Introdução à Software Básico: Operações Ponto Flutuante IA-32

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade de Brasília

Sumário

Puzzels

- Para cada expressão seguinte em C, verifique se é ou não verdadeira:
  - x == (int)(float) x
  - x == (int)(double) x
  - f == (float)(double) f
  - d == (float) d
  - f == -(-f);
  - 2/3 == 2/3.0
  - d $<$ 0.0 $\Rightarrow$ ((d*2) $<$ 0.0)
  - d $>$ f $\Rightarrow$ -f $<$ -d
  - d * d $\geq$ 0.0
  - (d+f)-d == f
- Sendo que as declarações inicias foram:
  - int x = ...;
  - float f = ..;
  - double d = ...;

Padrão IEEE

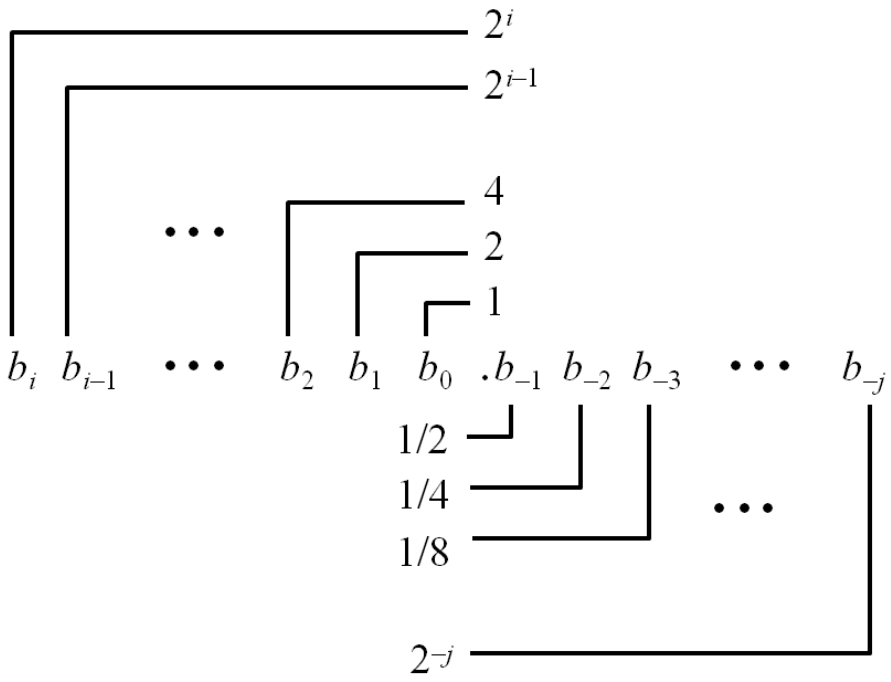
Padrão IEEE 754

- Estabelecido em 1985 como padrão para aritmética de ponto flutuante (antes dele vários formatos diferentes)
- Compatível com a maioria de CPUs

Objetivos do Padrão

- Poder realizar operações aritméticas, com facilidade no tratamento de arredondamento, overflow e underflow.
- Não foi pensado com foco para implementação em hardware, logo é difícil de implementar e otimizar.

Números Binários Fracionarios



Representação

- Bits a direita do "ponto binario" representa frações em potência de 2.
- Representação do número decimal: $\sum_{k=-j}^{i}(b_k \times 2^k)$

Números Binários Fracionarios

Exemplos

- $5\frac{3}{4} : 101.11_2$
- $2\frac{7}{8} : 10.111_2$
- $\frac{63}{64} : 0.111111_2$

Observação

- Dividir por 2 = fazer shift para direita
- Multiplicar por 2 = fazer shift para esquerda

Limitação

- Somente podem ser representados número da forma $X/2^k$
- Outros números tem representação com expansão repetidas de bits
- $\frac{1}{3} : 0.0101010101[01]_2$
- $\frac{1}{5} : 0.001100110011[0011]_2$
- $\frac{1}{10} : 0.0001100110011[0011]_2$

Números Binários Fracionarios

Numerical Form

- $-1^S \times \text{Mantissa} \times 2^{\text{Expoente}}$
  - S - 1 bit de sinal que determina se o numero é negativo ou positivo
  - Mantissa também é chamada de *significand*.
  - Expoente:
    - no padrão IEEE 754 o expoente pode ser base 2 ou 10 ($-1^S \times M \times 10^E$).
    - Nos sistemas computacionais normalmente é utilizado a base 2.

