Lab 8 – FPU

- **Floating Point Unit**
- **Koprocesor – umożliwia działania na liczbach ułamkowych (zmiennoprzecinkowych)**
- **8 rejestrów 80-bitowych: st0,…,st7**
- **Typy danych**
  - Pojedyncza precyzja. Liczby takie zajmują po 32 bity (4 bajty) i ich wartość maksymalna wynosi ok. 10^{39} (10^39). Znane są programistom języka C jako float.
  - Podwójna precyzja. 64 bity (8 bajtów), max = ok. 10^{409} (10^409). W języku C są znane jako double
  - Rozszerzona precyzja. 80 bitów (10 bajtów), max = ok. 10^{4930} (10^4930). W języku C są to long double

- Instrukcje przemieszczenia danych:
  - FLD/FILD [mem] - załaduj liczbę rzeczywistą/całkowitą z pamięci. Dla liczby rzeczywistej jest to 32, 64 lub 80 bitów. Dla całkowitej - 16, 32 lub 64 bity.
  - FST [mem32/64/80] - do pamięci idzie liczba ze st(0).
  - FSTP [mem32/64/80] - zapisz st(0) w pamięci i zdejmij je ze stosu. Znaczy to tyle, że st(1) o ile istnieje, staje się st(0) itd. każdy rejestr cofa się o 1.
  - FIST [mem16/32] - ewentualnie obciętą do całkowitej liczbę z st(0) zapisz do pamięci.
  - FISTP [mem16/32/64] - jak wyżej, tylko ze zdjęciem ze stosu.
  - FXCH st(i) - zamień st(0) z st(i).

- Instrukcje ładowania stałych
  - FLDZ - załaduj zero. st(0) = 0.0
  - FLD1 - załaduj 1. st(0) = 1.0
  - FLDPI - załaduj pi.
  - FLDL2T - załaduj log2(10)
  - FLDL2E - załaduj log2(e)
  - FLDLG2 - załaduj log(2)=log10(2)
  - FLDLN2 - załaduj ln(2)

- Działania matematyczne:

- dodawanie: FADD, składnia identyczna jak w odejmowaniu prostym

- odejmowanie:

  - FSUB [mem32/64] st := st-[mem]
  - FSUB st(0),st(i) st := st-st(i)
  - FSUB st(i),st(0) st(i) := st(i)-st(0)
  - FSUBP st(i), st(0) st(i) := st(i)-st(0) i zdejmij
  - FSUBP (bez argumentów) = FSUBP st(1),st(0)
  - FISUB [mem16/32int] st := st-[mem]

- odejmowanie odwrotne:

  - FSUBR [mem32/64] st := [mem]-st(0)
  - FSUBR st(0),st(i) st := st(i)-st(0)
  - FSUBR st(i),st(0) st(i) := st(0)-st(i)
  - FSUBRP st(i),st(0) st(i) := st(0)-st(i) i zdejmij
  - FSUBRP (bez argumentów) = FSUBRP st(1),st(0)
  - FISUBR [mem16/32int] st := [mem]-st

- mnożenie: FMUL, składnia identyczna jak w odejmowaniu prostym.

- dzielenie: FDIV, składnia identyczna jak w odejmowaniu prostym.
- dzielenie odwrotne: FDIVR, składnia identyczna jak w odejmowaniu odwrotnym.

- wartość bezwzględna: FABS (bez argumentów) zastępuje st(0) jego wartością bezwzględną.
- zmiana znaku: FCHS: st(0) := -st(0).
- pierwiastek kwadratowy: FSQRT: st(0) := SQRT[st(0)]
- reszty z dzielenia: FPREM, FPREM1 st(0) := st(0) mod st(1).
- zaokrąglanie do liczby całkowitej: FRNDINT: st(0) := (int)st(0).

- Komendy porównania:
  - FCOM st(n)/[mem] - porównaj st(0) z st(n) (lub zmienną w pamięci) bez zdejmowania st(0) ze stosu FPU
  - FCOMP st(n)/[mem] - porównaj st(0) z st(n) (lub zmienną w pamięci) i zdejmij st(0)
  - FCOMPP - porównaj st(0) z st(1) i zdejmij oba ze stosu
  - FICOM [mem] - porównaj st(0) ze zmienną całkowitą 16- lub 32-bitową w pamięci
  - FICOMP [mem] - porównaj st(0) ze zmienną całkowitą 16- lub 32-bitową w pamięci, zdejmij st(0)
  - FCOMI st(0), st(n) - porównaj st(0) z st(n) i ustaw flagi *procesora*, nie tylko FPU
  - FCOMIP st(0), st(n) - porównaj st(0) z st(n) i ustaw flagi *procesora*, nie tylko FPU, zdejmij st(0)

