Programare in limbaj de asamblare

Instructiuni in virgula mobila

Aritmetica in virgula mobila

Reprezentarea in virgula mobila

- Semn+Exponent+Mantisa
- Extinde posibilitatile de reprezentare ale numerelor: foarte mari & foarte mici (fractionare)
- NU respecta algebra numerelor reale:
  - numar finit de valori posibile
  - rezolutie si acuratete limitata (numar limitat de cifre semnificative)
  - anomalii la calculul operatiilor aritmetice:
    - truncheri, rotunjiri, depasirea capacitatii de reprez.

Anomalii si moduri de evitare a acestora

- adunarea si scaderea unor numere cu magnitudini diferite - numerele f. mici se pierd la trunchiere - gruparea operanzilor in raport de magnitudine
- compararea numerelor - diferente datorate rotunjirilor sau trunchierilor - comparare cu marja de eroare
- modelarea valorilor limita : 0, $+\infty$, $-\infty$
- Exemplu:

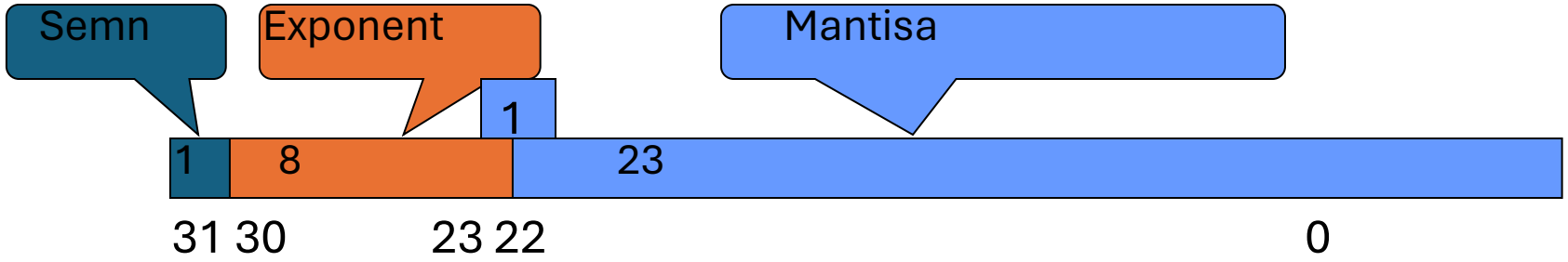
a = 0.23+0.26 = 0.4900000000000000188

b = 0.24+0.25 = 0.48999999999999911

if (a==b) ; poate fi fals!

Formate standard pt. virgula mobila (Standardul Intel - IEEE)

- Intel foloseste 3 formate: precizie simpla (32 biti) , dubla (64 biti) si extinsa (80 biti)



- Simpla precizie:
  - semn - 1 bit (0 -pozitiv; 1- negativ)
  - mantisa - 23+1 biti (m_{24} = 1, nu se reprezinta)
  - exponent - 8 biti - exces 127 (exponent + 127)

Simpla precizie

- mantisa $\in$ [1.000 - 2.000)
  - cifrele semnificative se reprezinta in format normalizat (exceptie: valoarea 0)
  - bitul din stanga (=1) nu se reprezinta
  - aprox. 6,5 cifre zecimale semnificative
- exponentul - reprezentat in codul exces 127
  - la valoarea exponentului se adauga 127 (ex: exponentul 0 se reprezinta prin valoarea 127)
  - se simplifica operatiile de comparare

