Linguagem de Montagem

Ponto Flutuante
Aula 12

Edmar André Bellorini

Ano letivo 2016

Introdução - x86 (32 bits)

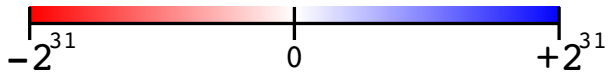
- Ponto-Flutuante (PF)
  - *Floating-Point* (FP)
  - É o formato binário de representação dos números reais
  - Baseia-se na notação científica dos números $\pm M * B^{\pm e}$, onde:
    M: Mantissa, também chamado de *significando*
    B: Base numérica
    e: Expoente

Representatividade

- Inteiro em Complemento de Dois
  - 32 bits

$-2.147.486.648$ até $+2.147.486.648$

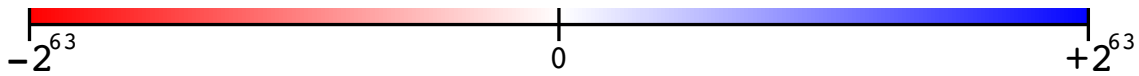
$-2,147 * 10^9$ até $+2,147 * 10^9$



  - 64 bits

$-9.223.372.036.854.780.000$ até $+9.223.372.036.854.780.000$

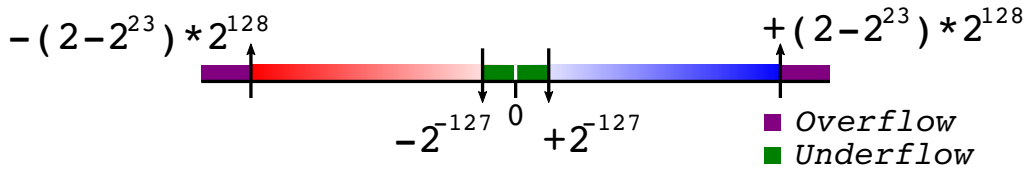
$-9,223 * 10^{18}$ até $+9,223 * 10^{18}$



Representatividade

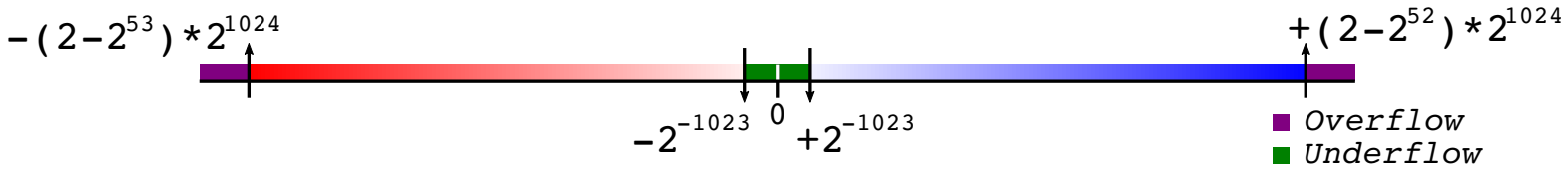
- IEEE 754
  - Precisão Simples (32 bits)

$-2,854 * 10^{45}$ até $-5,877 * 10^{-39}$, *zero*, $+5,877 * 10^{-39}$ até $+2,854 * 10^{45}$



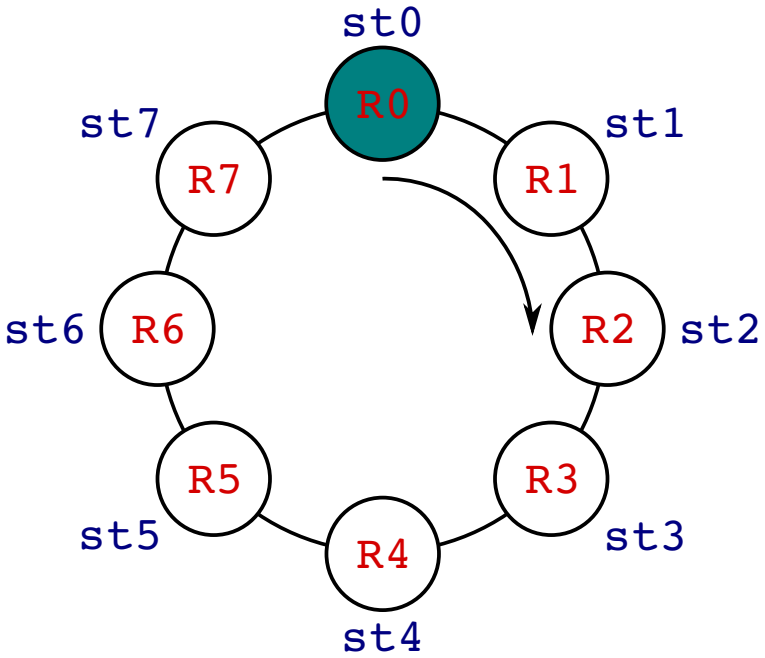
  - Precisão Dupla (64 bits)

