

16. Assemblerbefehle-Referenz

16.13. Gleitkomma-Befehle

16.13.1. Datentransportbefehle

FLD—Load Floating Point Value

Instruction	Description
FLD <i>m32fp</i>	Push <i>m32fp</i> onto the FPU register stack.
FLD <i>m64fp</i>	Push <i>m64fp</i> onto the FPU register stack.
FLD <i>m80fp</i>	Push <i>m80fp</i> onto the FPU register stack.
FLD ST(<i>i</i>)	Push ST(<i>i</i>) onto the FPU register stack.

Beschreibung

Schiebt den Quelloperanden vom Speicher auf den FPU-Register-Stack. Der Quelloperand kann eine 32bit-(float), 64bit-(double) oder 80bit-(long double) Gleitkommazahl sein. Die 32bit- und 64bit-Formate werden dabei in das interne 80bit-Gleitkommaformat umgewandelt. Der FLD Befehl kann auch den Wert aus einem FPU-Register ST(*i*) auf den Stack kopieren.

Beispiele

```
FLD dword ptr[f] ; float
FLD qword ptr[d] ; double
FLD tbyte ptr[ld] ; long double
FLD ST(0)        ; Register duplizieren
```

FILD—Load Integer

Instruction	Description
FILD <i>m16int</i>	Push <i>m16int</i> onto the FPU register stack.
FILD <i>m32int</i>	Push <i>m32int</i> onto the FPU register stack.
FILD <i>m64int</i>	Push <i>m64int</i> onto the FPU register stack.

Beschreibung

Konvertiert den vorzeichenbehafteten Integer Quelloperanden in das interne 80bit-Gleitkommaformat und legt den Wert auf den FPU-Register-Stack. Der Quelloperand kann ein WORD (short), DWORD (long) oder QWORD (_int64) Integer sein. Der Wert wird ohne Rundungsfehler geladen. Das Vorzeichen bleibt erhalten.

Beispiele

```
FILD word ptr[ebp-4] ; short int
FILD dword ptr[n]    ; long int
FILD qword ptr[timer] ; _int64
```

FLDc—Load Constant

Instruction	Description
FLD1	Push +1.0 onto the FPU register stack.
FLDZ	Push +0.0 onto the FPU register stack.
FLDPI	Push π onto the FPU register stack.
FLDL2T	Push $\log_2 10$ onto the FPU register stack.
FLDL2E	Push $\log_2 e$ onto the FPU register stack.
FLDLG2	Push $\log_{10} 2$ onto the FPU register stack.
FLDLN2	Push $\log_e 2$ onto the FPU register stack.

Beschreibung:

Legt eine von sieben häufig benötigten Konstanten (im 80bit-Gleitkommaformat) auf den FPU-Register-Stack. Die verfügbaren Konstanten sind 1.0, 0.0, $\log_2 10$, $\log_2 e$, π , $\log_{10} 2$, and $\log_e 2$. Jede dieser Konstanten ist im Prozessor als 66bit-Hex-Muster enthalten und wird bei der Umwandlung in das 80bit-Gleitkommaformat gerundet. Die Konstante π ist beispielsweise als $0.f.2^2$ einprogrammiert, wobei f das 66bit-Muster C90FDAA2 2168C234 C repräsentiert (die letzte Hex-Ziffer C steht nur für zwei Bits 11).

FST/FSTP—Store Floating Point Value

Instruction	Description
FST <i>m32fp</i>	Copy ST(0) to <i>m32fp</i>
FST <i>m64fp</i>	Copy ST(0) to <i>m64fp</i>
FST ST(<i>i</i>)	Copy ST(0) to ST(<i>i</i>)
FSTP <i>m32fp</i>	Copy ST(0) to <i>m32fp</i> and pop register stack
FSTP <i>m64fp</i>	Copy ST(0) to <i>m64fp</i> and pop register stack
FSTP <i>m80fp</i>	Copy ST(0) to <i>m80fp</i> and pop register stack
FSTP ST(<i>i</i>)	Copy ST(0) to ST(<i>i</i>) and pop register stack

Beschreibung:

Der FST Befehl kopiert den Wert vom obersten FPU-Register ST(0) in den Zieloperanden, der entweder eine Speicheradresse oder ein anderes FPU Register sein kann. Beim Kopieren an eine Speicheradresse wird der Wert in das einfache (DWORD) oder doppelt genaue (QWORD) Gleitkommaformat umgewandelt.

Der FSTP Befehl führt die gleiche Operation wie der FST Befehl durch. Danach wird der Wert vom FPU-Register-Stack abgeräumt. Bei dieser POP-Operation wird das ST(0) Register als 'leer' markiert und dann der stack pointer (TOP) um 1 erhöht.