Formate in virgula mobila

- Dubla precizie: 64 biti

  1 bit semn + 11 biti exponent+ (52+1) biti mantisa

- Precizie extinsa: 80 biti

  1 bit semn + 15 biti exponent + 64 biti mantisa

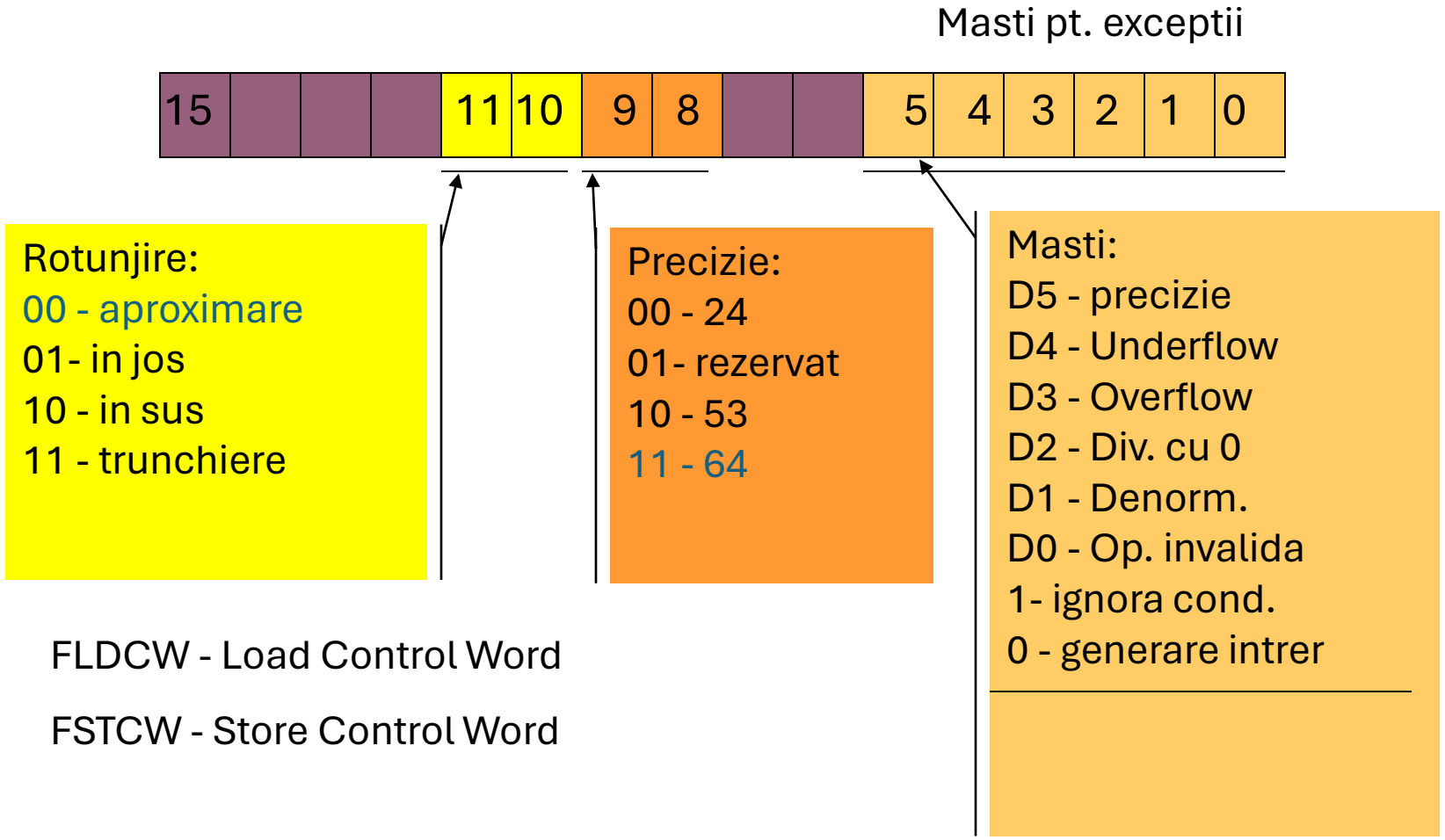
  - fara bit implicit pt. mantisa
  - format folosit pentru rezultate intermediare
  - toate calulele interne se fac in precizie extinsa
  - ajuta la evaluarea mai precisa a rotunjirilor

Unitatea de executie in virgula mobila - registre

- Registre de date - 8 reg*80 biti organizate ca o stiva
- Denumire: ST(0) - ST(7);
- ST(0) - varful stivei; ST(1) penultima data
- adresare relativa

