Operações em ponto flutuante

O coprocessador X87

- Os membros da família x86 separou um coprocessador matemático que manipula aritmética de ponto flutuante. O coprocessador (FPU) original foi o 8087. Mais tarde, variantes do (FPU) foram incorporadas no microprocessador.

- O microprocessador tem espaço para armazenar números em ponto flutuante.
- O microprocessador tem instruções para manipular números em ponto flutuante.
- A FPU é chamada seção "x87" ou FPU Register Stack, "x87 Stack", e as operações são frequentemente chamadas de "x87 instruction set".

Por que utilizar o x87

- Aplicações que manipulam números reais:
  - Jogos / Simuladores
  - CAD (Computer Aided Design)
  - Aplicações científicas

Representação em ponto flutuante

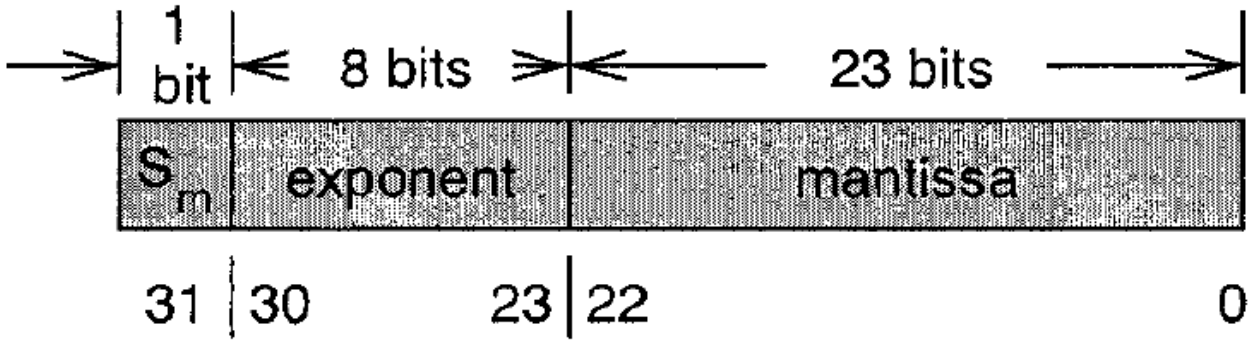
- Números em ponto flutuante podem ser usados para representar números muito pequenos ou números muito grandes.
- É utilizada a notação científica para representar tais números.
- O número é dividido em três partes:

  - Sinal

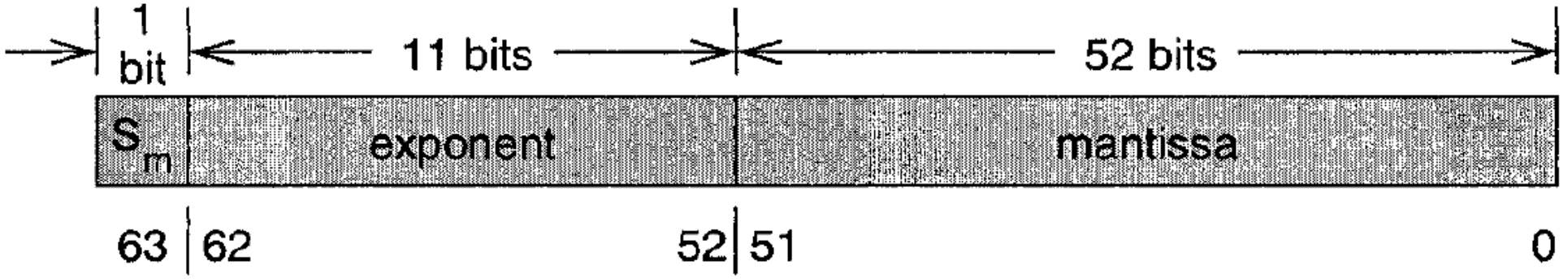
  - mantissa

  - expoente

Representação em ponto flutuante



(a)



(b)