Der FSTP Befehl kann den Wert auch im 80bit Format an die Speicheradresse kopieren. Wenn der Zieloperand im Speicher einfache oder doppelte Genauigkeit hat, wird die Mantisse auf die Größe des Zieloperanden gerundet. Der Exponent wird entsprechend der Breite und dem BIAS-Offset des Zieloperanden konvertiert.

Wenn der Wert zu groß ist für den Zieloperanden, wird eine numeric overflow exception (#O) generiert.

Beispiele

```

FST  dword ptr[f] ; float
FST  qword ptr[d] ; double
FSTP tbyte ptr[ld] ; long double
FSTP ST(1)         ; ST(1) entfernen
  
```

FIST/FISTP—Store Integer

Instruction	Description
FIST <i>m16int</i>	Store ST(0) in <i>m16int</i>
FIST <i>m32int</i>	Store ST(0) in <i>m32int</i>
FISTP <i>m16int</i>	Store ST(0) in <i>m16int</i> and pop register stack
FISTP <i>m32int</i>	Store ST(0) in <i>m32int</i> and pop register stack
FISTP <i>m64int</i>	Store ST(0) in <i>m64int</i> and pop register stack

Beschreibung

Der FIST Befehl konvertiert den Wert im ST(0) Register in einen Integer-Wert (mit Vorzeichen) und speichert diesen im angegebenen Zieloperanden. Das Ziel kann ein WORD (short), oder DWORD (long) Integer sein. Der Zieloperand wird mit der Adresse des untersten Bytes angegeben.

Der FSTP Befehl führt die gleiche Operation wie der FST Befehl durch. Danach wird der Wert vom FPU-Register-Stack abgeräumt. Bei dieser POP-Operation wird das ST(0) Register als 'leer' markiert und dann der stack pointer (TOP) um 1 erhöht.

Der FISTP Befehl kann den Wert in ST(0) auch als QWORD(_int64) speichern. Wenn der Wert in ST(0) Nachkommastellen enthält, wird dieser gerundet gemäß dem Rundungsverfahren, das im RC Feld des FPU Stuerwortes festgelegt ist.

Falls der zu konvertierende Wert zu groß ist für das Zielformat, wird eine invalid-arithmetic-operand exception (#IA) generiert.

Beispiel

```
FLD    qword ptr[d]    ; double d;
FISTP  dword ptr[n]    ; int n = (int)d;
```

16.13.2. Arithmetische Befehle

FADD/FADDP/FIADD—Add

Instruction	Description
FADD <i>m32fp</i>	Add <i>m32fp</i> to ST(0) and store result in ST(0)
FADD <i>m64fp</i>	Add <i>m64fp</i> to ST(0) and store result in ST(0)
FADD ST(0), ST(<i>i</i>)	Add ST(0) to ST(<i>i</i>) and store result in ST(0)
FADD ST(<i>i</i>), ST(0)	Add ST(<i>i</i>) to ST(0) and store result in ST(<i>i</i>)
FADD <i>m32fp</i>	Add <i>m32fp</i> to ST(0) and store result in ST(0)
FADD <i>m64fp</i>	Add <i>m64fp</i> to ST(0) and store result in ST(0)
FADD ST(0), ST(<i>i</i>)	Add ST(0) to ST(<i>i</i>) and store result in ST(0)
FADD ST(<i>i</i>), ST(0)	Add ST(<i>i</i>) to ST(0) and store result in ST(<i>i</i>)
FADDP ST(<i>i</i>), ST(0)	Add ST(0) to ST(<i>i</i>), store result in ST(<i>i</i>), and pop the register stack
FADDP	Add ST(0) to ST(1), store result in ST(1), and pop the register stack
FIADD <i>m32int</i>	Add <i>m32int</i> to ST(0) and store result in ST(0)
FIADD <i>m16int</i>	Add <i>m16int</i> to ST(0) and store result in ST(0)

Beschreibung:

Addiert den Quelloperanden zum Zieloperanden und speichert die Summe im Zielregister. Der Zieloperand ist immer ein FPU Register, der Quelloperand kann ein FPU Register oder eine Speicheradresse sein. Die Speicheradresse kann eine Gleitkommazahl mit einfacher oder doppelter Genauigkeit oder ein Wort- oder Doppelwort-Integer sein.

Die Version ohne Operand addiert den Inhalt von ST(0) zum Register ST(1), mit anschließendem POP nach ST(0).

Die Versionen mit einem Operand addiert den Inhalt der Speichervariablen zum ST(0) Register.

Die Versionen mit zwei Operanden addiert den Inhalt des Registers ST(0) zum Register ST(*i*) oder umgekehrt.

Der Wert in ST(0) kann beispielsweise verdoppelt werden durch: FADD ST(0), ST(0);

Die FADDP Varianten führen nach der Addition eine POP Operation aus. Dabei wird das ST(0) Register als ungültig markiert und der Stack Pointer (TOP) um 1 inkrementiert.