- Instrukcje trygonometryczne:
  - FSIN - st(0) := sinus [st(0)]
  - FCOS - st(0) := kosinus [st(0)]
  - FSINCOS - st(0) := kosinus [st(0)], st(1) := sinus [st(0)]
  - FPTAN - partial tangent = tangens st(0) := tg [st(0)]
  - FPATAN - arcus tangens st(0) := arctg [st(0)]
- Logarytmiczne, wykładnicze:
  - FYL2X st(1) := st(1)*log2[st(0)] i zdejmij
  - FYL2XPI st(1) := st(1)*log2[st(0) + 1.0] i zdejmij
  - F2XM1 st(0) := 2^[st(0)] - 1

- Instrukcje kontrolne:
  - FINIT/FNINIT - inicjalizacja FPU. Litera N po F oznacza, aby nie brać po uwagę potencjalnych niezałatwionych wyjątków.
  - FLDCW, FSTCW/FNSTCW - Load/Store control word - zapisuje 16 kontrolnych bitów do pamięci, gdzie można je zmieniać na przykład aby zmienić sposób zaokrąglania liczb.
  - FSTSW/FNSTSW - zapisz do pamięci (lub rejestru AX) słowo statusu, czyli stan FPU
  - FCLEX/FNCLEX - wyczyść wyjątki
  - FLDENV, FSTENV/FNSTENV - wczytaj/zapisz środowisko (rejestry stanu, kontrolny i kilka innych, bez rejestrów danych). Wymaga 14 albo 28 bajtów pamięci, w zależności od trybu pracy procesora (rzeczywisty - DOS lub chroniony - Windows/Linux).
  - FRSTOR, FSAVE/FNSAVE - jak wyżej, tylko że z rejestrami danych. Wymaga 94 lub 108 bajtów w pamięci, zależnie od trybu procesora.
  - FINCSTP, FDECSTP - zwiększ/zmniejsz wskaźnik stosu - przesuń st(0) na st(7), st(1) na st(0) itd. oraz w drugą stronę, odpowiednio.
  - FFREE - zwolnij podany rejestr danych
  - FNOP - no operation. Nic nie robi, ale zabiera czas.
  - WAIT/FWAIT - czekaj, aż FPU skończy pracę. Używane do synchronizacji z CPU.

```
;Program ładuje dwie liczby całkowite na stos, odejmuje je i sprawdza, czy wynik jest zerem.

section .text

    org 100h

    start:
        finit ; inicjalizacja stosu FPU
        fild word [label1] ;ładowanie pierwszej liczby całkowitej do rejestru st0
        fild word [label2] ; ładowanie drugiej liczby całkowitej do rejestru st0, pierwsza jest teraz w
    ;st1
        fsub st0,st1 ; odjęcie od siebie dwóch liczb całkowitych, wynik w st0
        ;można też po prostu fsub
        ficom word [zero] ;porównanie zawartości st0 z zerem
        fstsw ax ;zapisanie flag FPU do akumulatora
        sahf ;przepisanie flag FPU do rejestru flagowego

        jne niezero
        mov     ah, 9
        mov     dx, lzero
        int     21h
    niezero:
        mov     ax, 4C00h
        int     21h

    label1  dw      78
    label2  dq  78.0
    wynik   dw      0
    zero    dw  0
    lzero   db  'Wynik jest zerem$'
```

```
;Program ładuje dwie liczby na stos, a następnie sprawdza, czy są one sobie równe.
section .text
org 100h

    start:
        finit ; inicjalizacja stosu FPU
        fld qword [label1] ;ładowanie pierwszej liczby do rejestru st0
        fld qword [label2] ; ładowanie drugiej liczby do rejestru st0, pierwsza jest teraz w st1
        fcomi st0,st1 ; porównanie wartości obu rejestrów i ustawienie flag procesora oraz
    ;FPU
        jnz niezero
        mov     ah, 9
        mov     dx, zero
        int     21h

    niezero:
        mov     ax, 4C00h
        int     21h

    label1 dq       3.1415
    label2  dq  3.1415
    wynik  dq       0.0
    zero    db  'Wynik jest zerem$'
```

```
;Program sprawdzający, czy sinus pi jest rowny 0?

org 100h

start:
	finit				; inicjalizacja stosu

	fldpi				; ładowanie pi do st0
	fsin				; obliczenie sinusa z st0 i zapisanie wyniku w st0
	ftst				; porównanie wartości w st0 z zerem
	fstsw ax			; zapisanie flag z FPU do akumulatora
	sahf				; przepisanie flag z FPU do EFLAGS
	mov ah,9
	je jest_zero
	mov dx,nie_zero
	jmp short pisz
jest_zero:
	mov dx,zero
pisz:
	int 21h

	mov ax,4c00h
	int 21h

nie_zero    db		"To nie jest zero :/$"
zero			db		"To jest zero!$"
```

- Zadanie:

  Napisać program, który mnoży przez siebie dwie liczby i sprawdza, czy wynik jest dodatni, czy ujemny.

- Zadanie:

  Napisać program, który pobierze z klawiatury liczbę rzeczywistą, a następnie odejmie od niej pi i wynik zwróci na ekran. Należy połączyć kod c i asemblera.