Adresse.

Die effektive Adresse des Speicheroperanden *Quelle* wird in das angegebene Wortregister *Ziel* geladen. Dieser Befehl sollte nur verwendet werden, wenn die effektive Adresse erst zur Laufzeit des Programms feststeht.

Flags: - - -

LES *Ziel,Quelle*

Lade Zeiger nach ES und Register.

Lädt einen Zeiger (Doppelwort) aus dem Speicher *Quelle* in ES und das angegebene Wortregister *Ziel*. Das mit *Quelle* adressierte zuerst gelesene Speicherwort geht zu dem angegebenen Wortregister, das nächste nach ES.

Flags: - - -

LODSB

Lade Stringelement.

Lädt ein Byte vom Quellstring (adressiert durch DS:SI) nach AL. Danach wird SI in Abhängigkeit von DF um 1 erhöht (DF=0) oder um 1 erniedrigt (DF=1).

Flags: - - -

LODSW

Lade Stringelement.

Lädt ein Wort vom Quellstring (adressiert durch DS:SI) nach AX. Danach wird SI in Abhängigkeit von DF um 2 erhöht (DF=0) oder um 2 erniedrigt (DF=1).

Flags: - - -

LOOP *Ziel*

Schleifendurchlauf, bis CX=0. (In CX steht die Anzahl der Schleifendurchläufe.)

CX wird um 1 vermindert. Wenn CX dann nicht =0 ist, wird zum angegebenen *Ziel* gesprungen.

Flags: - - -

LOOPE *Ziel* oder: **LOOPZ *Ziel***

Schleifendurchlauf, solange gleich.

Wie **LOOP**, jedoch Abbruch des Schleifendurchlaufes, wenn ZF=0 ist.

Flags: - - -

LOOPNE *Ziel* oder: **LOOPNZ *Ziel***

Schleifendurchlauf, solange ungleich.

Wie **LOOP**, jedoch Abbruch des Schleifendurchlaufes, wenn ZF=1 ist.

Flags: - - -

MOV ***Ziel,Quelle***

Kopiere von der *Quelle* zum *Ziel*.

Flags: - - -

MOVSB

Kopiere Byte von String.
Kopiert ein Byte vom Quellstring adressiert durch DS:SI an die durch ES:DI adressierte Stelle des Speichers. Danach werden SI und DI in Abhängigkeit von DF um 1 erhöht (DF=0) oder um 1 erniedrigt (DF=1).

Flags: - - -

MOVSW

Kopiere Wort von String.
Kopiert ein Wort vom Quellstring adressiert durch DS:SI an die durch ES:DI adressierte Stelle des Speichers. Danach werden SI und DI in Abhängigkeit von DF um 2 erhöht (DF=0) oder um 2 erniedrigt (DF=1).

Flags: - - -

MUL ***Operand***

Multiplikation ohne Vorzeichen.

- **Ist der Operand ein Byte**, so wird er mit dem Inhalt von AL multipliziert. Das Ergebnis der Multiplikation steht in AX.
- **Ist der Operand ein Wort**, so wird er mit dem Inhalt von AX multipliziert. Das Ergebnis der Multiplikation steht dann in DX:AX.

Falls das höherwertige Byte (bzw. Wort) des Ergebnisses =0 ist, werden CF und OF auf 0 gesetzt, anderenfalls auf 1.

Flags: beeinflusst: O, C undefiniert: S, Z, A, P

NEG ***Operand***

Negieren (Zweier-Komplement bilden).
Bildet das Zweier-Komplement des Operanden (Subtraktion von 0).

Flags: beeinflusst: O, S, Z, A, P C=1 Wenn der Operand 0 ist, dann wird C=0.

NOP

Keine Operation (nur ein „Füllbefehl“, um unbenutzte Bytes zu füllen).

Flags: - - -

NOT ***Operand***

Bitweises Negieren des Operanden.

Flags: - - -

OR *Ziel,Quelle*

Oder.