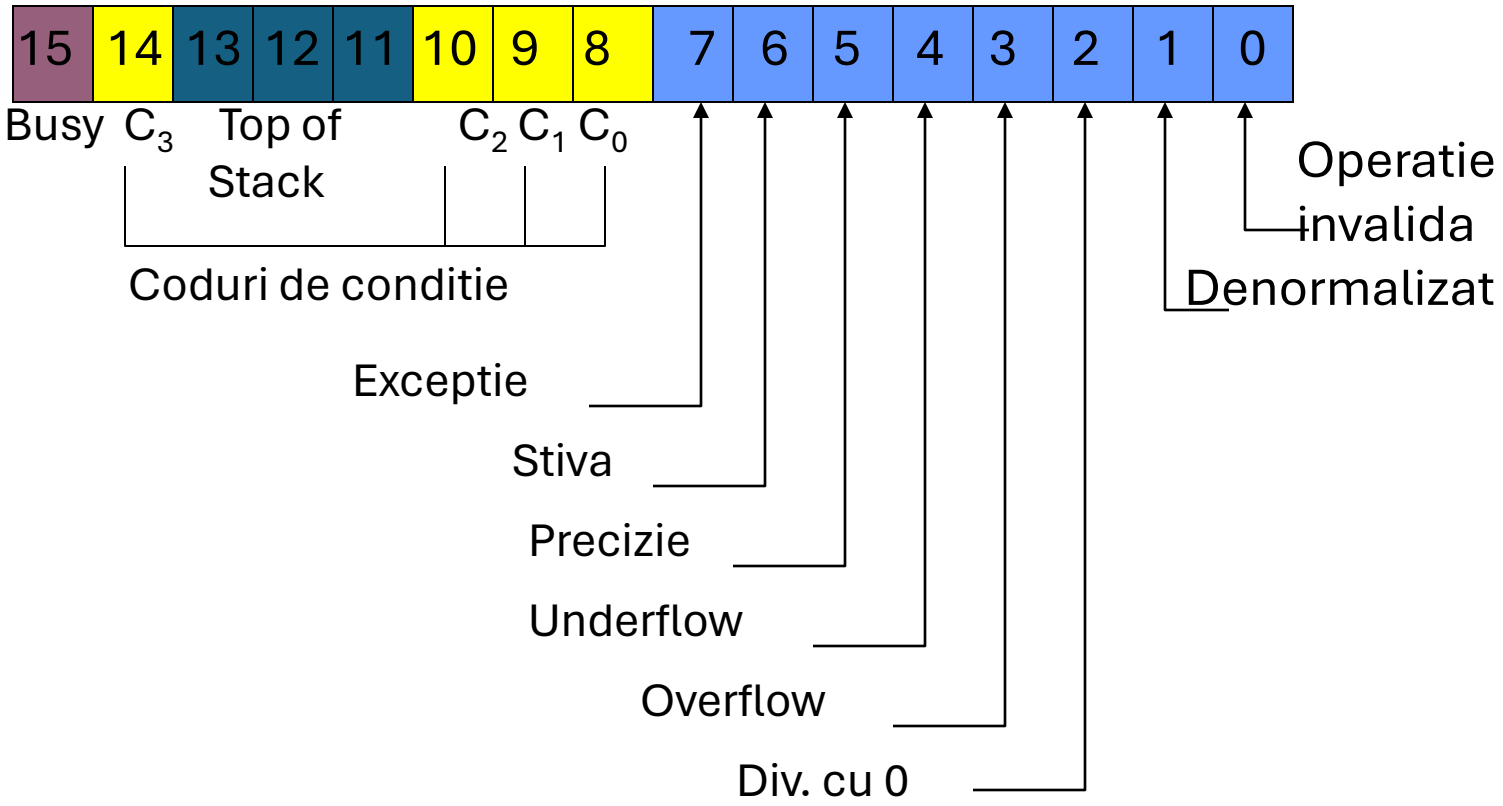
Unitatea de executie in virgula mobila - registre

- Registrul de control:



Registrele coprocesorului matematic

- Registrul de stare:



Tipuri de date acceptate de FPU

- Intregi:
  - complement fata de 2 pe 16, 32 si 64 biti
- Zecimal impachetat
  - semn + 17 cifre zecimale (80 biti)
- Flotant:
  - precizie simpla (32), dubla (64) extinsa (80)

Instructiunile unitatii de executie in virgula mobila

Instructiuni de transfer

- **Scriere pe stiva: FLD**
  - decrementeaza ind. de stiva, apoi incarca un operand pe 32,64 sau 80 biti pe varful stivei;
  - valoarea este extinsa la 80 biti

    fld st(1)

    fld mem_32

    fld MyRealVar

    fld mem_64[bx]
  - ex: fld st(0)   ; dubleaza valoarea de pe varful stivei

Instructiuni de transfer

- **Citire de pe stiva FST si FSTP:**
  - incarca un operand pe 32,64 sau 80 biti de pe varful stivei intr-un registru sau intr-o locatie de memorie
  - valoarea citita este rotunjita
  - la instructiunea FSTP, dupa citire, indicatorul de stiva se incrementeaza (operatie “POP”)

```
fst mem_32                  fst mem_80
fstp mem_64                 fst st(2)
fstp mem_64[ebx*8]          fstp st(1)
```