$-8,096 * 10^{323}$ até $-1,112 * 10^{-308}$, *zero*, $+1,112 * 10^{-308}$ até $+8,096 * 10^{323}$



Arquitetura FPU

- Registradores F.P.
  - Funcionam como uma pilha (*floating-point stack*)
  - 32 (*dd*), 64 (*dq*) ou 80 (*dt*) bits cada um
  - Nomenclatura "st_i" indica posição fixa
    - *st*0 aponta sempre para TOPO
  - Nomenclatura "R_x" indica posição rotacional (gdb)



Declaração

- Declaração de variáveis inicializadas

```
section .data
    fp1 : dd 1.2      ; precisao simples
    fp2 : dd 1.3e10
    fp3 : dq 1.4e+10  ; precisao dupla
    fp4 : dq 1.6e-10
    fp5 : dq -1.7
    fp6 : dt 1.8      ; precisao dupla extendida
```

  - somente com *dd* (*resd*), *dq* (*resq*) ou *dt* (*rest*)
  - formato: $[\pm][9]^{+}.[9]^{*}[e^{\pm}[9]^{+}]$

Movimentação de dados - LOAD

- Carregar memória para FPU
  - Rotaciona registradores FPU (sentido **horário**)
  - Carrega m32/m64/m80 para st0

```
fld m32/m64/m80
```

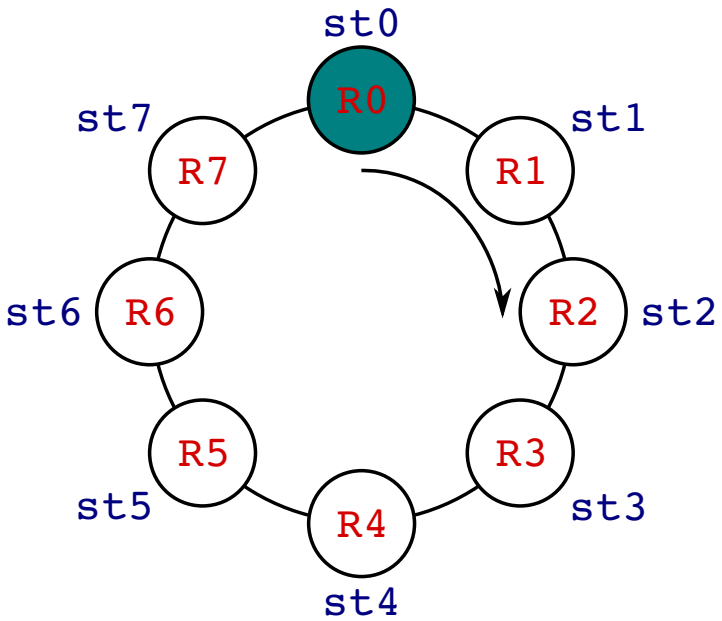
- Carregar st[1-7] para st0
  - Rotaciona registradores FPU (sentido horário)
  - Carrega m32/m64 para st0

```
fld st[1-7]
```

- Importante
  - *st*0 deve estar livre (*Empty*)
  - caso exista valor alocado (*Valid*), *fld* vai corromper *st*0

Exemplo a12e01.asm - movimentação de dados

- Considerações para o passo-a-passo



Legenda:

st_i ponteiros para Regs FPU

R_i Reg. livre

fp_x R_i Reg. carregado com fp_x

Reg. TOPO da pilha (st0)

fp_x R_i Reg. carregado com fp_x e TOPO

a12e01 - pt00

- Dados inicializados:

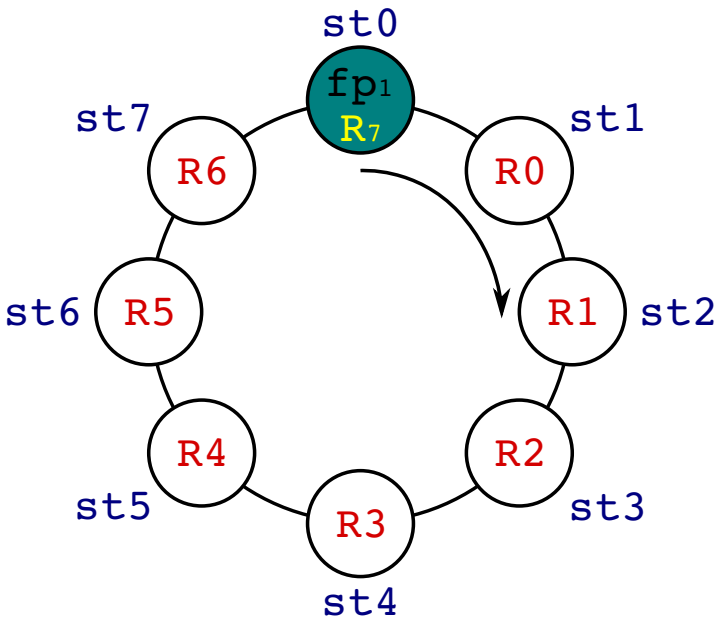
```
1       section .data
2         fp1 : dd 1.1      ; precisao simples
3         fp2 : dd 1.2e3
4         fp3 : dq 1.3e3    ; precisao dupla
5         fp4 : dq 1.4e-3
6         fp5 : dq -1.5e3
7         fp6 : dt 1.6      ; precisao dupla ext.
```