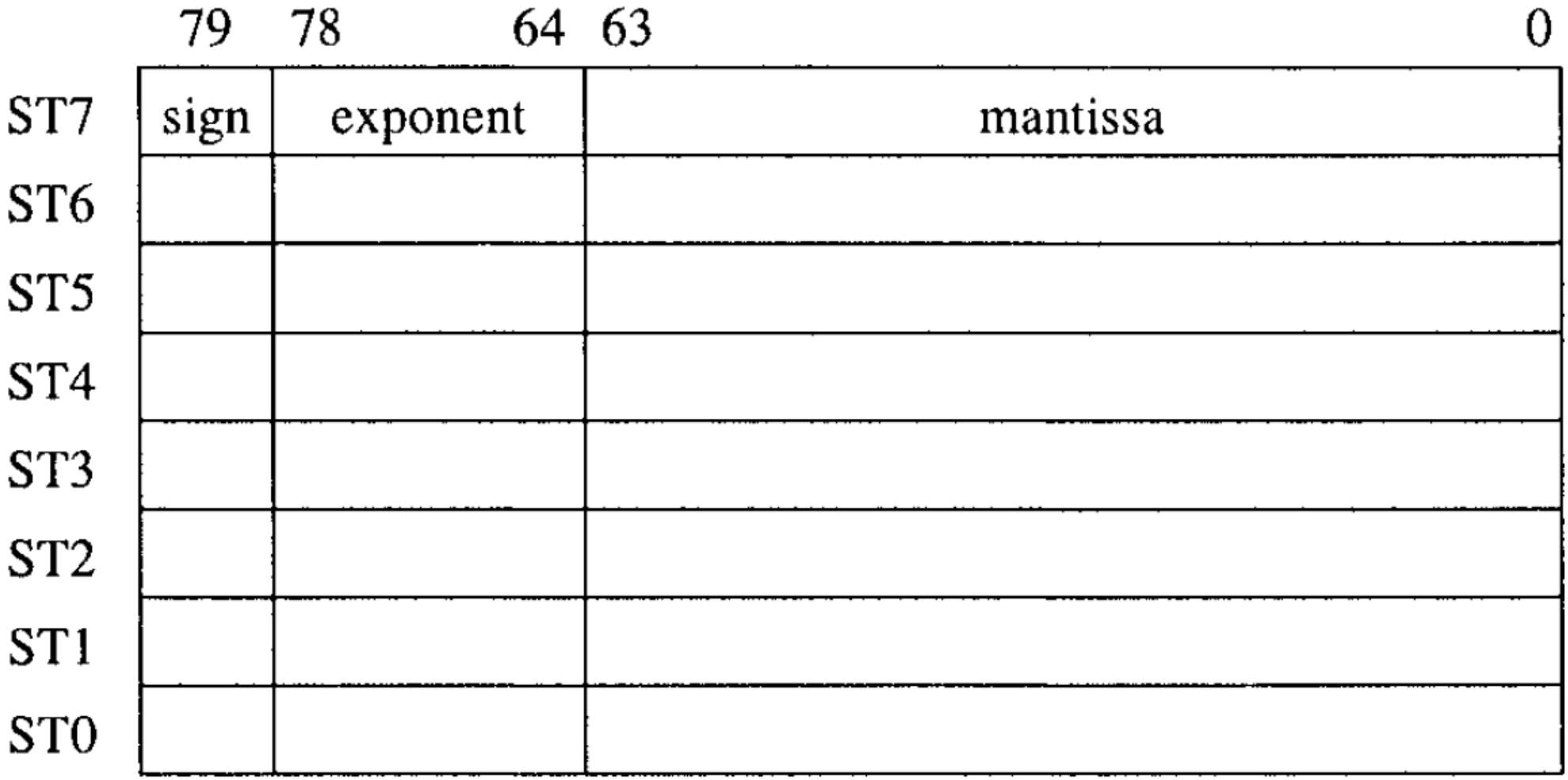
Instruções de ponto flutuante

- A FPU tem 8 registradores, st0 até st7, interligados em uma pilha. Números são empilhados da memória para a pilha, e são desempilhados da pilha para a memória.
- Endereçados individualmente (R0,...,R7)
- Um campo chamado TOP (de 3 bits) aponta para o topo da pilha