Tamanhos

- E = codificação do expoente, M = codificação da Mantissa
- *Single precision*: E - 8 bits, M - 23 bits (32 total)
- *Double precision*: E - 11 bits, M - 52 bits (64 total)
- Em IA-32 existe a extended precision.

Números Binários Normalizados

Condição

- E $\neq$ 000...00 e E $\neq$ 111...11

Expoente é codificado com um BIAS

- Bias: Modificador do expoente
  - *single precision*: 127 (E: 1...254,Expoente= $-126...127$)
  - *double precision*: 1023 (E: 1...2046,Expoente= $-1022...1023$)
  - BIAS $= 2^{m-1} - 1$, onde m é a quantidade de bits de E

Mantissa é codificada com "1" como precedente

- Mantissa $= 1.XXXXXXXXX_2$
  - somente são codificados os bits decimais, o "1" inicial é simplesmente assumido que existe. ($M = XXXXX_2$)
  - se todos os bits são zeros, então Mantissa $= 1.0$

Números Binários Normalizados

Exemplo

- Float F $= 15213.0$;
  - $15213_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$
- Mantissa é 1.1101101101101_2, logo é codificada como $M = 11011011011010000000000_2$
- Expoente é 13, porém como o BIAS é 127, então $E = 140 = 10001100_2$.

Floating Point Representation (Class 02):

Hex:	4	6	6	D	B	4	0	0
Binary:	0100	0110	0110	1101	1011	0100	0000	0000
140:	100	0110	0					
15213:			*1*110	1101	1011	01		

Números Binários Não-Normalizados

Condição

- E = 000...00

Codificação

- Mantissa é $0.XXXXXXXX_2$ (O valor "0" inicial não é codificado)
- O expoente é interpretado como -BIAS+1

Casos

- $E = 0000...00$, $M = 000...00$
  - Representa o valor 0.
- $E = 0000...00$, $M \neq 000...00$
  - Números próximos de 0.0
  - A precisão vai diminuindo assim que os números ficam menores
  - "Gradual Underflow"

Números Binários Especiais

Condição

- E = 111...11

Casos

- $E = 1111...11$, $M = 000...00$
  - Representa o valor infinito ou divisão por zero.
  - Indica que aconteceu overflow
  - Tanto para o valor positivo como negativo
- $E = 1111...11$, $M \neq 000...00$
  - Not-a-Number (NaN)
  - Representa um valor que não pode ter representação númerica
  - Por exemplo: sqrt(-1)

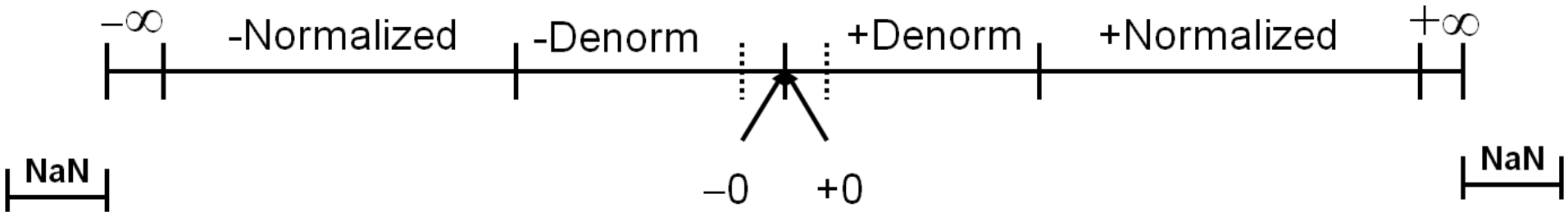
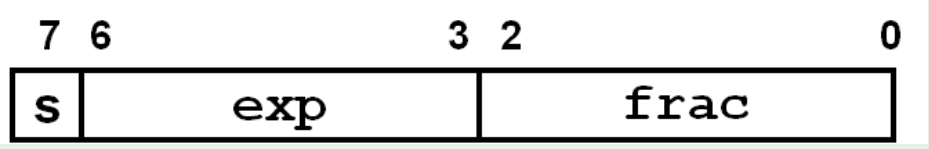