- Montar e Linkar:

```
$: nasm -f elf64 a12e01.asm
$: gcc -m64 a12e01.o -o a12e01.x
```

a12e01 - pt01

```
19      fld dword [fp1]
```



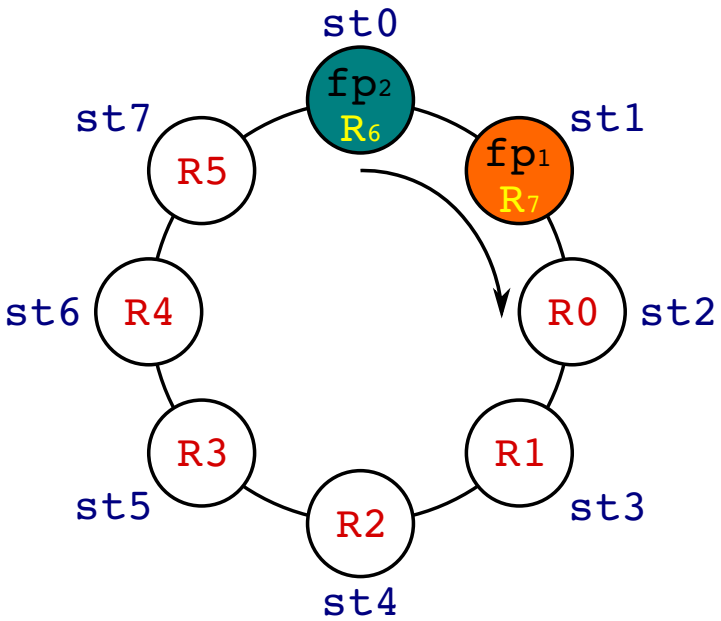
a12e01 - pt01 - gdb

- gdb:

```
(gdb) b f1
Ponto de parada 1 at 0x4004f7
(gdb) r
Breakpoint 1, 0x00000000004004f7 in f1 ()
(gdb) i float
=>R7: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R6: Empty   0x00000000000000000000
  R5: Empty   0x00000000000000000000
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000
  (gdb)
```

a12e01 - pt02

```
24      fld dword [fp2]
```



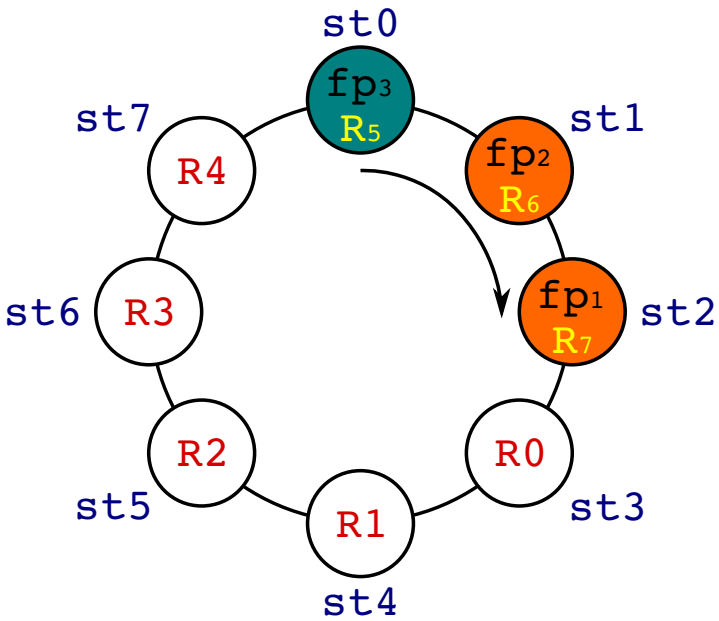
a12e01 - pt02 - gdb

- gdb:

```
(gdb) b f2
Ponto de parada 2 at 0x4004fe
(gdb) c
Breakpoint 2, 0x00000000004004fe in f2 ()
(gdb) i float
  R7: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
=>R6: Valid   0x40099600000000000000 +1200
  R5: Empty   0x00000000000000000000
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000
  (gdb)
```