Operacao para empilhar reduz em 1 o valor de TOP

Operacao para desempilhar incrementa em 1 o valor de TOP

Instruções de ponto flutuante



FPU data registers

Flags

- Registradores de estado:

FPU flag	CPU flag
C0	CF
C2	PF
C3	ZF

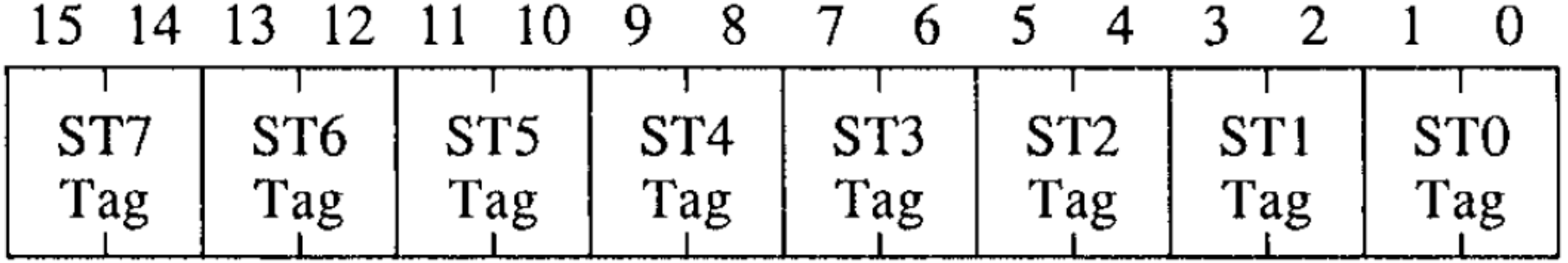
- O flag C1 é utilizado para indicar overflow

Flags

- Utiliza a instrução `fstsw` para armazenar os estados dos flags da FPU em AX.
- Utiliza `sahf` para carregar os valores nos registradores de flags da CPU

Instruções de ponto flutuante

- O FPU trabalha com os seguintes registradores de tag:



- Esses registradores armazenam status do conteúdo dos registradores de dados:

00 — valid
01 — zero
10 — special (invalid, infinity, or denormal)
11 — empty

Movimentação de dados

- A instrução **fld** empurra **src** para a pilha:

```
fld     src
```

- Decrementa o ponteiro do topo da pilha e armazena **src**
- O src pode estar na memória ou em um registrador.
- O **src** pode ser um número de simples precisão (32 bits) precisão dupla (64 bits) ou precisão extendida (80bits)

Movimentação de dados

- Instruções para empurrar constantes para a pilha:

Instruction	Description
`fldz`	Push +0.0 onto the stack
`fld1`	Push +1.0 onto the stack
`fldpi`	Push π onto the stack
`fldl2t`	Push $\log_2 10$ onto the stack
`fldl2e`	Push $\log_2 e$ onto the stack
`fldlg2`	Push $\log_{10} 2$ onto the stack
`fldln2`	Push $\log_e 2$ onto the stack

Movimentação de dados

- Para carregar um inteiro na pilha:

```
fild    src
```

- **src** deve ser um operando de 16 ou 32 bits localizado na memória.
- A instrução **fst** lê valor do topo da pilha e armazena em dest:

```
fst    dest
```

- **dest** pode ser um registrador ou memória
- **fst** lê o valor na pilha sem removê-lo

Movimentação de dados

- Para remover o valor da pilha (desempilhar) utiliza-se:

```
fstp    dest
```

- Lê o valor no topo da pilha e armazena o seu valor inteiro correspondente em **dest:**

```
fist    dest
```

Soma

- Soma o número em ponto flutuante armazenado na memória em **src** com o número armazenado em ST0

```
fadd      src      ;STO = STO + src
```

- A versão de dois operandos:

```
fadd      dest,src    ;dest = dest + src
```

- Na instrução *fadd* **dest** e **src** são registradores FPU.