Beispiele

```

FIADD dword ptr[n]   ; int
FADD  dword ptr[f]   ; float
FADD  qword ptr[d]   ; double
FLD   ST(0)
FADDP ST(1),ST(0)    ; ST(0) = 2* ST(0)
  
```

FSUB/FSUBP/FISUB—Subtract

Instruction	Description
FSUB <i>m32fp</i>	Subtract <i>m32fp</i> from ST(0) and store result in ST(0)
FSUB <i>m64fp</i>	Subtract <i>m64fp</i> from ST(0) and store result in ST(0)
FSUB ST(0), ST(<i>i</i>)	Subtract ST(<i>i</i>) from ST(0) and store result in ST(0)
FSUB ST(<i>i</i>), ST(0)	Subtract ST(0) from ST(<i>i</i>) and store result in ST(<i>i</i>)
FSUBP ST(<i>i</i>), ST(0)	Subtract ST(0) from ST(<i>i</i>), store result in ST(<i>i</i>), and pop register stack
FSUBP	Subtract ST(0) from ST(1), store result in ST(1), and pop register stack
FISUB <i>m32int</i>	Subtract <i>m32int</i> from ST(0) and store result in ST(0)
FISUB <i>m16int</i>	Subtract <i>m16int</i> from ST(0) and store result in ST(0)

Beschreibung:

Subtrahiert den Quelloperanden von dem Zieloperanden und speichert die Differenz im Zieloperanden. Der Zieloperand ist immer ein FPU-Register. Der Quelloperand kann ein Register oder eine Speichervariable sein. Operanden im Speicher können Gleitkommazahlen im 32bit- (float) oder 64bit- (double) Format oder auch 16bit- (short) oder 32bit- (long) Integer-Zahlen sein.

Die Version ohne Operandenangabe subtrahiert den Inhalt des Registers ST(0) von ST(1) und speichert das Ergebnis in ST(1), mit anschließendem POP nach ST(0). Mit einer Operandenangabe wird der Inhalt einer Speichervariablen (Gleitkomma- oder Integer-Zahl) vom Wert in ST(0) subtrahiert und wieder in ST(0) abgelegt. Die Zwei-Operanden Version subtrahiert den Inhalt von ST(*i*) vom Wert in ST(0) oder umgekehrt.

Die FSUBP Befehle räumen anschließend den FPU-Register-Stack wieder ab. Bei dieser POP-Operation wird das ST(0) Register als 'leer' markiert und dann der stack pointer (TOP) um 1 erhöht.

Die FISUB Befehle konvertieren den den Integer-Operanden vor der Subtraktion in die intern benötigte 80bit-Form.

Beispiel

```
FLD    qword ptr[a]
FSUB   qword ptr[b] ; ST(0) = [a] - [b]
```

FSUBR/FSUBRP/FISUBR—Reverse Subtract

Instruction	Description
FSUBR <i>m32fp</i>	Subtract ST(0) from <i>m32fp</i> and store result in ST(0)
FSUBR <i>m64fp</i>	Subtract ST(0) from <i>m64fp</i> and store result in ST(0)
FSUBR ST(0), ST(<i>i</i>)	Subtract ST(0) from ST(<i>i</i>) and store result in ST(0)
FSUBR ST(<i>i</i>), ST(0)	Subtract ST(<i>i</i>) from ST(0) and store result in ST(<i>i</i>)
FSUBRP ST(<i>i</i>), ST(0)	Subtract ST(<i>i</i>) from ST(0), store result in ST(<i>i</i>), and pop register stack
FSUBRP	Subtract ST(1) from ST(0), store result in ST(1), and pop register stack
FISUBR <i>m32int</i>	Subtract ST(0) from <i>m32int</i> and store result in ST(0)
FISUBR <i>m16int</i>	Subtract ST(0) from <i>m16int</i> and store result in ST(0)

Beschreibung:

Diese Befehle führen die gleichen Operationen aus wie FSUB, FSUBP, and FISUB, aber mit vertauschten Quelloperanden. Die umgekehrte Subtraktion liefert daher ein anderes Vorzeichen. Sie ermöglichen oft eine effizientere Codierung durch Operandentausch.