Figura: Resumo

Exemplo com Representação de 8 bits

- Mesma forma geral que o formato IEEE
- Mesmas regras para valores normalizados, não-normalizados, NaN e infinito.



Exemplo com Representação de 8 bits

- Valores relacionados ao expoente

Exp	exp	E	2^E	
0	0000	-6	1/64	(denorms)
1	0001	-6	1/64	
2	0010	-5	1/32	
3	0011	-4	1/16	
4	0100	-3	1/8	
5	0101	-2	1/4	
6	0110	-1	1/2	
7	0111	0	1	
8	1000	+1	2	
9	1001	+2	4	
10	1010	+3	8	
11	1011	+4	16	
12	1100	+5	32	
13	1101	+6	64	
14	1110	+7	128	
15	1111	n/a		(inf, NaN)

Exemplo com Representação de 8 bits

- Faixa Dinâmica

	s	exp	frac	E	Value	
Denormalized numbers	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	← closest to zero
	0	0000	010	-6	2/8*1/64 = 2/512	
	...					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	← largest denorm
	0	0001	000	-6	8/8*1/64 = 8/512	← smallest norm
	0	0001	001	-6	9/8*1/64 = 9/512	
	...					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	← closest to 1 below
Normalized numbers	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	← closest to 1 above
	0	0111	010	0	10/8*1 = 10/8	
	...					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	← largest norm
	0	1111	000	n/a	inf	

Números Interesantes

- Valores relacionados ao expoente

Description	exp	frac	Numeric Value
Zero	00…00	00…00	0.0
Smallest Pos. Denorm. • Single ≈ 1.4×10^{-45} • Double ≈ 4.9×10^{-324}	00…00	00…01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
Largest Denormalized • Single ≈ 1.18×10^{-38} • Double ≈ 2.2×10^{-308}	00…00	11…11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
Smallest Pos. Normalized • Just larger than largest denormalized	00…01	00…00	$1.0 \times 2^{-\{126,1022\}}$
One	01…11	00…00	1.0
Largest Normalized • Single ≈ 3.4×10^{38} • Double ≈ 1.8×10^{308}	11…10	11…11	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$

- ϵ é o limite superior do erro de arrendondamento em aritmética de ponto flutuantes.
- single precision: $\epsilon = 2^{-23}$
- double precision: $\epsilon = 2^{-52}$

Propriedades Importantes

- O número 0 em representação de ponto flutuante é igual ao numero 0 em representação de inteiros (todos os bits zero)
- A comparação entre dois números ponto flutuante pode ser feita utilizando comparação inteira sem sinal, considerando o seguinte:
  - primeiro comparar o bit de sinal
  - considerar $-0 = 0$
  - NaN é um problema, é maior que os outros números? qual o resultado da compração?
  - Todos os outros casos a comparação inteira sem sinal vai funcionar (normalizado vs não-normalizado, normalizado vs infinito)

Modelo Conceitual

- Primeiro calcular o valor exato
- Fazer que o valor encaixe na precissão desejada
  - Pode dar overflow se o expoente é muito grande
  - Pode ser necessário fazer arrendondamento para que a mantissa possa ser expressada com o número de bits desejado.
- Modos de arredondamento:

	$1.40	**$1.60**	**$1.50**	**$2.50**	**–$1.50**
• **Zero**	$1.00	$1.00	$1.00	$2.00	–$1.00
• **Round down (-∞)**	$1.00	$1.00	$1.00	$2.00	–$2.00
• **Round up (+∞)**	$2.00	$2.00	$2.00	$3.00	–$1.00
• **Nearest Even (default)**	$1.00	$2.00	$2.00	$2.00	–$2.00

- "Round down" é conhecido como FLOOR, e o "round up" como CEIL.