Soma

- Versão para dar um pop na pilha:

```
faddp     dest,src
```

- Para adicionar inteiros:

```
fiadd     src     ;ST0 = ST0 + src
```

- O inteiro **src** tem que ser um inteiro de 16 ou 32 bits

Subtração

- Subtrai o número em ponto flutuante armazenado na memória em **src** com o número armazenado em ST0:

```
fsub    src
```

Ou seja:

```
ST0 = ST0−src
```

subtração

- A versão de subtração entre registradores FPU:

```
fsub    dest,src
```

- Isso executa a operação:

```
dest=dest-src
```

- Podemos também utilizar a versão pop:

```
fsubp    dest,src
```

Subtração

- Operação reversa:

```
fsubr     src
```

- Realiza a operação:

```
ST0 = src - ST0
```

- Possui uma versão pop:

```
fsubrp
```

Multiplicação

- **src** pode ser um número em ponto flutuante de 16 ou 32 bits armazenado na memória.

```
fmul     src
```

- A instrução ***fmul*** multiplica o valor armazenado em **src** com o valor armazenado em ST0 e armazena o resultado em ST0.

STO = STO * src

Multiplicação

- Podemos utilizar outra versão de multiplicação que trabalha com 2 registradores FPU:

```
fmul     dest,src
```

- Esta versão corresponde a $dest = dest * src$

multiplicação

- Multiplica o conteúdo de ST0 por um inteiro armazenado na memória:

```
fimul     src
```

- O valor de **src** pode ser de 32 ou 64 bits

Divisão

- Divide o conteúdo de ST0 por **src** e armazena o resultado em ST0:

```
fdiv    src
```

ou seja `ST0 = ST0/src`

- **src** é um número em ponto flutuante de precisão simples ou dupla.

Divisão

- Versão com dois operandos do **fdiv:**

```
fdiv    dest,src
```

ou seja:

```
dest = dest/src
```

Divisão

- A versão reversa de fdiv:

```
fdivr    src
```

Executa:

