11. Aritmética de vírgula flutuante

Vamos falar de aritmética não-inteira (i.e., fraccionária, ou de "vírgula flutuante"). Este capítulo pode ser adiado pois precisas de conceitos teóricos importantes sobre representação de números de vírgula flutuante. Precisas de saber o que é o "significando" (também conhecido como coeficiente ou *mantissa)* e o "expoente" e a sua representação normalizada em binário (formato IEEE 754).

Vamos resolver um novo programa: o `a11-float.asm`. Monta-o e corre-o. O que faz?

Operações em números fraccionários

Todas as operações aritméticas mencionadas até aqui trabalham sobre números inteiros. Para trabalhar com números fraccionários vamos precisar de usar um circuito completamente diferente do CPU.

Os circuitos do CPU que trabalham com números de vírgula flutuante eram opcionais e comprados à parte até à família 80386 – o co-processador matemático 80387. Também por esta razão ainda se dá de forma muito notória um nome especial a estes circuitos: a FPU *(floating-point unit* ou unidade de vírgula flutuante).

As instruções da FPU são novas e trabalham com **oito novos registos** (`st0`, `st1`, ... até `st7`) que guardam cada um número de vírgula flutuante de 80 bits e funcionam como uma pilha. Por razões de eficiência esta pilha é circular, mas é acedida relativamente ao seu "topo" actual. A notação `st0` (ou `ST(0)`) significa "*stack top* + 0".

As instruções da FPU que fazem operações aritméticas lêem preferencialmente os seus argumentos da pilha. Então, para fazer um cálculo, tenho de colocar os valores do mesmo na pilha (com instruções específicas), e só depois executo a instrução aritmética em si. Esta instrução retira da pilha os seus argumentos e substitui-os pelo resultado da operação em si.

Por exemplo, para fazer a conta

$$(5{,}6 \times 2{,}4) + (3{,}8 \times 10{,}3)$$

tenho de fazer os seguintes passos:

1. Colocar 5,6 na pilha
2. Colocar 2,4 na pilha
3. Executar o produto (na pilha saem os dois valores e fica o resultado)
4. Colocar 3,8 na pilha
5. Colocar 10,3 na pilha
6. Executar o produto (na pilha saem os dois valores e fica o resultado)
   (não esquecer que o resultado do produto anterior ainda lá está)
7. Executar a soma

Por exemplo, depois de executar o passo 5, a pilha terá o seguinte aspecto:

`st2`	13,44 (= 5,6 × 2,4)
`st1`	3,8
`st0`	10,3

Para simplificar estes passos, as instruções permitem geralmente que o último argumento faça parte da instrução em si.

Assim, a sequência final de passos será:

1. Colocar 5,6 na pilha
2. Multiplicar por 2,4 (na pilha sai o 5,6 e fica o resultado)
3. Colocar 3,8 na pilha
4. Multiplicar por 10,3 (na pilha sai o 3,8 e fica o resultado)
   (não esquecer que o resultado do produto anterior ainda lá está)
5. Somar o topo da pilha+1 ao topo da pilha

Nota que todos os números de vírgula flutuante usados devem estar em memória e eu devo referir-me a eles pelo seu endereço.

O código pode então ser o seguinte:

```
section .data

valor1      dq  5.6
valor2      dq  2.4
valor3      dq  3.8
valor4      dq 10.3
```

```
section .text

            finit
            fld    qword[valor1]  ; Passo 1
            fmul   qword[valor2]  ; Passo 2
            fld    qword[valor3]  ; Passo 3
            fmul   qword[valor4]  ; Passo 4
            faddp                 ; Passo 5
```

Todas as instruções de vírgula flutuante começam com a letra `f` que é a abreviatura de *"floating-point"*.

Imagina agora que o resultado desta conta (52,58) é a quantidade de dinheiro que gastei após um levantamento de 100 Euros. Quero saber quanto dinheiro me sobra. Preciso de fazer:

100 - 52,58

Repara que o 52,58 já está na pilha (é o `st0`). Posso subtrair 100, mas isso faz a conta ao contrário e dá-me um resultado simétrico* ao que quero. Devo então usar uma operação comum que é a troca do topo da pilha com outra posição qualquer da pilha, antes de fazer a subtracção.