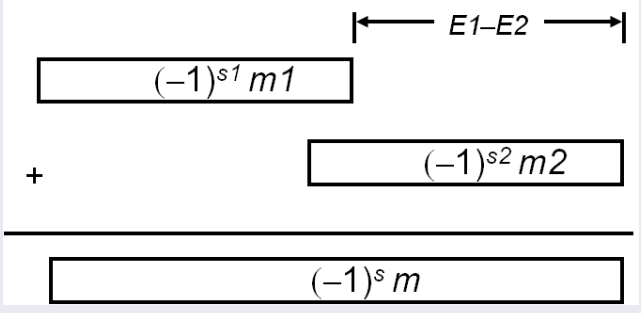
Multiplicação

- Operandos:
  - $(-1)^{s1} \times M1 \times 2^{E1}$
  - $(-1)^{s2} \times M2 \times 2^{E2}$
- Valor Exato:
  - $(-1)^{s} \times M \times 2^{E}$
  - $s = s1$ XOR $s2$
  - Mantissa é $M1 \times M2$
  - Expoente é $E1 + E2$
- Colocando na precisão desejada:
  - Se $M \geq 2$, shift M um bit para direita e incrementa E
  - Se E fica precisa mais bits do que os reservados então overflow
  - Arredondar M para que fique entre os bits reservados

Operações com Ponto Flutuante

Adição

- Operandos:
  - $(-1)^{s1} \times M1 \times 2^{E1}$
  - $(-1)^{s2} \times M2 \times 2^{E2}$
- Valor Exato:
  - $(-1)^{s} \times M \times 2^{E}$
  - s e M é resultado do somo alinhada
  - $E = E1$



- Colocando na precisão desejada:
  - Se $M \geq 2$, shift M um bit para direita e incrementa E
  - Se $M < 1$, shift M para esquerda k posições, decrementa E por k.
  - Se E fica precisa mais bits do que os reservados então overflow
  - Arredondar M para que fique entre os bits reservados

Propriedades da Adição

- Commutativa: SIM
- Associativa: Não, devido ao overflow e arredondamento
- 0 é a identidade aditiva: SIM
- Todo elemento tem um inverso aditivo: SIM, com exeção do infinito e NaN
- $a \geq b \Rightarrow a + c \geq b + c$: SIM, com exeção de infinito e NaN

Propriedades da Multiplicação

- Commutativa: SIM
- Associativa: Não, devido ao overflow e arredondamento
- 1 é a identidade aditiva: SIM
- A multiplicação é distributiva sobre a adição: NÃO, devido ao overflow e arredondamento
- $a \geq b \ \& \ c \geq 0 \Rightarrow a \times c \geq b \times c$: SIM, com exeção de infinito e NaN

Precisão

- FLOAT e DOUBLE

Conversão

- *Casting* entre inteiro, single e double muda os valores númericos
  - Ponto flutuante para inteiro: trunca a parte fracionaria
  - inteiro para double: Conversão exata a mantissa tem 53 bits maior que o inteiro.
  - inteiro para float: Vai arredondar para caber em 23 bits.

Puzzels

- Para cada expressão seguinte em C, verifique se é ou não verdadeira:
  - x == (int)(float) x Não: 24 bit significand
  - x == (int)(double) x SIM: 53 bit significand
  - f == (float)(double) f SIM: Aumento precisão
  - d == (float) d Não: Perdeu Precisão
  - f == -(-f); SIM: sinal mudou duas vezes
  - 2/3 == 2/3.0 Não: 2/3 == 0
  - d $<$ 0.0 $\Rightarrow$ ((d*2) $<$ 0.0) Sim
  - d $>$ f $\Rightarrow$ -f $<$ -d Sim
  - d * d $\geq$ 0.0 Sim
  - (d+f)-d == f Não: Não associativo
- Sendo que as declarações inicias foram:
  - int x = ...;
  - float f = ..;
  - double d = ...;