Beispiele

```
FLD    qword ptr[a]
FSUBR  qword ptr[b] ; ST(0) = [b] - [a]
FLD    qword ptr[c]
FSUBRP ; ST(1) = [c] - ([b] - [a]) ; nach POP in ST(0)
```

FMUL/FMULP/FIMUL—Multiply

Instruction	Description
FMUL <i>m32fp</i>	Multiply ST(0) by <i>m32fp</i> and store result in ST(0)
FMUL <i>m64fp</i>	Multiply ST(0) by <i>m64fp</i> and store result in ST(0)
FMUL ST(0), ST(i)	Multiply ST(0) by ST(i) and store result in ST(0)
FMUL ST(i), ST(0)	Multiply ST(i) by ST(0) and store result in ST(i)
FMULP ST(i), ST(0)	Multiply ST(i) by ST(0), store result in ST(i), and pop the register stack
FMULP	Multiply ST(1) by ST(0), store result in ST(1), and pop the register stack
FIMUL <i>m32int</i>	Multiply ST(0) by <i>m32int</i> and store result in ST(0)
FIMUL <i>m16int</i>	Multiply ST(0) by <i>m16int</i> and store result in ST(0)

Beschreibung

Multipliziert den Quelloperanden mit dem Zieloperanden und speichert das Produkt im Zieloperanden. Der Zieloperand ist immer ein FPU-Register. Der Quelloperand kann ein Register oder eine Speichervariable sein. Operanden im Speicher können Gleitkommazahlen im 32bit- (float) oder 64bit- (double) Format oder auch 16bit- (short) oder 32bit- (long) Integer-Zahlen sein.

Die Version ohne Operandenangabe multipliziert den Inhalt des Registers ST(1) mit ST(0) und speichert das Ergebnis in ST(1), mit anschließendem POP nach ST(0). Bei Angabe eines Operanden wird der Inhalt einer Speichervariablen (Gleitkomma- oder Integer-Zahl) vom Wert in ST(0) subtrahiert und wieder in ST(0) abgelegt. Die Zwei-Operanden Version multipliziert den Inhalt von ST(i) mit dem Wert in ST(0) oder umgekehrt.

Die FMULP Befehle räumen anschliessend den FPU-Register-Stack wieder ab. Bei dieser POP-Operation wird das ST(0) Register als 'leer' markiert und dann der stack pointer (TOP) um 1 erhöht.

Die FIMUL Befehle konvertieren den Integer-Wert vor der Subtraktion in die interne 80bit-Form.

Beispiele:

```
FLD    qword ptr[a]
FMUL   ST(0), ST(0) ; ST(0) = [a] * [a]
FMUL   qword ptr[b] ; ST(0) = [b] * [a]*[a]
```

FDIV/FDIVP/FIDIV—Divide

Instruction	Description
FDIV <i>m32fp</i>	Divide ST(0) by <i>m32fp</i> and store result in ST(0)
FDIV <i>m64fp</i>	Divide ST(0) by <i>m64fp</i> and store result in ST(0)
FDIV ST(0), ST(i)	Divide ST(0) by ST(i) and store result in ST(0)
FDIV ST(i), ST(0)	Divide ST(i) by ST(0) and store result in ST(i)
FDIVP ST(i), ST(0)	Divide ST(i) by ST(0), store result in ST(i), and pop the register stack
FDIVP	Divide ST(1) by ST(0), store result in ST(1), and pop the register stack
FIDIV <i>m32int</i>	Divide ST(0) by <i>m32int</i> and store result in ST(0)
FIDIV <i>m16int</i>	Divide ST(0) by <i>m16int</i> and store result in ST(0)

Beschreibung:

Dividiert den Zieloperanden durch den Quelloperanden und speichert den Quotienten im Zieloperanden. Der Zieloperand (Dividend) ist immer ein FPU-Register. Der Quelloperand (Divisor) kann ein Register oder eine Speichervariable sein. Operanden im Speicher können

Gleitkommazahlen im 32bit- (float) oder 64bit- (double) Format oder auch 16bit- (short) oder 32bit- (long) Integer-Zahlen sein.

Die Version ohne Operandenangabe dividiert den Inhalt des Registers ST(1) durch den Inhalt von ST(0) und speichert das Ergebnis in ST(1), mit anschließendem POP nach ST(0). Bei Angabe eines Operanden wird der Wert in ST(0) durch den Inhalt der Speichervariablen (Gleitkomma- oder Integer-Zahl) dividiert und wieder in ST(0) abgelegt. Die Zwei-Operanden Version dividiert den Inhalt von ST(0) durch den Wert in ST(i) oder umgekehrt.

Die FDIVP Befehle räumen anschließend den FPU-Register-Stack wieder ab. Bei dieser POP-Operation wird das ST(0) Register als 'leer' markiert und der stack pointer (TOP) um 1 erhöht.

Die FIDIV Befehle konvertieren den Integer-Wert vor der Subtraktion in die interne 80bit-Form.