Verknüpft *Ziel* und *Quelle* bitweise durch **ODER**. Das Ergebnis steht in *Ziel*.

Flags: beeinflusst: S, Z, P undefiniert: A O=0 C=0

OUT *Kanal-Nummer,Quelle*

Ausgabe.

Überträgt ein Byte, Wort oder Doppelwort von *Quelle* (AL, AX oder EAX) zu einem Ausgabe-Kanal (Port-Baustein). Die *Kanal-Nummer* ist entweder eine 1-Byte-Konstante (0 ... 255), oder der Inhalt von DX, womit 64K verschiedene Kanäle adressierbar sind.

Flags: - - -

POP *Ziel*

Vom Stapel nehmen.

Das oberste Wort wird vom Stapel genommen und nach *Ziel* geladen. Danach erhöht sich SP um 2 und zeigt auf das nächste vom Stapel zu lesende Wort.

Flags: - - -

POPA

Alle wichtigen Register vom Stapel nehmen.

Speichert die Registerinhalte, die durch **PUSHA** auf den Stapel gebracht wurden, zurück in die Register. Dies sind in der Reihenfolge: DI, SI, BP, SP, BX, DX, CX, AX.

Flags: - - -

POPF

Flags vom Stapel nehmen.

Das oberste Wort wird vom Stapel ins Flag-Register übertragen. Danach erhöht sich SP um 2 und zeigt auf das nächste vom Stapel zu lesende Wort.

Flags: beeinflusst: Alle

PUSH *Quelle*

Auf den Stapel legen.

Der Stackpointer wird um 2 verringert, dann wird das Wort von *Quelle* auf den Stapel übertragen. SP zeigt dann auf das zuletzt auf den Stapel gebrachte Wort.

Flags: - - -

PUSHA

Alle wichtigen Register auf den Stapel legen.
Speichert die Registerinhalte in der Reihenfolge AX, CX, DX, BX, SP, BP, SI, DI auf den Stapel. Der von SP gespeicherte Wert ist der vor der Ausführung des Befehles vorhandene Wert.

Flags: - - -

PUSHF

Flags auf den Stapel legen.
Der Stackpointer wird um 2 verringert, dann wird das Flag-Register auf den Stapel übertragen. SP zeigt dann auf das zuletzt auf den Stapel gebrachte Wort.

Flags: - - -

RCL *Operand,Anzahl*

Linksrotieren durchs Carry-Flag.
Rotiert den Operanden *Anzahl* mal Bit-weise nach links durch das Carry-Flag. Der Inhalt des Carry-Flag geht jeweils nach Bit 0. Das höchstwertige Bit geht jeweils nach CF. Beim 1-Bit-Shift gilt: Ist das höchstwertige Bit gleich CF, wird OF auf 0 gesetzt, sonst auf 1.

Flags: beeinflusst: O, C

RCR *Operand,Anzahl*

Rechtsrotieren durchs Carry-Flag.
Rotiert den Operanden *Anzahl* mal Bit-weise nach rechts durch das Carry-Flag. Der Inhalt des Carry-Flag geht jeweils ins höchstwertigste Bit. Der Inhalt von Bit 0 geht jeweils nach CF. Beim 1-Bit-Shift gilt: Stimmen nach der Operation die beiden höchstwertigen Bits überein, wird OF auf 0 gesetzt, sonst auf 1.

Flags: beeinflusst: O, C

REP

Wiederhole CX mal.
REP ist ein **Präfix** zu einem Stringbefehl. Der nachfolgende Befehl wird so oft wiederholt, bis CX auf Null heruntergezählt ist. Die Flags werden dadurch nicht beeinflusst.

Flags: - - -

REPE oder: REPZ

Wiederhole CX mal, solange gleich.
Wie **REP**. Liegt einer der Befehle **SCASx** oder **CMPSx** vor, wird die Wiederholung abgebrochen, sobald ZF=0 ist.