Os passos serão então (após os anteriores):

6. Colocar 100 na pilha
7. Trocar o topo da pilha pelo topo da pilha+1
8. Subtrair do topo da pilha+1, o topo da pilha

O código que falta será o seguinte:

```
section .data

; ...
valor5      dq 100.0

section .text

            ; ...
            fld    qword[valor5]  ; Passo 6
            fxch                  ; Passo 7
            fsubp                 ; Passo 8
```

As instruções novas são as seguintes:

Operações de vírgula flutuante	
`finit`	Inicializa a FPU. Limpa a pilha e as *flags* de excepções, e estabelece que situações de excepção (ver abaixo) não devem interromper o código, arredondamento ao mais próximo e precisão de 64 bits.

* De sinal oposto.

`fld m/s`	Carregar constante de vírgula flutuante *(floating-point load)*. Faz o equivalente a um `push`, mas para a pilha de vírgula flutuante, de uma constante guardada em `m` ou de um registo `s`.
`fstp m/s`	Guardar constante de vírgula flutuante *(floating-point store and pop)*. Faz o equivalente a um `pop`, mas da pilha de vírgula flutuante para uma variável `m` ou para outro registo `s`.
`fxch s`	Troca o topo da pilha (`st0`) e `s`. Se `s` for `st1`, posso omiti-lo.
`fadd m/s`	Soma o topo da pilha (`st0`) com `m/s`. Substitui o topo da pilha pelo resultado.
`fsub m/s`	Subtrai do topo da pilha (`st0`), `m/s`. Substitui o topo da pilha pelo resultado.
`fmul m/s`	Multiplica o topo da pilha (`st0`) por `m/s`. Substitui o topo da pilha pelo resultado.
`fdiv m/s`	Divide o topo da pilha (`st0`) por `m/s`. Substitui o topo da pilha pelo resultado.
`faddp`	Soma o topo da pilha+1 (`st1`) com o topo da pilha (`st0`). Guarda o resultado em `st1` e faz o equivalente a um `pop` (ou seja, substitui `st0` e `st1` pelo resultado).
`fsubp`	Subtrai do topo da pilha+1 (`st1`), o topo da pilha (`st0`). Guarda o resultado em `st1` e faz o equivalente a um `pop` (ou seja, substitui `st0` e `st1` pelo resultado).
`fmulp`	Multiplica o topo da pilha+1 (`st1`) pelo o topo da pilha (`st0`). Guarda o resultado em `st1` e faz o equivalente a um `pop` (ou seja, substitui `st0` e `st1` pelo resultado).
`fdivp`	Divide o topo da pilha+1 (`st1`) pelo o topo da pilha (`st0`). Guarda o resultado em `st1` e faz o equivalente a um `pop` (ou seja, substitui `st0` e `st1` pelo resultado).

Em qualquer uma destas instruções, `m` pode ser uma referência à memória *(quad-word* – número de vírgula flutuante de precisão dupla – ou *double-word* – de precisão simples – e.g.: `qword[numero]`) e `s` pode ser uma referência à pilha de vírgula flutuante (e.g.: `st0`). `m/s` pode ser qualquer uma das duas referências.

Existem inúmeras mais instruções disponíveis, para fazer todo o tipo de operações que encontrarias numa calculadora científica (senos, logaritmos, raízes, arredondamentos, etc.), assim como formas diferentes de usar estas mesmas instruções. Consulta o manual de referência [IA32-MAN] para mais detalhes (ver Bibliografia).

Comparações e tomada de decisões

Para comparar dois números de vírgula flutuante uso instruções especializadas:

Comparações de vírgula flutuante	

`fcom st,s`	Compara o topo da pilha (`st0` ou `st`) com o registo `s`.
	Para poder agora usar as habituais instruções de salto condicional, tenho antes de fazer:
	`fstsw ax`
	`sahf`
	Isto copia as *flags* de códigos de condição da FPU para o registo `eflags` normal do CPU.