Beispiele:

```
FLD    qword ptr[b]
FLD    qword ptr[a]    ; ST(0) = [a] ; ST(1) = [b]
FDIVP                      ; ST(0) = [b] / [a]
```

FDIVR/FDIVRP/FIDIVR—Reverse Divide

Instruction	Description
FDIVR <i>m32fp</i>	Divide <i>m32fp</i> by ST(0) and store result in ST(0)
FDIVR <i>m64fp</i>	Divide <i>m64fp</i> by ST(0) and store result in ST(0)
FDIVR ST(0), ST(i)	Divide ST(i) by ST(0) and store result in ST(0)
FDIVR ST(i), ST(0)	Divide ST(0) by ST(i) and store result in ST(i)
FDIVRP ST(i), ST(0)	Divide ST(0) by ST(i), store result in ST(i), and pop the register stack
FDIVRP	Divide ST(0) by ST(1), store result in ST(1), and pop the register stack
FIDIVR <i>m32int</i>	Divide <i>m32int</i> by ST(0) and store result in ST(0)
FIDIVR <i>m16int</i>	Divide <i>m16int</i> by ST(0) and store result in ST(0)

Beschreibung:

Diese Befehle führen die gleichen Operationen aus wie FDIV, FDIVP, and FIDIV, aber mit vertauschten Operanden. Die umgekehrte Division liefert daher den Kehrwert. Diese Befehle ermöglichen oft eine effizientere Codierung durch Operandentausch.

Beispiele:

```
FLD    qword ptr[a]
FDIVR qword ptr[b]    ; ST(0) = [b] / [a]
FLD    qword ptr[c]
FDIVRP                      ; ST(1) = [c] / ([b]/[a]) ; nach POP in ST(0)
```

FABS—Absolute Value

Instruction	Description
FABS	Replace ST(0) with its absolute value.

Beschreibung

Löscht das Vorzeichenbit von ST(0) um den Absolutwert (Betrag) zu ermitteln.

FCHS—Change Sign

Instruction	Description
FCHS	Complements sign of ST(0)

Beschreibung

Wechselt das Vorzeichenbit in ST(0). Diese Operation macht aus einem positiven Wert einen negativen Wert mit gleichem Betrag, und umgekehrt.

FSQRT—Square Root

Instruction	Description
FSQRT	Computes square root of ST(0) and stores the result in ST(0)

Beschreibung

Berechnet die Quadratwurzel des Wertes in ST(0) und speichert diesen Wert wieder in ST(0).

Beispiele:

```

FLD    qword ptr[a]
FMUL   qword ptr[a] ; ST(0) = a*a
FLD    qword ptr[b]
FMUL   qword ptr[b] ; ST(0) = b*b
FADDP                      ; ST(0) = a*a + b*b
FSQRT                      ; ST(0) = sqrt(a*a + b*b)
  
```

16.13.3. Vergleichsbefehle

FCOM/FCOMP/FCOMPP—Compare Floating Point Values

Instruction	Description
FCOM <i>m32fp</i>	Compare ST(0) with <i>m32fp</i> .
FCOM <i>m64fp</i>	Compare ST(0) with <i>m64fp</i> .
FCOM ST(<i>i</i>)	Compare ST(0) with ST(<i>i</i>).
FCOM	Compare ST(0) with ST(1).
FCOMP <i>m32fp</i>	Compare ST(0) with <i>m32fp</i> and pop register stack.
FCOMP <i>m64fp</i>	Compare ST(0) with <i>m64fp</i> and pop register stack.
FCOMP ST(<i>i</i>)	Compare ST(0) with ST(<i>i</i>) and pop register stack.
FCOMP	Compare ST(0) with ST(1) and pop register stack.
FCOMPP	Compare ST(0) with ST(1) and pop register stack twice.

Beschreibung

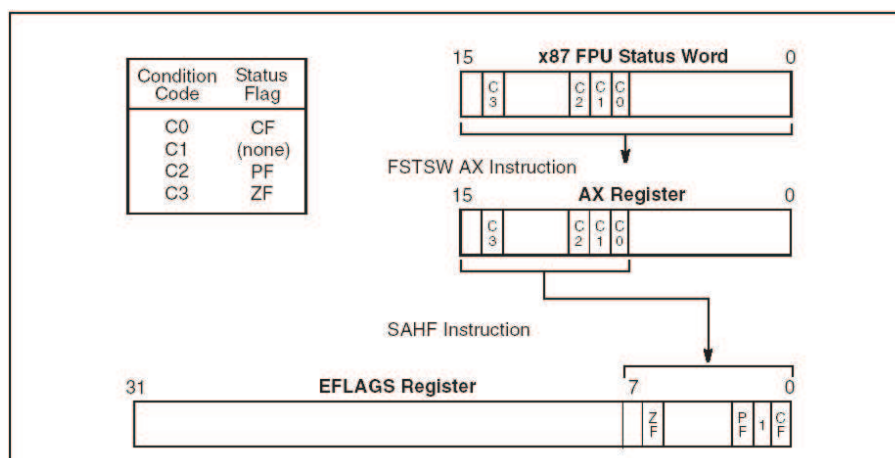
Vergleicht das Register ST(0) mit dem Inhalt des Quelloperanden und setzt die Condition Code Flags C0, C2 und C3 im FPU Statusregister entsprechend dem Vergleichsergebnis gemäß nachfolgender Tabelle. Der Quelloperand kann ein FPU Register, oder eine Speichervariable sein. Die Variante ohne Operanden vergleicht die Register ST(0) mit ST(1). Das Vorzeichen einer Null wird ignoriert, sodass -0.0 gleich $+0.0$ betrachtet wird.