Instructiuni de transfer

- **instructiunea FXCH:** schimba continutul varfului de stiva cu un alt registru

```
fxch  st(2)        ; st(0)<=>st(2)
fxch                ; st(0)<=>st(1)
```

Instructiuni de conversie

- Convertesc din intreg sau BCD in flotant si invers
- Conversie intreg<=>flotant
  - **Instructiunea FILD**: incarca un intreg pe 16,32 sau 64 biti pe stiva cu conversie din intreg (C2) in flotant extins

    fild mem_16

    fild mem_32[ecx*4]

    fild mem_64[ebx+ecx*8]
  - **Instructiunile FIST si FISTP**: descarca un flotant extins de pe stiva si il converteste in intreg pe 16,32, 64

    fist mem_16[bx]

    fist mem_64

    fistp mem_32

Instructiuni de conversie

- Conversie flotant <=> BCD
  - **instructiunea FBLD**: converteste BCD (80biti) in flotant extins si pune valoarea pe stiva
  - **instructiunea FBSTP:** converteste flotant extins in BCD (80 biti) si descarca stiva
  - exemplu: conversie BCD - intreg pe 64 biti

```
fbld    bcd_80 ; conversie BCD=> flotant.
fist    mem_64 ;conversie flotant=> intreg.
```

Instructiuni aritmetice

- **instructiuni de adunare: FADD, FADDP**

```
fadd                ;pop st(0); pop st(1); push st(0)+st(1)
faddp               ; se pune tot pe stiva
fadd st( i), st(0)  ; st(i)=st(i)+st(0)
fadd st(0), st(i)   ; st(0)=st(0)+st(i)
faddp st( i), st(0) ; st(i)=st(i)+st(0) si pop st(0)
fadd mem            ; st(0)=st(0)+mem(32 sau 64 flotant)
```

Instructiuni aritmetice

- **Instructiuni de scadere: FSUB, FSUBP, FSUBR, si FSUBRP**

```
fsub sau   fsubp     ; pop st(0);pop st(1); push st(1)-st(0)
fsubr  sau  fsubrp ; pop st(0);pop st(1); push st(0)-st(1)
fsub st( i). st(0)     ; st(i)=st(i)-st(0)
fsub st(0), st(i)      ; st(0)=st(0)-st(i)
fsubp st(i), st(0)    ; st(i)=st(i)-st(0) si pop st(0)
fsub mem               ; st(0)=st(0)-mem
fsubr st( i), st(0)    ; st(i) = st(0)-st(i)
fsubrp st( i), st(0)
fsubr st(0), st(i)                fsubr mem
```

Instructiuni aritmetice

- **Instructiuni de inmultire FMUL, FMULP**

```
fmul                ; pop st(0); pop st(1); push st(0)*st(1)
fmulp               ;idem
fmul st(0), st(i)   ; st(0)=st(0)*st(i)
fmul st( i), st(0)  ; st(i)=st(i)*st(0)
fmul mem                    ; st(0)=st(0)*mem(32 sau 64)
fmulp st( i), st(0) ; st(i)=st(i)*st(0) si pop st(0)
```

Instructiuni aritmetice

- **Instructiuni de impartire:FDIV, FDIVP, FDIVR, si FDIVRP**

fdiv (st(0):=st(1)/st(0))
fdivp
fdivr (st(0):= st(0)/st(1))
fdivrp

fdiv st(0), st(i)
fdiv st(i), st(0)
fdivp st(i), st(0)
fdivr st(0), st(i)
fdivr st(i), st(0)
fdivrp st(i), st(0)

fdiv mem
fdivr mem

Instructiuni aritmetice

- Radacina patrata - **FSQRT**
  - calculeaza radacina patrata din valoarea continuta in varful stivei; rezultatul se pune in ST(0)
  - nu are parametri
  - exemplu: z= sqrt (x*x+y*y)

```
fld   x
fld   st(0)
fmul          ;x*x
fld     y
fld  st(0)
fmul          ;y*y
fadd      ;x*x+y*y
fsqrt     ;sqrt(x*x+y*y)
fst    z
```