a12e01 - pt03

```
29      fld qword [fp3]
```



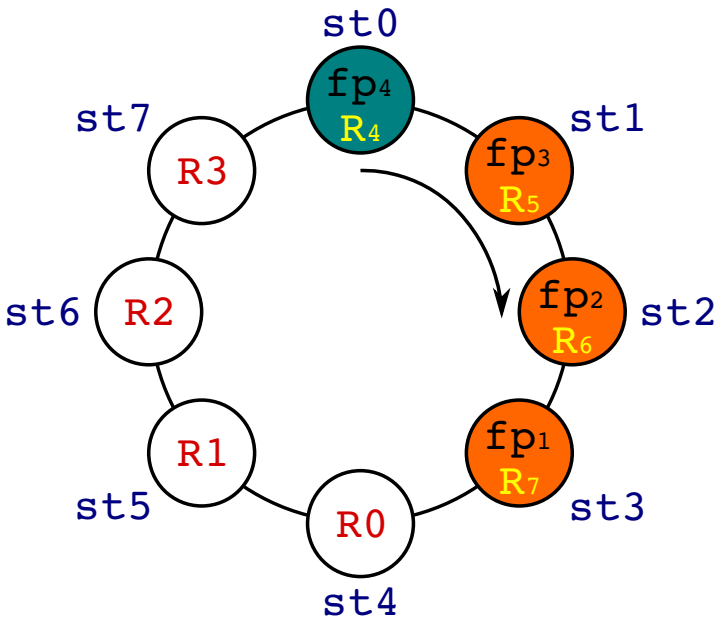
a12e01 - pt03 - gdb

- gdb:

```
 (gdb) b f3
 Ponto de parada 3 at 0x400505
 (gdb) c
Breakpoint 3, 0x0000000000400505 in f3 ()
 (gdb) i float
  R7: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R6: Valid   0x40099600000000000000 +1200
=>R5: Valid   0x4009a28000000000000 +1300
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000
  (gdb)
```

a12e01 - pt04

```
34      fld qword [fp4]
```



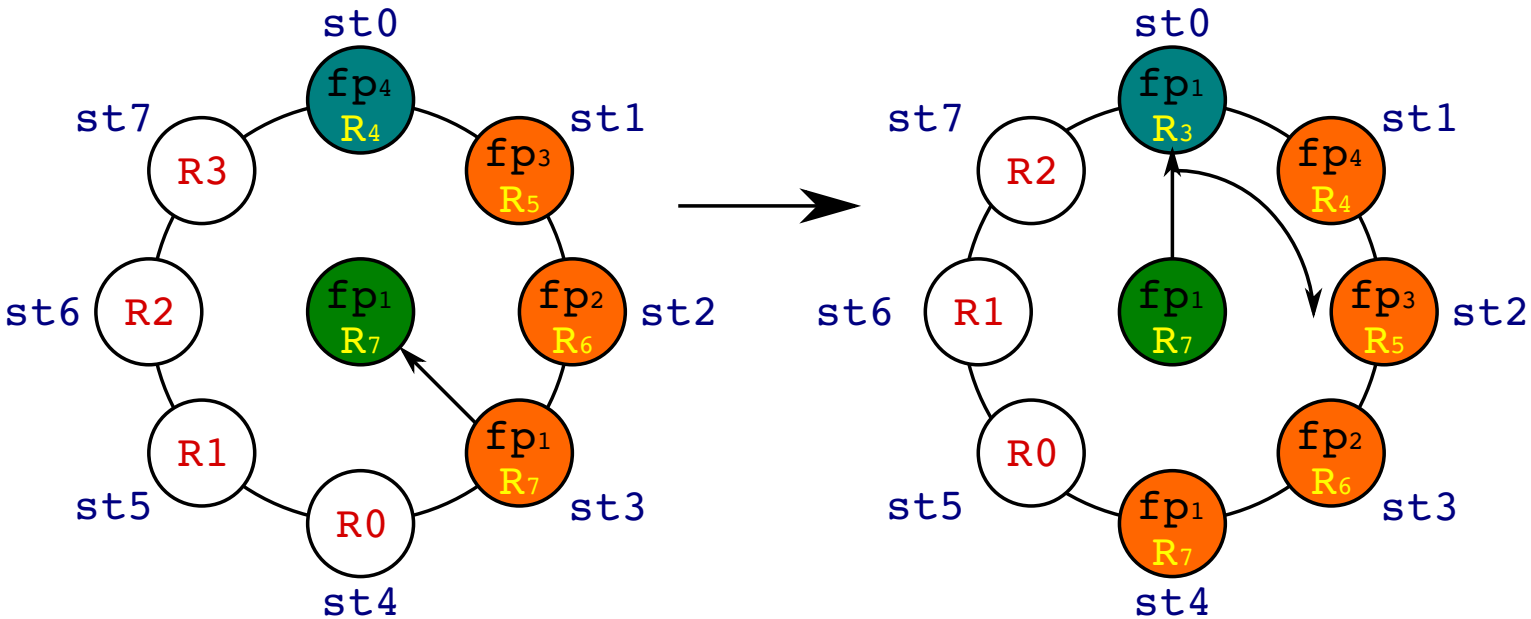
a12e01 - pt04 - gdb

- gdb:

```
 (gdb) b f4
 Ponto de parada 4 at 0x40050c
 (gdb) c
Breakpoint 4, 0x000000000040050c in f4 ()
 (gdb) i float
  R7: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R6: Valid   0x40099600000000000000 +1200
  R5: Valid   0x4009a28000000000000 +1300
=>R4: Valid   0x3ff5b780346dc5d63800 +0,00139999999999999986
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000
  (gdb)
```