Condition	C3	C2	C0
ST(0) > SRC	0	0	0
ST(0) < SRC	0	0	1
ST(0) = SRC	1	0	0
Unordered*	1	1	1

NOTE: * Die Flags werden nicht verändert, wenn eine "unmasked invalid-arithmetic-operand (#IA) exception" auftritt.

Der FCOMP Befehl räumt nach dem Vergleich den obersten Registerwert wieder vom FPU Registerstack ab, der FCOMPP die beiden obersten Registerinhalte. Dabei wird das (die) oberste(n) Register als ‚leer‘ markiert und dann der Stackpointer TOP um 1 (2) inkrementiert.

Um das Ergebnis dieser Vergleichsoperation mit bedingten Sprüngen (z.B. JA, JE, JB) auszuwerten, müssen die Condition-Flags C0, C1 und C2 in das EFLAGS-Register der Integer-Einheit transferiert werden. Dazu sind die beiden Befehle FSTSW und SAHF vorgesehen.



Beispiel

```

FLD    qword ptr[a]
FLD    qword ptr[b]
FCOMPP
FSTSW ax
SAHF
JA     a_above_b
JE     a_equal_b
JB     a_below_b

```

FICOM/FICOMP—Compare Integer

Instruction	Description
FICOM <i>m16int</i>	Compare ST(0) with <i>m16int</i>
FICOM <i>m32int</i>	Compare ST(0) with <i>m32int</i>
FICOMP <i>m16int</i>	Compare ST(0) with <i>m16int</i> and pop stack register
FICOMP <i>m32int</i>	Compare ST(0) with <i>m32int</i> and pop stack register

Beschreibung

Vergleicht das Register ST(0) mit dem Integer-Quelloperanden und setzt die Condition Code Flags C0, C2 und C3 im FPU Statusregister entsprechend dem Vergleichsergebnis (siehe Tabelle unter FCOM, FCOMP, FCOMPP). Der Integer-Wert wird vor dem Vergleich in das 80bit-Gleitkommaformat umgewandelt.

Der FICOMP Befehl räumt nach dem Vergleich den obersten Registerwert wieder vom FPU Registerstack ab. Dabei wird das oberste Register als ‚leer‘ markiert und dann der Stackpointer TOP inkrementiert.

FCOMI/FCOMIP/FUCOMI/FUCOMIP - Compare Floating Point Values and Set EFLAGS

Instruction	Description
FCOMI ST, ST(i)	Compare ST(0) with ST(i) and set status flags accordingly
FCOMIP ST, ST(i)	Compare ST(0) with ST(i), set status flags accordingly, and pop register stack
FUCOMI ST, ST(i)	Compare ST(0) with ST(i), check for ordered values, and set status flags accordingly
FUCOMIP ST, ST(i)	Compare ST(0) with ST(i), check for ordered values, set status flags accordingly, and pop register stack

Beschreibung

Vergleicht das Register ST(0) mit dem Register ST(1) und setzt das Zero-Flag ZF, Parity-Flag PF und Carry-Flag CF im EFLAGS Register der Integer-Einheit entsprechend dem Ergebnis des Vergleichs (siehe nachfolgenden Tabelle). Das Overflow Flag OF wird immer gelöscht.

Condition	ZF = Zero-Flag	PF = Parity-Flag	CF = Carry-Flag
ST(0) > SRC	0	0	0
ST(0) < SRC	0	0	1
ST(0) = SRC	1	0	0
Unordered*	1	1	1

NOTE: * Die Flags werden nicht verändert, wenn eine "unmasked invalid-arithmetic-operand (#IA) exception" auftritt.

Der FICOMP Befehl und FUCOMP Befehl räumt nach dem Vergleich den obersten Registerwert wieder vom FPU Registerstack ab. Dabei wird das oberste Register als ‚leer‘ markiert und dann der Stackpointer TOP inkrementiert.

Die FCOMI/FCOMIP/FUCOMI/FUCOMIP Befehle wurden mit der P6 Prozessorfamilie (Pentium II) eingeführt und stehen bei älteren Prozessoren nicht zur Verfügung.