Instructiuni aritmetice

- Scalare **FSCALE**
  - descarca 2 valori de pe stiva si reincarca valoarea:

  st(0)*(2^st(1))
  - inmultire sau impartire cu puteri intregi ale lui 2
  - daca st(1) nu este intreg atunci se trunchiaza la zero
  - exemplu:

```
var_intreaga   word   16
fild   var_intreaga
fld    x
fscale       ; x*(2^16)
```

Instructiuni aritmetice

- Rest partial: **FPREM1**
  - calculeaza restul impartirii st(0)/st(1),

  (calculeaza corect restul daca exp(st(0))-exp(st(1))<64, altfel operatia trebuie repetata)
  - **nu** descarca 2 valori de pe stiva,
  - rezultatul se pune in st(0)
- Rotunjire: **FRNDINT**
  - rotunjeste valoarea din varful stivei la un intreg, conform schemei de rotunjire indicate prin registrul de control

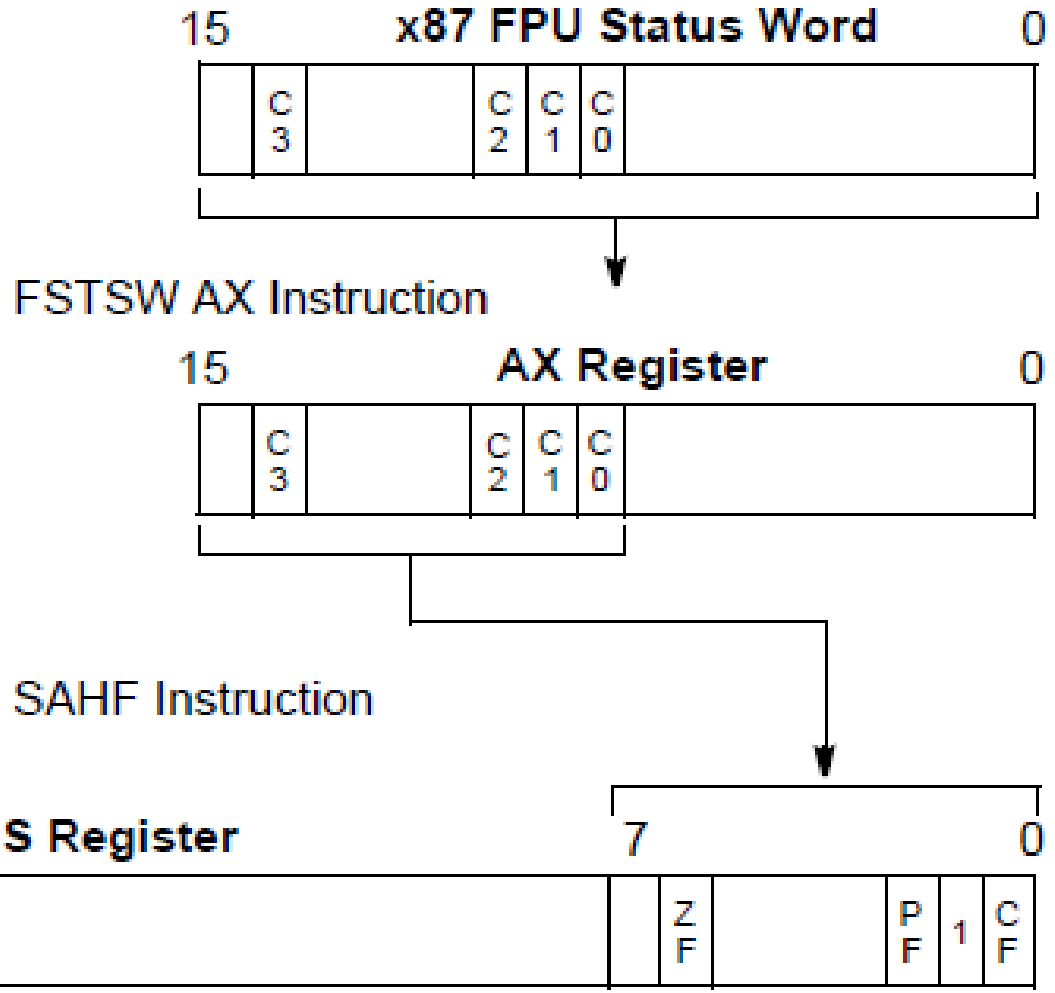
Instructiuni aritmetice

- Valoare absoluta: **FABS**
  - transforma valoarea din varful stivei intr-un numar pozitiv, prin stergerea bitului de semn
  - st(0) = abs(st(0))
- Schimbare semn: **FCHS**
  - schimba semnul valorii din varful stivei
  - st(0)= - st(0)