`fcomi st,s`	Compara o topo da pilha (`st0` ou `st`) com o registo `s`. Não são necessárias mais instruções para poder fazer o salto condicional imediatamente a seguir, mas esta instrução está apenas disponível nos processadores da família P6 ou acima.

Nota que dados erros de arredondamento e dada a imprecisão com que certos números decimais são representados, dois resultados de duas operações diferentes que matematicamente seriam iguais, podem não o ser em vírgula flutuante.

Neste caso, a condição resultado1=resultado2 daria (inesperadamente) falso.

Sempre que possível devemos comparar a igualdade de dois números dentro de uma gama de erro aceitável para a aplicação em questão, ou seja, em vez de:

$$\text{resultado1} = \text{resultado2}$$

devemos fazer:

$$\text{valor\_absoluto}(\text{resultado1} - \text{resultado2}) < \text{erro}$$

Detecção de situações de excepção *(overflow*, etc.)

Mesmo lidando com uma gama numérica expandida, os números de vírgula flutuante têm limites e podem ocorrer situações de transbordo.

Essas e outras condições de excepção são:

- *Overflow* (transbordo): quando o resultado é demasiado grande (i.e., tende para $\pm\infty$), o expoente não pode ser aumentado levando a uma situação de *overflow*.
- *Underflow:* quando o resultado é demasiado pequeno (i.e., tende para zero), o expoente não pode ser reduzido levando a uma situação de *underflow*. Antes do *underflow* ocorrer, o número entra num estado "denormalizado" para perder gradualmente precisão conforme se aproxima ainda mais de zero.
- Indeterminações: certas operações (como 0/0 ou ∞-∞) não têm um resultado, sendo matematicamente definidas como "indeterminadas".

Note-se que os infinitos têm uma representação em IEEE 754.

Em todas estas situações, o resultado que é devolvido pela FPU é um padrão especial do IEEE 754 que representa NaN *("Not a Number")*. Qualquer operação em que um dos argumentos seja um NaN tem como resultado outro NaN. Assim, para verificar se algum cálculo intermédio de uma expressão teve uma excepção, só tenho de verificar se o resultado final é um NaN. Um NaN* é codificado com todos os bits do expoente a 1 e pelo menos um bit do significando com um valor $\neq 0$. O bit de sinal pode ter qualquer valor.

Outra forma de detectar excepções é olhar para os bits da *status word* da FPU que indicam presença de excepções:

```
            fclex           ; limpa os bits de excepcoes
                            ; (finit tambem faz isso)
            ; ... (operacoes)

            fstsw  ax
            and    ax, 0x003F
            jnz    houve_excepcao
```

Existem algumas outras excepções que não iremos estudar *(stack overflow/underflow*, argumento inválido, argumento "denormal", resultado inexacto, etc.).

Consulta o manual de referência [IA32-MAN] para mais detalhes (ver Bibliografia).

Exemplo final: Calculadora

Monta e corre o exemplo final: `a11-calculadora-float.asm`. O que faz?

Esta calculadora recebe números em decimal, notação habitual, e exibe um resultado em decimal com duas casas depois da vírgula.

Para fazer a conversão *string* para vírgula flutuante e vice-versa, esta calculadora usa instruções de que não falámos aqui. Deves no entanto já ter alguma flexibilidade de consultar os manuais de referência para a compreenderes sozinho. Tenta.

Exercício

Pega na calculadora de 64 bits que construíste (ou na original, de 32 bits, se não resolveste o exercício). Assume que os números que a pessoa introduz são constantes de vírgula flutuante no seu formato IEEE 754. Assume também que o número hexadecimal que a calculadora vai exibir, é o resultado da operação no mesmo formato IEEE 754.

Altera então as quatro operações da calculadora para fazerem esses cálculos em vírgula flutuante.

---

* QNaN ou SNaN: não vamos detalhar as diferenças.