Beispiel

```
FLD    qword ptr[a]
FLD    qword ptr[b]
FCOMI  ST(0),ST(1)
JA     a_above_b
JE     a_equal_b
JB     a_below_b
```

16.13.4. Trigonometrische Befehle

FSIN/FCOS/FSINCOS—Sine/Cosine/Sine and Cosine

Instruction	Description
FSIN	Replace ST(0) with its sine.
FCOS	Replace ST(0) with its cosine
FSINCOS	Compute the sine and cosine of ST(0); replace ST(0) with the sine, and push the cosine onto the register stack.

Beschreibung

FSIN berechnet den Sinus und FCOS berechnet den Cosinus des Quelloperanden ST(0) und speichert das Ergebnis wieder in ST(0). Der Quelloperand muss im Bogenmaß (Radian) angegeben werden im Bereich -2^{63} to 2^{63} .

FSINCOS berechnet Sinus und Cosinus gleichzeitig, speichert den Sinus in ST(1) und den Cosinus Wert in ST(0). Dieser Befehl ist schneller als die Ausführung von FSIN und FCOS nacheinander.

Beispiel:

```
PUSH    6
FLDPI                               ; ST(0) =  $\pi$ 
FIDIV   dword ptr[esp]; ST(0) =  $\pi/6$ 
FSINCOS                               ; ST(1) =  $\sin(\pi/6) = 0.5$ ; ST(0) =  $\cos(\pi/6) = 0.866$ 
POP     eax                          ; undo PUSH 6
```

FPTAN/FPATAN—Partial Tangent/ Partial Arctangent

Instruction	Description
FPTAN	Replace ST(0) with its tangent and push 1 onto the FPU stack.
FPATAN	Replace ST(1) with $\arctan(ST(1)/ST(0))$ and pop the register stack

Beschreibung

FPTAN berechnet den Tangens des Quelloperanden ST(0) und speichert das Ergebnis in ST(0). Danach wird eine 1.0 auf den FPU Registerstack gepusht. Der Quelloperand muss im Bogenmaß (Radian) angegeben werden im Bereich $\pm 2^{63}$.

Betrachtet man die resultierenden Werte als $ST(0)=X$ - und $ST(1)=Y$ -Koordinatenpaar, so besitzt der Vektor vom Nullpunkt durch den Punkt X/Y den Steigungswinkel und die Orientierung gemäß dem Quelloperanden $ST(0)$.

FPATAN berechnet den Arcustangens des Quelloperanden in $ST(1)$ geteilt durch den Quelloperanden in $ST(0)$, speichert das Ergebnis in $ST(1)$ mit anschließendem POP des FPU Registerstack. Das Ergebnis im Register $ST(0)$ hat das gleiche Vorzeichen wie der Quelloperand $ST(1)$ und einen Betrag kleiner als π .

Der FPATAN Befehl ermittelt den Winkel zwischen der X-Achse und dem Vektor vom Ursprung zum Punkt (X,Y) , wobei Y (die Ordinate) in $ST(1)$ und X (die Abszisse) in $ST(0)$ steht. Der resultierende Winkel hängt von den Vorzeichen der Werte X und Y ab (und nicht von dem Vorzeichen des Quotienten Y/X). So liegt der Punkt $(-X,Y)$ im zweiten Quadranten und ergibt einen Winkel zwischen $\pi/2$ und π , während ein Punkt $(-X,Y)$ im vierten Quadranten liegt, d.h. im Winkelbereich von 0 bis $-\pi/2$. Ein Punkt $(-X,-Y)$ liegt im dritten Quadranten mit einem Winkel zwischen $-\pi/2$ und $-\pi$.

Beispiel:

```

FLD1
FADD ST(0), ST(0)
FLD ST(0)
FCHS          ; ST(0) = -2 ; ST(1) = 2
FPATAN        ; ST(0) = 3/4  $\pi$ 
FPTAN         ; ST(0) = -1 ; ST(1) = 1
  
```

16.13.5. Logarithmus- und Exponential-Befehle

FYL2X—Compute $y * \log_2 x$

Instruction	Description
FYL2X	Replace $ST(1)$ with $(ST(1) * \log_2 ST(0))$ and pop the register stack

Beschreibung

Berechnet die Formel $Y * \log_2(X)$ mit $ST(1)=Y$ und $ST(0)=X$. Das Ergebnis wird gespeichert im Register $ST(1)$, mit anschließendem POP nach $ST(0)$. Der Quelloperand X in $ST(0)$ muss eine positive Zahl (>0) sein.