Instructiuni de comparare

- se compara 2 valori de pe varful stivei si se seteaza corespunzator indicatorii de conditie din registrul de stare al coprocesorului
- nu exista salt conditionat bazat pe acesti indicatori;
- indicatorii trebuie copiati in registrul AX (cu FSTSW) si apoi din AH in registrul de stare al procesorului x86 (cu SAHF)
- echivalare indicatori: C0=>CF, C1=>?, C2=>PF, C3=>ZF
- se pot utiliza numai salturile conditionate pentru intregi fara semn !!!!: JA, JAE, JBE, JB, JE, JZ

Condition Code	Status Flag
C0	CF
C1	(none)
C2	PF
C3	ZF



Instructiuni de comparare

- **FCOM, FCOMP, FCOMPP**
  - sintaxa:

```
fcom            ; compara st(0) si st(1)
fcomp           ; idem, + descarca st(0)
fcompp          ; idem,  + descarca st(0) si st(1)
fcom st( i)     ; compara st(0) cu st(i)
fcomp st( i)    ; idem, + descarca st(0)
fcom mem        ; compara st(0) si mem
fcomp mem       ; idem + descarca st(0)
```

Instructiuni de comparare

- **FUCOM, FUCOMP, FUCOMPP**
  - sintaxa:

    fucom

    fucomp

    fucompp

    fucom st(i)

    fucomp st(i)
  - identice cu cele anterioare, dar nu genereaza exceptie in cazul compararii unor valori care nu sunt numere (exceptie NAN)

Instructiuni de comparare

- **FTST**
  - compara st(0) cu 0.0
  - nu face diferenta intre + si - 0.0
  - nu descarca stiva
- **FXAM**
  - examineaza valoarea din st(0) si seteaza corespunzator indicatorii de conditie
  - nu descarca stiva

Instructiuni de comparare noi

- Familia de procesoare P6
- Comparare valori reprezentate in virgula mobile si setare direct EFLAGS
- FCOMI, FCOMIP, FUCOMI, and FUCOMIP
- Flags afectate: ZF, PF, CF

Condition	C3	C2	C0
ST(0) > Source Operand	0	0	0
ST(0) < Source Operand	0	0	1
ST(0) = Source Operand	1	0	0
Unordered	1	1	1

Comparison Results	ZF	PF	CF
ST0 > ST(*i*)	0	0	0
ST0 < ST(*i*)	0	0	1
ST0 = ST(*i*)	1	0	0
Unordered	1	1	1

Incarcare constante

- permit incarcarea pe stiva a unor constante utilizate frecvent

```
fldz      ; +0.0.
fld1       ; +1.0.
fldpi     ; pi
fldl2t    ; log 2 (10).
fldl2e    ; log 2 (e).
fldlg2   ; log 10 (2).
fldln2   ; ln(2).
```

Instructiuni pentru functii logaritmice si trigonometrice

- **F2XM1**
  - calculeaza: $2^{st(0)} - 1$
  - st(0) trebuie sa fie in intervalul: [-1.0 .. +1.0]
  - exemplu: $10^x = 2^{x*lg2(10)}$

```
fld   x
fldl2t
fmul
f2xm1
fld1
fadd
```

Instructiuni logaritmice

- **FYL2X**
  - calculeaza: st(1)*lg_2(st(0))
  - util in calculul unor logaritmi in alta baza decat 2
  - st(0) trebuie sa fie >0
- **FYL2P1**
  - calculeaza: st(1)*lg_2(st(0)+1)

Functii trigonometrice

- **FSIN, FCOS, FSINCOS**
  - calculeaza sinusul, cosinusul sau ambele functii pt. valoarea din varful stivei; rezultatul se pune in varful stivei
  - FSIN: st(0)=sin(st(0))
  - FCOS: st(0)=cos(st(0))
  - FSINCOS st(0)= cos(st(0)) si st(1)=sin(st(0))
  - unghiurile se considera in radiani si trebuie sa fie in intervalul $(-2^{63} ..+2^{63})$

Instructiuni trigonometrice

- **FPTAN**
  - calculeaza tangenta din st(0)
  - rezultatul si apoi valoarea 1.0 se pun pe stiva
- **FPATAN**
  - calculeaza arctangenta din st(0)/st(1)
  - extrage 2 valori de pe stiva si pune inapoi o valoare