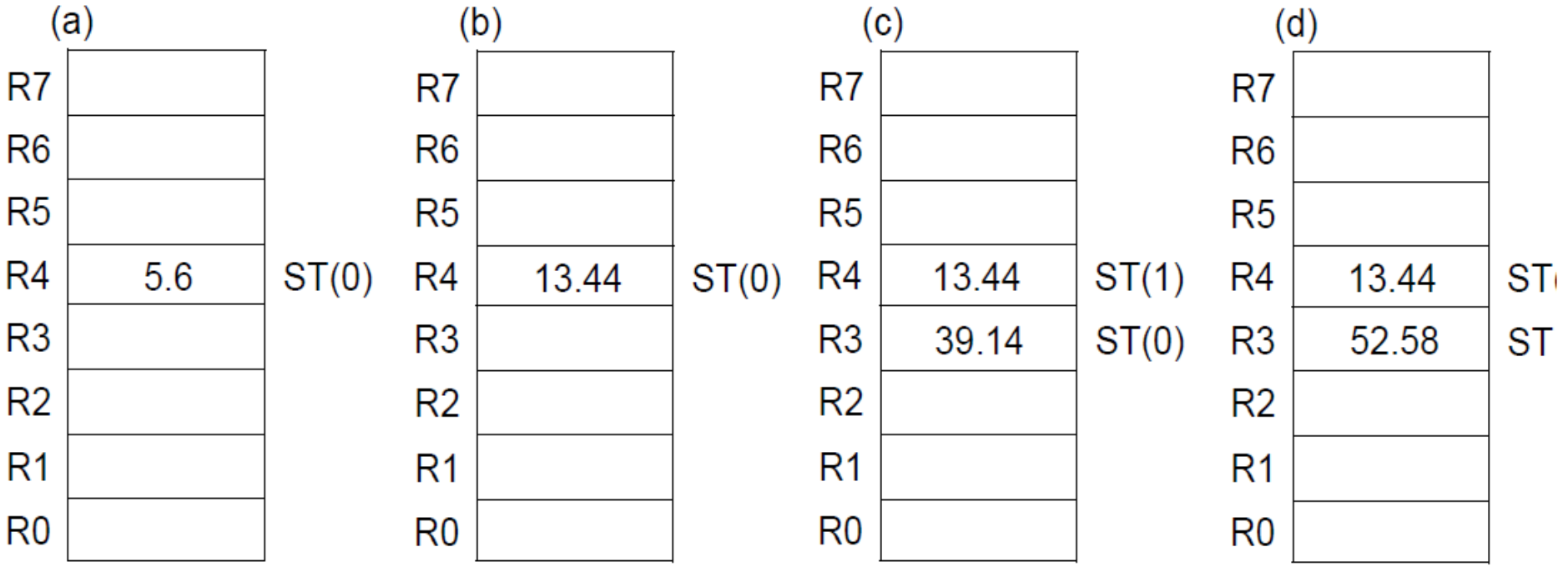
```
ST0 = src/ST0
```

Exemplo de Operação

Computation
Dot Product = (5.6 x 2.4) + (3.8 x 10.3)

Code:

```
FLD  value1 ;(a) value1=5.6
FMUL value2 ;(b) value2=2.4
FLD  value3 ; value3=3.8
FMUL value4 ;(c)value4=10.3
FADD ST(1)  ;(d)
```



Outros exemplos

- Calcular 2 * pi * r

FLDS 8(%EBP)

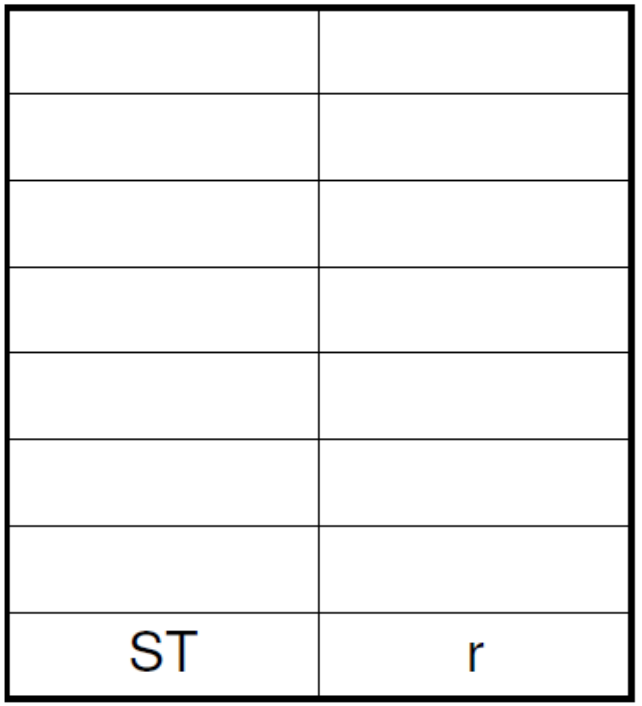
FLDPI

PUSHL $2

FILDL (%ESP)

FMULP %ST, %ST(1)

FMULP %ST, %ST(1)



Outros exemplos

- Calcular 2 * pi * r

FLDS 8(%EBP)

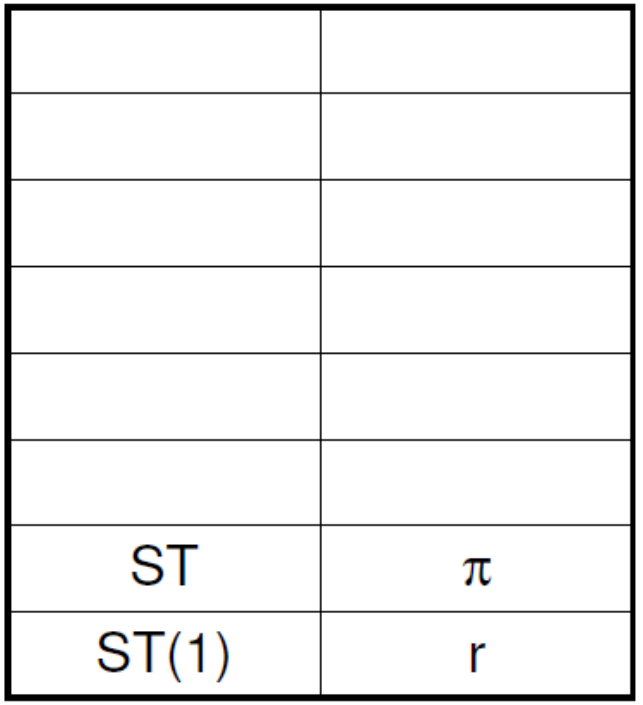
FLDPI

PUSHL $2

FILDL (%ESP)