a12e01 - pt05

```
39      fld st3
```



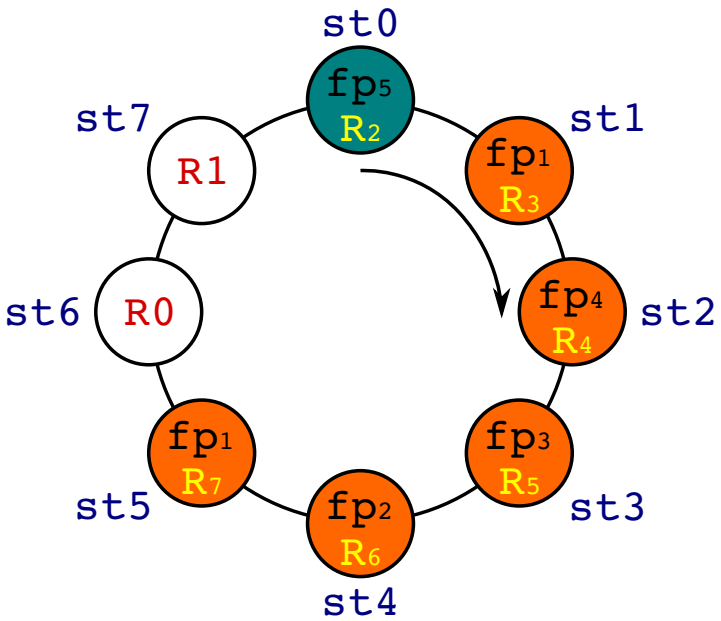
a12e01 - pt05 - gdb

- gdb:

```
 (gdb) b f5
 Ponto de parada 5 at 0x40050e
 (gdb) c
Breakpoint 5, 0x000000000040050e in f5 ()
 (gdb) i float
  R7: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R6: Valid   0x4009960000000000000 +1200
  R5: Valid   0x4009a280000000000000 +1300
  R4: Valid   0x3ff5b780346dc5d63800 +0,0013999999999999999986
=>R3: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000
  (gdb)
```

a12e01 - pt06

```
44      fld qword [fp5]
```



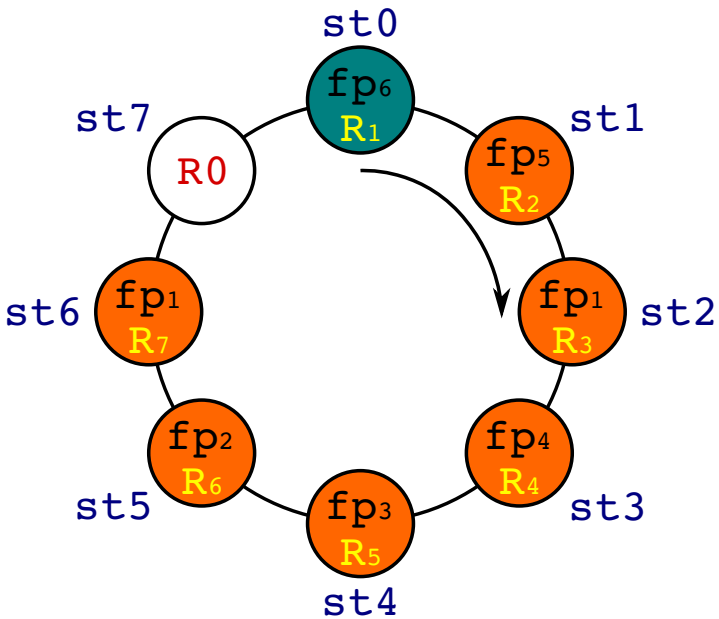
a12e01 - pt06 - gdb

- gdb:

```
 (gdb) b f6
 Ponto de parada 6 at 0x400515
 (gdb) c
Breakpoint 6, 0x0000000000400515 in f6 ()
 (gdb) i float
  R7: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R6: Valid   0x4009960000000000000 +1200
  R5: Valid   0x4009a28000000000000 +1300
  R4: Valid   0x3ff5b780346dc5d63800 +0,0013999999999999999986
  R3: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
=>R2: Valid   0xc009bb8000000000000 -1500
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000
  (gdb)
```