Der FYL2X Befehl ist dazu gedacht, die Umrechnung von Logarithmen mit unterschiedlicher Basis (b) zu vereinfachen:

$$\log_b(x) = \log_2(x) / \log_2(b)$$

Beispiel

$$2^{10} = 10^{3.01029995663981195213738894724493} = 1.024 * 10^3$$

FYL2XP1—Compute $y * \log_2(x + 1)$

Instruction	Description
FYL2XP1	Replace ST(1) with $ST(1) * \log_2(ST(0) + 1.0)$ and pop the register stack

Beschreibung

Berechnet die Formel $Y * \log_2(X + 1.0)$ mit $ST(1) = Y$ und $ST(0) = X$. Das Ergebnis wird im Register ST(1) gespeichert, mit anschließendem POP nach ST(0). Der Quelloperand X in ST(0) muss im Bereich $-(1 - 1/\sqrt{2})$ bis $+(1 - 1/\sqrt{2})$ liegen.

Dieser Befehl ergibt eine maximale Genauigkeit für Werte von epsilon [der Wert im Register ST(0)] die nahe bei 0 liegen. Für sehr kleine epsilon (ϵ) Werte können mehr signifikante Nachkommastellen berechnet werden, wenn man den FYL2XP1 Befehl verwendet, statt den Wert ($\epsilon + 1$) als Argument für FYL2X zu benutzen. Der Term ($\epsilon + 1$) wird häufig in Zinseszins und Rentenberechnungen verwendet.

Das Ergebnis kann leicht in einen Logarithmenwert zu einer anderen Basis (z.B. e oder 10) konvertiert werden durch Einsetzen des entsprechenden Skalierungsfaktors im ST(1) Operanden. Die folgende Gleichung kann benutzt werden, um den Skalierungsfaktor für eine andere Logarithmenbasis (n) zu berechnen:

$$\text{scale factor} = \log_n(2).$$

F2XM1—Compute $2^X - 1$

Instruction	Description
F2XM1	Replace ST(0) with $(2^{ST(0)} - 1)$

Beschreibung

Berechnet den Exponentialwert von 2 hoch X minus 1. Der Quelloperand X liegt im Register ST(0), das Ergebnis wird ebenfalls im Register ST(0) abgelegt. Der Wert des Quelloperanden in ST(0) muss im Bereich -1.0 bis $+1.0$ liegen, ansonsten ist das Ergebnis undefiniert.

Beispiel:

```

FLD  [zehn]      ; ST(0) = 0.5
F2XM1             ; ST(0) = (2.00.5 - 1) = ( $\sqrt{2} - 1$ ) = 0,41421356
FLD1
FADDP ST(1),ST(0) ; ST(0) =  $\sqrt{2}$  = 1,41421356
  
```

FXTRACT—Extract Exponent and Significand

Instruction	Description
FXTRACT	Separate value in ST(0) into exponent and significand, store exponent in ST(0), and push the significand onto the register stack

Beschreibung

Zerlegt den Quelloperanden im ST(0) Register in Exponent und Mantisse, speichert den Exponenten in ST(0). Danach wird die Mantisse auf den FPU Registerstack gepusht. Als Ergebnis enthält ST(0) die Mantisse und ST(1) den Exponenten.

Der FXTRACT und der F2XM1 Befehl sind sehr hilfreich bei Exponential- und Bereichs-skalierungs-Berechnungen. Der FXTRACT Befehl ist ebenfalls nützlich zur Konvertierung von Gleitkommazahlen in eine Dezimaldarstellung, wie z.B. zur Bildschirmausgabe.

Beispiel:

```

FLD  [zehn]      ; ST(0) = 10.0
F2XM1            ; ST(0) = (2.010.0-1) = 1023.0
FLD1
FADDP            ; ST(0) = 1024.0 = 2.010.0
FXTRACT          ; ST(0) = 2.0
  
```

FSCALE—Scale

Instruction	Description
FSCALE	Scale ST(0) by ST(1).

Beschreibung

Rundet den Wert im Quelloperanden ST(1) ab auf einen ganzzahligen Wert und addiert diesen Wert zum Exponenten des Zieloperanden ST(0).

Formel: $ST(0) = X * 2^{\text{int}(Y)}$ mit $X=ST(0)$, $Y=ST(1)$

Dieser Befehl erlaubt eine schnelle Multiplikation oder Division mit ganzzahligen 2er-Potenzen.

In den meisten Fällen wird nur der Exponent verändert und die Mantisse bleibt erhalten. Nur in Ausnahmefällen bei einem nicht normalisierten Wert in ST(0) wird die Mantisse verändert.

Der FSCALE Befehl kann dazu verwendet werden, das Ergebnis eines FXTRACT Befehls umzukehren. Die Kombination von FXTRACT und FSCALE ermöglicht es daher, die Multiplikation von 2er-Potenzen in Form einfacher Logarithmen-Addition durchzuführen.

Beispiel

```

FXTRACT      ;
FSCALE       ;
FSTP ST(1)   ;
  
```