FMULP %ST, %ST(1)

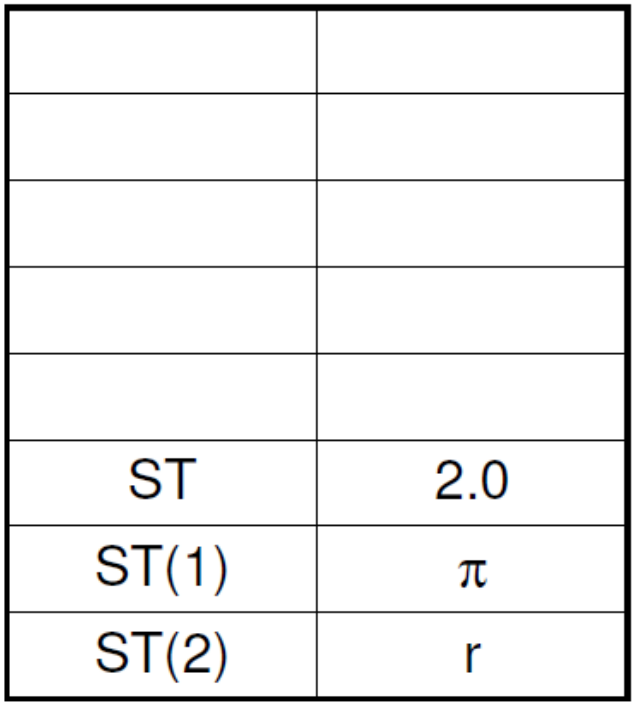
FMULP %ST, %ST(1)



Outros exemplos

- Calcular 2 * pi * r

```
FLDS 8(%EBP)

FLDPI

PUSHL $2

FILDL (%ESP)

FMULP %ST, %ST(1)

FMULP %ST, %ST(1)
```



Outros exemplos

- Calcular 2 * pi * r

FLDS 8(%EBP)

FLDPI

PUSHL $2

FILDL (%ESP)

FMULP %ST, %ST(1)

FMULP %ST, %ST(1)

ST	2.0 π
ST(1)	r

Outros exemplos

- Calcular 2 * pi * r

FLDS 8(%EBP)

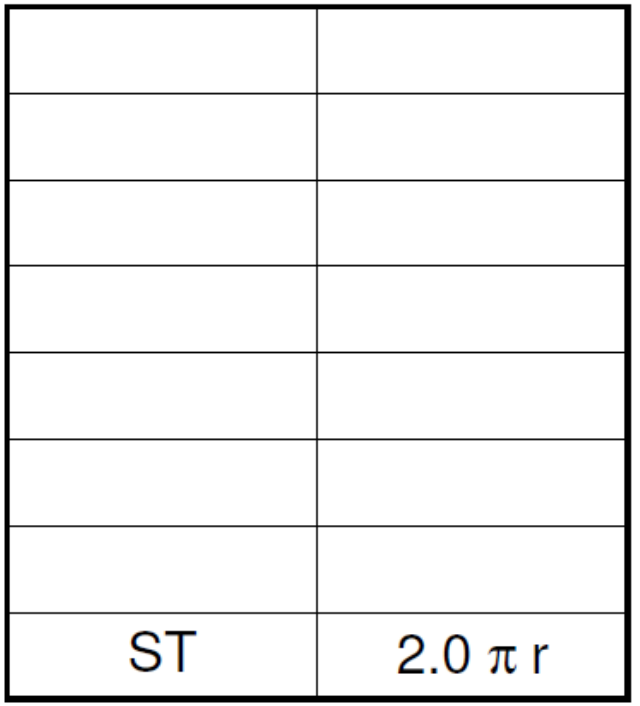
FLDPI

PUSHL $2

FILDL (%ESP)