Ponto Flutuante em IA-32

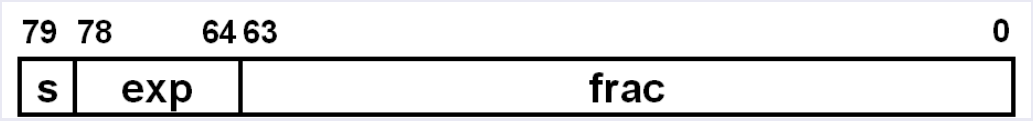
Precisão

- FLOAT - 32 bits
- DOUBLE - 64 bits
- Extended - 80 bits

Operações

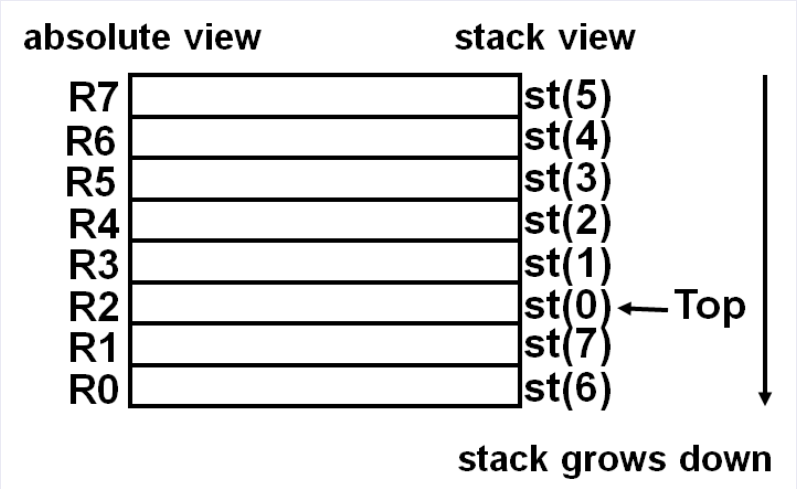
- Hardware específico para somar, multiplicar e dividir ponto flutuante
- Registradores específicos para ponto flutuante

Registrador FPU



FPU stack

- A pilha cresce de R7 para R0
- Os registros são referênciados com relação ao inicio da pilha: st(0) é o topo, seguido de st(1), st(2), etc.



Instruções

- Grande variedade de instruções de ponto flutuante
  - aprox. 50 instruções
  - load, store, add e mult
  - sin, cos, tan, arctan, and log!

```
Instruction     Effect                          Description

fldz            push 0.0                        Load zero
flds S          push S                          Load single precision real
fmuls S         st(0) <- st(0)*S                Multiply
faddp           st(1) <- st(0)+st(1); pop       Add and pop
```

Figura: exemplo de instruções de FPU

Exemplo

- Calcular o produto interno de dois vetores

```
float ipf (float x[],
           float y[],
           int n)
{
  int i;
  float result = 0.0;

  for (i = 0; i < n; i++) {
    result += x[i] * y[i];
  }
  return result;
}
```

```
   pushl %ebp              # setup
   movl %esp,%ebp
   pushl %ebx

   movl 8(%ebp),%ebx       # %ebx=&x
   movl 12(%ebp),%ecx      # %ecx=&y
   movl 16(%ebp),%edx      # %edx=n
   fldz                    # push +0.0
   xorl %eax,%eax          # i=0
   cmpl %edx,%eax          # if i>=n done
   jge .L3
.L5:
   flds (%ebx,%eax,4)      # push x[i]
   fmuls (%ecx,%eax,4)     # st(0)*=y[i]
   faddp                   # st(1)+=st(0); pop
   incl %eax               # i++
   cmpl %edx,%eax          # if i<n repeat
   jl .L5
.L3:
   movl -4(%ebp),%ebx      # finish
   leave
   ret                     # st(0) = result
```