Flags: - - -

REPNE oder: **REPNZ**

Wiederhole CX mal, solange nicht gleich.
Wie **REP**. Liegt einer der Befehle **SCASx** oder **CMPSx** vor, wird die Wiederholung abgebrochen, sobald ZF=1 ist.

Flags: - - -

RET oder: **RET** ***Direktwert***

Rücksprung aus einem **NEAR**-Unterprogramm.
Die Rücksprungadresse wird als Wort vom Stapel genommen. Falls ein *Direktwert* (eine Zahl) vorkommt, wird dieser anschließend zum Stapelzeiger addiert. Dadurch können Parameter, die vor Aufruf der Prozedur auf den Stapel gebracht wurden (zwecks Übergabe an die Prozedur) wieder freigegeben werden. (Bei 32-Bit-Software wird der doppelte Wert zu SP addiert.)

Flags: - - -

RETF oder: **RETF** ***Direktwert***

Rücksprung aus einem **FAR**-Unterprogramm.
Die Rücksprungadresse wird als Doppel-Wort vom Stapel genommen. Falls ein *Direktwert* (eine Zahl) vorkommt, wird dieser anschließend zum Stapelzeiger addiert. Dadurch können Parameter, die vor Aufruf der Prozedur auf den Stapel gebracht wurden (zwecks Übergabe an die Prozedur) wieder freigegeben werden. (Bei 32-Bit-Software wird der doppelte Wert zu SP addiert.)

Flags: - - -

ROL ***Operand,Anzahl***

Links rotieren.
Rotiert den Operanden *Anzahl* mal Bit-weise nach links. Der Inhalt des höchstwertigen Bits geht jeweils nach Bit 0 und wird gleichzeitig nach CF kopiert. Beim 1-Bit-Shift gilt: Ist das höchstwertige Bit gleich CF, wird OF auf 0 gesetzt, sonst auf 1.

Flags: beeinflusst: O, C

ROR ***Operand,Anzahl***

Rechts rotieren.
Rotiert den Operanden *Anzahl* mal Bit-weise nach rechts. Der Inhalt des Bits 0 geht jeweils in das höchstwertige Bit und wird gleichzeitig nach CF kopiert. Beim 1-Bit-Shift gilt: Stimmen nach der Operation die beiden höchstwertigen Bits überein, wird OF auf 0 gesetzt, sonst auf 1.

Flags: beeinflusst: O, C

SAHF

Übertrage AH ins Flag-Register.
Überträgt AH in das niederwertige Byte des Flagwortes.
Zuordnung der AH-Bits: 7=SF 6=ZF 4=AF 2=PF 0=CF

Flags: beeinflusst: S, Z, A, P, C

SAL *Operand,Anzahl* oder: **SHL *Operand,Anzahl***

Schiebe arithmetisch nach links.
Schiebt den Operanden *Anzahl* mal Bit-weise nach links. Der Inhalt des höchstwertigen Bits geht jeweils nach CF. Von Bit 0 aus wird jeweils eine 0 nachgezogen. Beim 1-Bit-Shift gilt: Ist das höchstwertige Bit gleich CF, wird OF auf 0 gesetzt, sonst auf 1.

Flags: beeinflusst: O, C

SAR *Operand,Anzahl* oder: **SHR *Operand,Anzahl***

Schiebe arithmetisch nach rechts.
Schiebt den Operanden *Anzahl* mal Bit-weise nach rechts. Der Inhalt von Bit 0 geht jeweils nach CF. Das höchstwertigste Bit bleibt erhalten!
Beim 1-Bit-Shift gilt: OF wird auf 0 gesetzt.

Flags: beeinflusst: O, S, Z, P, C undefiniert: A

SBB *Ziel,Quelle*

Subtrahieren mit Übertrag.
Subtrahiert *Quelle* und Carry-Flag von *Ziel*. Das Ergebnis steht in *Ziel*.