Alte instructiuni

- FINIT - initializare coprocesor
  - aduce coprocesorul intr-o stare cunoscuta:
    - reg. de control (CW)= 37H
    - reg. de stare (SW) = 0
- FWAIT - forteaza procesorul x86 sa astepte terminarea instructiunii curente din coprocesor
  - evita executia in paralel a unor operatii care afecteaza aceleasi variabile

Operatii cu registrele speciale

- FLDCW si FSTCW - incarcare si salvare registru de control
  - sintaxa: fldcw mem_16 ; CW<=mem_16
    fstcw mem_16 ; mem_16<=CW

- FLDSW si FSTSW - incarcare si salvare registru de stare
  - sintaxa: fldsw ax ; SW<=AX
    fstsw ax ; AX<=SW
    fldcw mem_16 ; CW<=mem_16
    fstcw mem_16 ; mem_16<=CW

Salvare si refacere stare coprocesor

- FLDENV si FSTENV - incarca si salveaza “mediul” de lucru al coprocesorului
  - se transfera 14 octeti avand urmatorul format:
    - reg. de control (CW) - 2 octeti
    - reg. de stare (SW) - 2 octeti
    - cuvantul atasat (tag) - 2 octeti
    - poantor de instructiuni - 20 biti
    - codul de instructiune 11 biti
    - poantor de date 20 biti
    - restul nefolositi

Operatii cu registrele speciale

- FSAVE/FNSAVE si FRSTORE
  - salveaza si respectiv refac “starea” coprocesorului, adica registrele de control, stare si de date; folosite pt. comutare de context
  - se transfera 94 octeti =
    - 14 octeti pt. “mediu” +
    - 80 octeti - continutul stivei (ST(0) ... ST(7))
  - sintaxa: fsave mem_94_oct
    fnsave mem_94_oct
    frstore mem_94_oct

Alte instructiuni

- FINCSTP si FDECSTP - incrementare si decrementare poantor de stiva (modulo 8)
- FNOP - no operation - pt. temporizare
- FFREE - modifica tag-ul atasat unui registru, pentru a marca faptul ca registru este gol
  - sintaxa:

  ffree st(i)   ; marcheaza reg. i ca fiind gol
- FCLEX/FNCLEX - sterge toti bitii de exceptie, indicatorul Busy si eroarea de stiva

Instructiuni cu intregi

- inainte de operatia aritmetica valoarea intreaga este convertita in virgula flotanta pe 80 biti
    - sintaxa:

fiadd int fidiv int

fisub int fidivr int

fisubr int ficom int

fimul int ficomp int

- int - intreg pe 16 sau 32 biti

Exemple

- Calculul cotangentei

```
fld   arc
fsincos
fdivr
fst   cotangenta
```

- Arccotangenta actg(x)=atg(1/x)

```
fld1                ; pune “1” pe stiva
fld   cotang
fpatan              ; atan(st(1)/st(0))
fst         arc
```

Exemple

- Arcsinus: asin(x) = atan(sqrt(x*x/(1-x*x)))

```
fld    sinus
fld st(0)  ;Duplica x pe stiva
fmul                ; calculeaza x*x.
fld st(0)  ;Duplica x*x pe stiva.
fld1                ;Calculeaza 1-X**2.
fsubr
fdiv                ;Calculeaza X**2/(1-X**2).
fsqrt               ;Calculeaza sqrt(x**2/(1-X**2)).
fld1                ;Calculeaza  arctangent.
fpatan
```

Exemple

- Logaritm in baza 10 din x
  - $lg_{10}(x)=lg_2(x)/lg_2(10)$

```
fld1
fld         x
fyl2x                   ;Calculeaza 1*lg(x).
fldl2t              ;Incarca lg(10).
fdiv                    ;Calculeaza lg(x)/lg(10).
```

Exercitii

Sa se scrie o functie care calculeaza expresia R=(A^2-1.33*B)/$C^{1/2}$ (numerele sunt reprezentate in virgula mobila pe 64 de biti), adresele operanzilor sunt transmise ca si parametru.

Sa se scrie o functie care returneaza A^2 daca A<1 si $\sqrt{A}$ in rest. A este reprezentat in virgula mobila pe 64 de biti. Adresa lui A este transmisa ca parametru functiei.