a12e01 - pt07

```
49      fld tword [fp6]
```



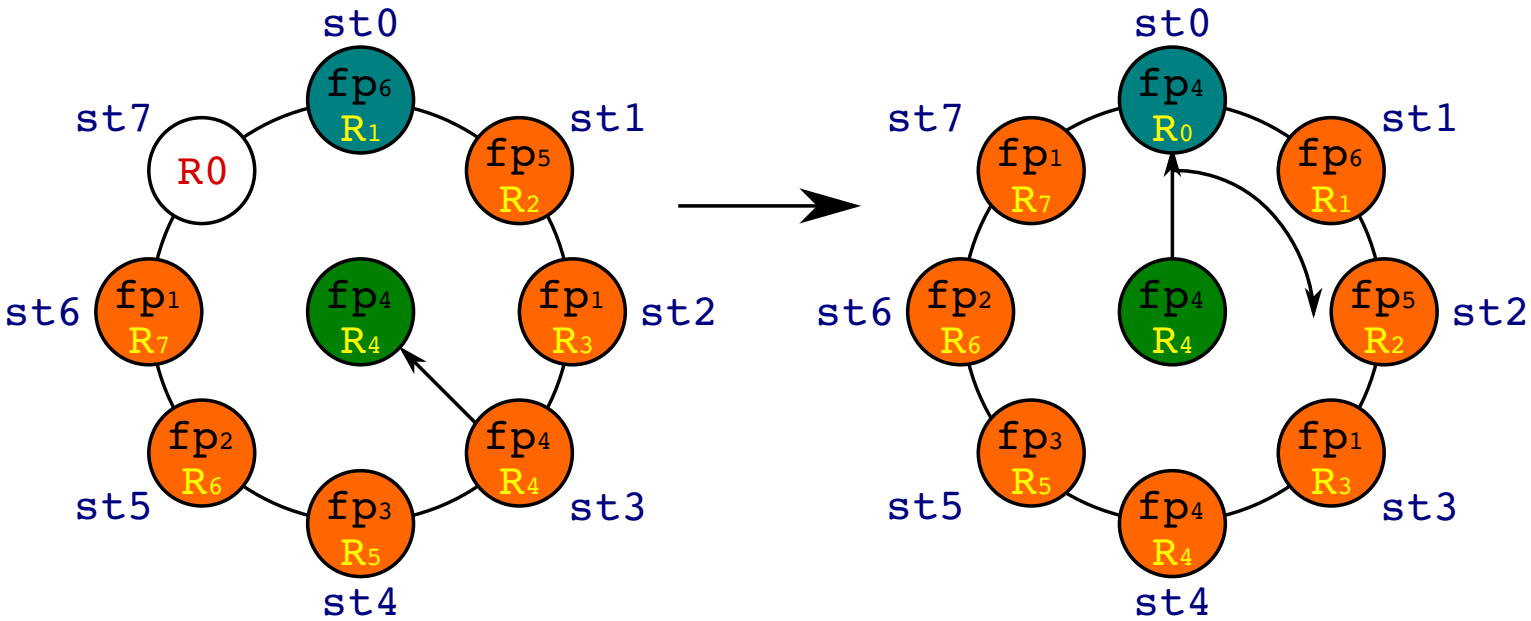
a12e01 - pt07 - gdb

- gdb:

```
 (gdb) b f7
 Ponto de parada 7 at 0x40051c
 (gdb) c
Breakpoint 7, 0x000000000040051c in f7 ()
 (gdb) i float
  R7: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R6: Valid   0x4009960000000000000 +1200
  R5: Valid   0x4009a28000000000000 +1300
  R4: Valid   0x3ff5b780346dc5d63800 +0,0013999999999999999986
  R3: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R2: Valid   0xc009bb8000000000000 -1500
=>R1: Valid   0x3fffccccccccccccccd +1,6
  R0: Empty   0x00000000000000000000
  (gdb)
```