Figura: exemplo de instruções de FPU

Exemplo

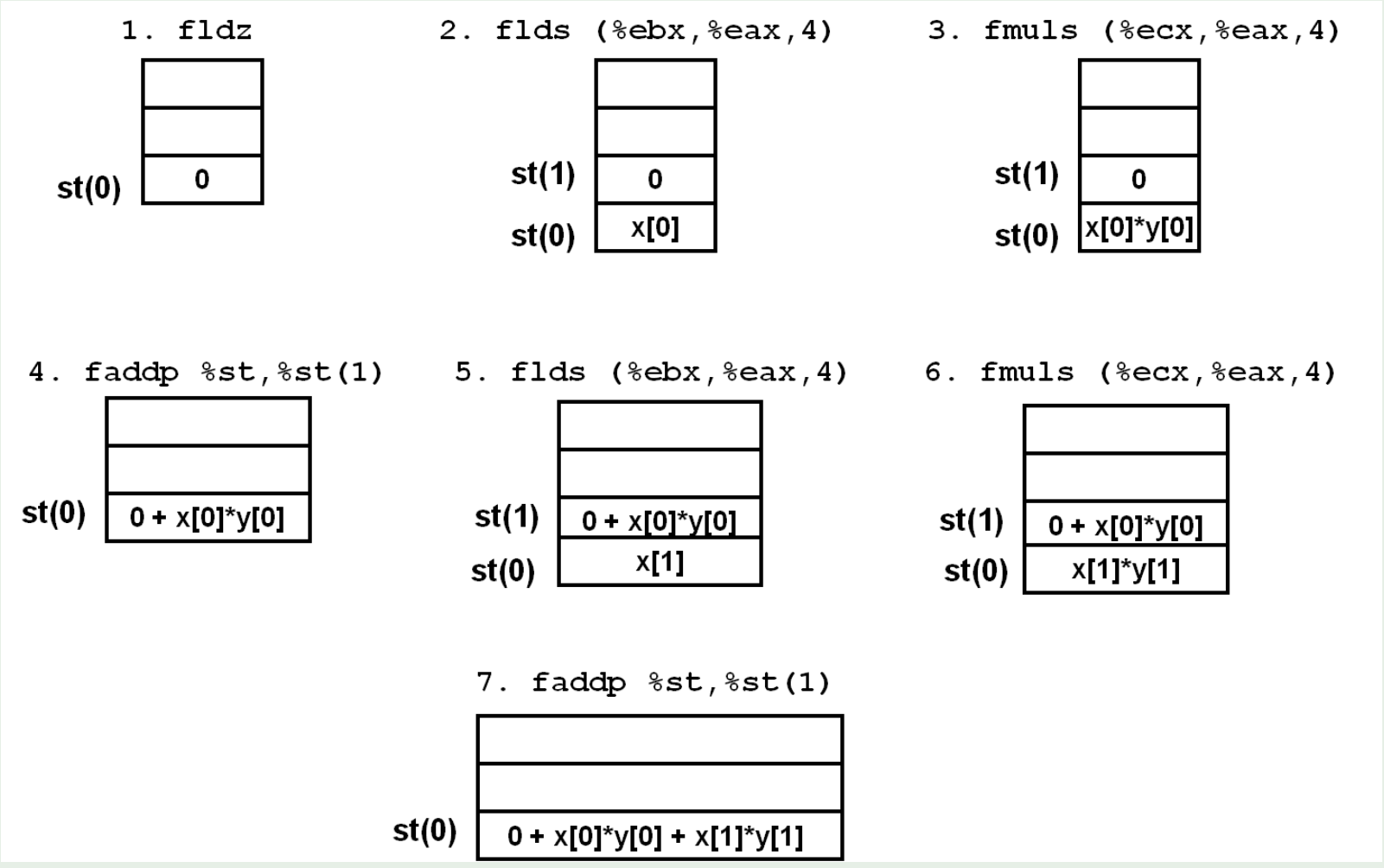


Figura: exemplo de instruções de FPU

Próxima Aula

Introdução a Assembly x64