FMULP %ST, %ST(1)

FMULP %ST, %ST(1)



Exemplo

- Calcula a raiz quadrada de um número:

```
global _start

section .data
    val: dq 123.45  ;declare quad word (double precision)

section .bss
    res: resq 1     ;reserve 1 quad word for result

section .text
    _start:

    fld qword [val] ;load value into st0
    fsqrt           ;compute square root of st0 and store in st0
    fst qword [res] ;store st0 in result

    ;end program
```

Exemplo

```
;; c^2 = a^2 + b^2 - cos(C)*2*a*b
;; C is stored in ang

global _start

section .data
    a: dq 4.56   ;length of side a
    b: dq 7.89   ;length of side b
    ang: dq 1.5  ;opposite angle to side c
                 ; (around 85.94 degrees)

section .bss
    c: resq 1    ;the result – length of side c

section .text
    _start:

    fld qword [a]   ;load a into st0
    fmul st0, st0   ;st0 = a * a = a^2

    fld qword [b]   ;load b into st1
    fmul st1, st1   ;st1 = b * b = b^2

    fadd st1, st0   ;st1 = a^2 + b^2

    fld qword [ang] ;load angle into st0
    fcos            ;st0 = cos(ang)
    fmul qword [a]  ;st0 = cos(ang) * a
    fmul qword [b]  ;st0 = cos(ang) * a * b
    fadd st0, st0   ;st0 = cos(ang) * a * b +
cos(ang) * a * b = 2(cos(ang) * a * b)

    fsubp st1, st0  ;st1 = st1 - st0 = (a^2 + b^2) - (2
* a * b * cos(ang))  ;and pop st0

    fsqrt           ;take square root of st0 = c

    fst qword [c]   ;store st0 in c – and we're done!

    ;end program
```

Comparação

- A instrução é utilizada para comparar dois números de ponto flutuante:

```
fcom    src
```

- Esta instrução compara o valor de ST0 com o valor de **src** e configura flags FPU.
- **src** pode estar na memória ou no registrador.

Comparação

- Na instrução **fcom** os flags são configurados da seguinte maneira:

		FPU flag	CPU flag
ST0 > src	C3 C2 C0 = 0 0 0	C0	CF
ST0 = src	C3 C2 C0 = 1 0 0	C2	PF
ST0 < src	C3 C2 C0 = 0 0 1	C3	ZF
Not comparable	C3 C2 C0 = 1 1 1		

Comparação

- Se nenhum operando for passado na instrução fcom os dois valores armazenados em STO e ST1 são comparados.
- Uma versão pop de fcom remove (desempilha) o valor armazenado em STO **fcomp *src.***
- Uma versão pop de fcom remove (desempilha) os valores armazenados em STO e ST1 da pilha:

```
fcompp
```

Comparação

- Para comparar o topo da pilha com um valor inteiro na memória:

```
ficom    src
```

- **src** pode ser um inteiro de 16 ou 32 bits
- Existe também a versão pop desta instrução:

```
ficomp
```

Comparação

- Existe uma instrução especial que trata da comparação com 0.0:

```
ftst
```

- Compara o topo da pilha com 0.0 e atualiza flags

Declarações comuns de variáveis

- Variáveis:

  dd – Precisão simples
  dq – Precisão dupla
  dt – Precisão extendida

  y dq 5.0

Outros exemplos

y = x^(1/2) **-raiz quadrada**

FILD word [x]
FSQRT
FSTP dword [y]

Outros exemplos

A = pi*(r^2) **-** área do círculo

FILD word [x]
FLD ST0
FMULP ST1, ST0
FLDPI
FMULP ST1, ST0
FSTP dword [y]

Exercícios

Crie uma função que leia três posições de memória (lados de um triangulo) e determine os ângulos desse triângulo.

Crie uma função que receba três argumentos (inteiros, ‘a’, ‘b’ e ‘c’) e atualize duas posições de memória com as raízes da equação quadrática correspondente.

Instruções x87

- Disponível em:

  http://www2.math.uni-wuppertal.de/~fpf/Uebungen/GdR-SS02/opcode_f.html