a12e01 - pt08

```
54      fld st3
```



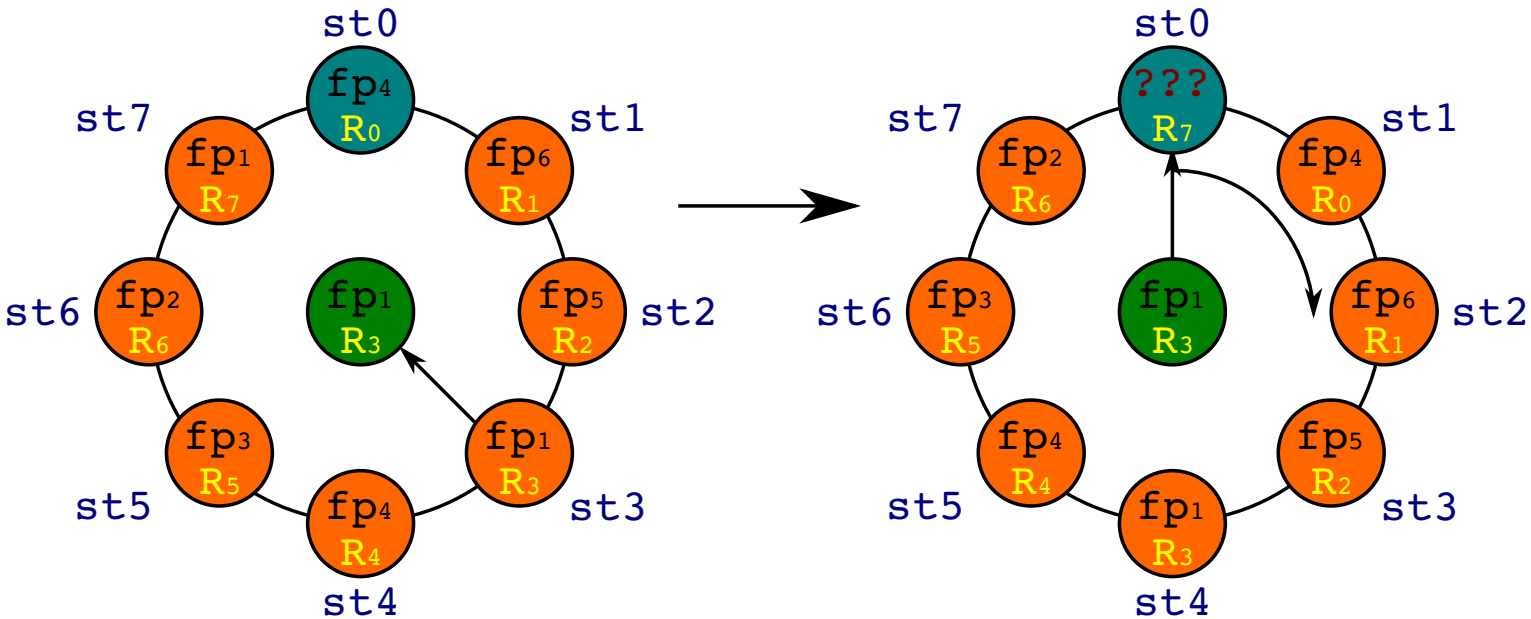
a12e01 - pt08 - gdb

- gdb:

```
 (gdb) b f8
 Ponto de parada 8 at 0x40051e
 (gdb) c
Breakpoint 8, 0x000000000040051e in f8 ()
 (gdb) i float
  R7: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R6: Valid   0x40099600000000000000 +1200
  R5: Valid   0x4009a28000000000000 +1300
  R4: Valid   0x3ff5b780346dc5d63800 +0,0013999999999999999986
  R3: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R2: Valid   0xc009bb8000000000000 -1500
  R1: Valid   0x3fffcccccccccccccccd +1,6
=>R0: Valid   0x3ff5b780346dc5d63800 +0,0013999999999999999986
  (gdb)
```

a12e01 - pt09

```
61      fld st3
```



a12e01 - pt09 - gdb

- gdb:

```
 (gdb) b fim
 Ponto de parada 9 at 0x400520
 (gdb) c
Breakpoint 9, 0x0000000000400520 in f9 ()
 (gdb) i float
=>R7: Special 0xffffc000000000000000 Real Indefinite (QNaN)
  R6: Valid   0x40099600000000000000 +1200
  R5: Valid   0x4009a28000000000000000 +1300
  R4: Valid   0x3ff5b780346dc5d63800 +0,0013999999999999999986
  R3: Valid   0x3fff8ccccd0000000000 +1,10000002384185791
  R2: Valid   0xc009bb8000000000000000 -1500
  R1: Valid   0x3fffcccccccccccccccd +1,6
  R0: Valid   0x3ff5b780346dc5d63800 +0,0013999999999999999986
  (gdb)
```

Movimentação de Dados

- Armazenar st0 na memória
  - armazena st0 na memória m32/m64/m80

```
fst m32/m64/m80
```

- Armazenar st0 na memória e rotacionar registradores FPU
  - armazena st0 na memória m32/m64/m80
  - rotaciona registradores FPU (sentido **anti-horário**)

```
fstp m32/m64/m80
```

- Armazenar st0 em registrador FPU (*Empty*)
  - sem rotação

```
fst st[1-7]
```

  - com rotação

```
fstp st[1-7]
```

Exemplo a12e02.asm - Movimentação de Dados - STORE

- Cuidado:
  - Ao armazenar um Reg. FPU na memória, deve-se levar em consideração o tamanho do espaço em memória **e** o conteúdo do registrador
- Apresentação de *float*

```
(gdb) p/f nomeVar
```