Flags: beeinflusst: O, S, Z, A, P, C

SCASB und **SCASW**

String durchsuchen.
Vergleicht (subtrahiert) ein Byte (bei **SCASB**) oder ein Wort (bei **SCASW**) der Zeichenkette adressiert durch ES:DI mit AL bzw. AX. Die Flags werden entsprechend gesetzt.

Flags: beeinflusst: O, S, Z, A, P, C

SHL *Operand,Anzahl* oder: **SAL *Operand,Anzahl***

Schiebe logisch nach links.
Schiebt den Operanden *Anzahl* mal Bit-weise nach links. Der Inhalt von Bit 7 geht jeweils nach CF. Zum niederwertigsten Bit wird eine 0 nachgezogen.
Beim 1-Bit-Shift gilt: OF wird auf 0 gesetzt.

Flags: beeinflusst: O, C

SHR ***Operand,Anzahl*** oder: **SAR** ***Operand,Anzahl***

Schiebe logisch nach rechts.

Schiebt den Operanden *Anzahl* mal Bit-weise nach rechts. Der Inhalt von Bit 0 geht jeweils nach CF. Zum höchstwertigen Bit wird eine 0 nachgezogen.

Beim 1-Bit-Shift gilt: OF wird auf 0 gesetzt.

Flags: beeinflusst: O, S, Z, P, C undefiniert: A

STC

Carry-Flag setzen.

Flags: C=1

STD

Richtungs-Flag setzen. Bei Stringoperationen werden Zeiger nun rückwärts bewegt.

Flags: D=1

STI

Interrupt-Flag setzen. Maskierbare Interrupts sind nun erlaubt.

Flags: I=1

STOSB und **STOSW**

Byte bzw. Wort in String speichern.

Überträgt den Inhalt von AL (**STOSB**) bzw. AX (**STOSW**) in eine Zeichenkette adressiert durch ES:DI. Anschließend wird DI in Abhängigkeit von DF um 1 (**STOSB**) bzw. um 2 (**STOSW**) erhöht (DF=0) oder entsprechend erniedrigt (DF=1).

Flags: - - -

SUB ***Ziel,Quelle***

Subtrahiere

Subtrahiert *Quelle* von *Ziel*. Das Ergebnis steht in *Ziel*.

Flags: - - -

TEST ***Operand1,Operand2***

Testen

Die beiden Operanden werden **UND**-verknüpft und die Flags entsprechend gesetzt. Das Ergebnis der Verknüpfung wird nicht abgespeichert.

Flags: beeinflusst: S, Z, P undefiniert: A O=0 C=0

WAIT

Warten

Der Prozessor bleibt im Wartezustand, bis er durch ein Signal an seinem **TEST**-Pin wieder aktiviert wird. Nur Interrupts werden zwischenzeitlich ausgeführt.

Flags: - - -

XCHG *Operand1,Operand2*

Austauschen

Die Inhalte von *Operand1* und *Operand2* werden getauscht.

Flags: - - -

XLAT oder: XLATB

Übersetze

Ein Element wird aus einer maximal 256 Byte großen Tabelle gelesen. Die Anfangsadresse (Offset) der Tabelle muß in BX stehen. Der Inhalt von AL gibt – als Index genommen – die Lage des gesuchten Tabellenelementes relativ zum Tabellenanfang an. Das gelesene Tabellenelement wird nach AL gebracht. AL:=[BX+AL]

Flags: beeinflusst: O, S, Z, P, C undefiniert: A

XOR *Ziel,Quelle*

Exclusiv-Oder

Der Operand *Ziel* und *Quelle* werden bitweise **EXCLUSIV-ODER** verknüpft. Das Ergebnis steht in *Ziel*.

Tip: Soll ein Register auf 0 gesetzt werden, empfiehlt sich eine **XOR**-Verknüpfung des Registers mit sich selbst. Das geht schneller, als die 0 zu laden.

Beispiel: **XOR AX,AX**

Flags: beeinflusst: S, Z, P undefiniert: A O=0 C=0