- Apresentação de *double*

```
(gdb) x/1fg &nomeVar
```

- Apresentação de *double extendido*

```
(gdb) x/10xb &nomeVar
```

  - Será mostrado os 10 bytes, em HEX, da palavra de 80 bits

Aritmética para F.P. - ADIÇÃO - pt01

- Desalocar Reg. FPU (torná-lo *Empty*)

```
FFREE sti
```

- operando implícito: *st*0

```
FADD [TO] st[1-7]/mem32/mem64/mem80
```

  - Operação: $st0 = st0 + fonte$
  - TO: inverte alvo/fonte
    $fonte = st0 + fonte$

```
FADDP st[1-7]
```

  - semelhante ao FADD TO, porém rotaciona Regs. FPU
  - **alvo** deve ser outro Reg. FPU

Aritmética para F.P. - SUBTRAÇÃO - pt02

- operando implícito: *st*0

```
FSUB [TO] st[1-7]/mem32/mem64/mem80
```

  - Operação: *st*0 = *st*0 - *fonte*
  - TO: inverte alvo/fonte
    fonte = *st*0 - *fonte*

```
FSUBP st[1-7]
```

  - semelhante ao FSUB TO, porém rotaciona Regs. FPU
  - **alvo** deve ser outro Reg. FPU

```
FSUBR[P][TO] st[1-7]
```

  - subtração reversa: *st*0 = *fonte* - *st*0

Aritmética para F.P. - MULTIPLICAÇÃO - pt03

- operando implícito: *st*0

```
FMUL [TO] st[1-7]/mem32/mem64/mem80
```

  - Operação: *st*0 = *st*0 * *fonte*
  - TO: inverte alvo/fonte
    fonte = *st*0 * *fonte*

```
FMULP st[1-7]
```

  - semelhante ao FMUL TO, porém rotaciona Regs. FPU
  - **alvo** deve ser outro Reg. FPU

Aritmética para F.P. - DIVISÃO - pt03

- operando implícito: *st*0

```
FDIV [TO] st[1-7]/mem32/mem64/mem80
```

  - Operação: *st*0 = *st*0 / *fonte*
  - TO: inverte alvo/fonte
    fonte = *st*0 / *fonte*

```
FDIVP st[1-7]
```

  - semelhante ao FDIV TO, porém rotaciona Regs. FPU
  - **alvo** deve ser outro Reg. FPU

Aritmética para F.P. - OUTRAS - pt03

- FABS (valor absoluto)
- FCHS (negação de PF)
- FDECSTP (Rotaciona Regs. FPU - anti-horário)
- FINCSTP (Rotaciona Regs. FPU - horário)
- FLD1 (carrega **1** em st0)
- FLDPI (carrega **PI** em st0)
- FSQRT (calcula raiz quadrada de st0)
- FXCH (realiza a troca de valores entre st0 e st[1-7])

- Leitura Recomentada:

▸ CMSC 313 – NASM Floating Point Documentation

▸ SIMPLY FPU

Exemplo a12e03.asm

- Arquitetura x86_64 (64 bits)
  - se código for x86 (32 bits) cuidado com *doubles* vs pilha 32 bits
- Cálculo executado: $(2,0 + 1,5 * 0,03)/0,00321$
  - Usa-se Regs. FPU
- Usa-se *printf*
  - Número de *doubles* usados é alocado em **RAX**
  - *doubles* são alocados em registradores SSEs (xmm_{1-7})
    - Movimentação de dados: **movlps** xmm_i, m64
    - Será estudado nas próximas aulas (instruções MMX)
  - printf e o problema do alinhamento
    - a pilha do programa deve estar alinhada em 16 bytes alinhamento é realizado ajustando pilha em 8 bytes:

```
main:
  sub rsp, 8
  ...
  add rsp, 8
fim:
  ret
```

Exercícios de Fixação

- EF1201: Crie uma função que retorne a raiz positiva ou negativa de uma função quadrática
  - forma da função quadrática: $ax^2 + bx + c = 0$
  - Bhaskara: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
  - funcionamento:
    - solicitar entrada para constantes a, b e c
      usar tipo de dado *double*
    - retornar
      $x1 = solucaoX'$
      $x2 = solucaoX''$
  - chamada da função:

```
double bhaskara(double a, double b double c, char r)
```

onde: r é "n" ou "p" (raiz x' ou raiz x'')

Relatório

- Somente relatório
  - O modelo de relatório para a disciplina de LM está disponível em anexo da Aula 01
    - Arquivo **modeloRelatorioLM.odt**
    - A data do relatório é a data de entrega (ver moodle)
  - Somente serão aceitos os relatórios em formato .pdf com nome do arquivo seguindo o padrão:
    TY.PXX.nome.sobrenome.pdf
    - TY é o número da turma prática (1, 2, 3 ou 4)
    - PXX é o número da prática, neste caso: P12
  - Ex.: aluno Bhaskara Akaria da turma prática 9 (de 1114):
    - T9.P12.Bhaskara.Akaria.pdf