Insper

Elementos de Sistemas

Controle de Programa

"Se tudo parece sobre controle, você não está indo rápido o bastante."
"If everything seems under control, you're just not going fast enough."
Mario Andretti (1940), Piloto de corridas italiano
apud Nisan, N. & Schocken, S. 2005. Elements of Computing Systems

Ciência da Computação - Luciano Soares <lpsoares@insper.edu.br>

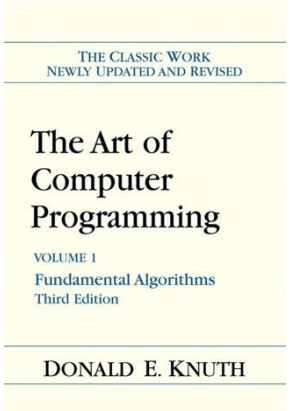
Donald Knuth

Donald Knuth é chamado de "pai da análise de algoritmos”, com várias contribuições na teoria de computação.



TEX





Insper

Arquiteturas de Máquinas Virtuais

Baseada em Pilha



Java Virtual Machine (JVM)



.Net Common Language Runtime (CLR)

Baseada em Registradores



Android Runtime (ART)



Lua Virtual Machine

Compilação JIT x AOT

- **JIT (Just in Time)**, compilação para linguagem de máquina é feita durante execução.

  AOT (Ahead of Time), o código é transformado em código de máquina durante a instalação.

Motivação

$$x = (-b + \sqrt{b^2 - 4 \cdot a \cdot c}) / 2a$$

```
if ~(a == 0)
   x = (-b + sqrt(power(b,2) - 4 * a * c)) / (2 * a)
else
   x = - c / b
```

Para traduzir esse código de alto nível para código de máquina virtual à pilha, temos de lidar com:

- Operações aritméticas
- Operações booleanas
- Fluxo de programa
- Subrotinas

Subrotinas, Funções, Procedimentos

Subrotinas, Funções, Procedimentos ou ainda Métodos em linguagens orientadas a objetos são mais um mecanismo de abstração para encapsular operações simples em uma mais poderosa.

São trechos de programas que podem ser invocados pelos programas, que executam algo e retornam no ponto que foram chamados quando finalizados.

Linguagens de programação procedurais modernas, permitem o programador criar novas subrotinas livremente, sendo usadas da mesma forma que as instruções nativas da linguagem.

Razão das sub-rotinas

- Criar sub-rotinas como sqrt e power conforme necessário;
- Usar as sub-rotinas como se fossem operações elementares.
    - Umas chamando outras;
    - Uma chamando a si mesma (recursão).

Funcionamento de Subrotinas

Para cada chamada de sub-rotina, os seguintes procedimentos devem ser feitos:

- Passar os parâmetros para a sub-rotina chamada;
- Salvar o estado da rotina que está chamando;
- Alocar espaço das variáveis locais da sub-rotina chamada;
- Saltar para executar o código da sub-rotina chamada;
- Retornar os resultados da sub-rotina chamada;
- Reciclar o espaço de memória ocupado pela sub-rotina;
- Recuperar o estado da sub-rotina chamadora;
- Continuar código da sub-rotina chamadora.

Chamada de sub rotinas

call <nome da sub-rotina>

Uma rotina coloca os argumentos a passar na pilha e chama a sub-rotina e espera para retornar na linha seguinte.

Da mesma forma que um comando nativo (built-in), as chamadas de sub-rotinas podem remover ou colocar elementos do topo da pilha.

```
// x+2
push x
push 2
add
...
```

```
// x^3
push x
push 3
call power
...
```

```
// (x^3+2)^y
push x
push 3
call power
push 2
add
push y
call power
...
```

```
// Power function
// result = first arg
// raised to the power
// of the second arg.
function power
// code omitted
push result
return
```


Insper

Mantendo estado das subrotinas

Quando uma sub-rotina chama outras, esta deve salvar seu estado na pilha, para quando a rotina invocada terminar, seu estado ser recuperado puxando os dados da pilha.

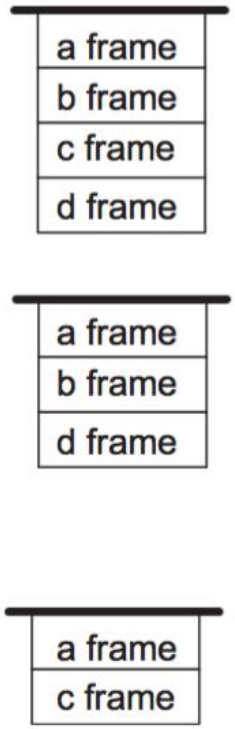
Código

```
subroutine a:
   call b
   call c
   ...
subroutine b:
   call c
   call d
   ...
subroutine c:
   call d
   ...
subroutine d:
   ...
```

Fluxo

```
start a
  start b
    start c
      start d
      end d
    end c
    start d
    end d
  end b
  start c
    start d
    end d
  end c
end a
```

Pilha



Insper

Espaço de Memória para Sub-rotinas

argument:

Espaço para recuperação de dados passados para a sub-rotina. Exemplo:

```
push argument 0
```

local:

Espaço para uso de posições de memórias temporárias para operações da sub-rotina:

```
pop local 0
push local 1
```

this e that:

Ponteiros para memórias alocadas no heap:

```
push constant 30
push constant 2048
pop pointer 1
pop that 0
```


Insper

Construção de sub-rotinas

Em linguagem da Máquina Virtual a Pilha:

- **`function`** *<função> nVars*

  (definição de uma função que possui *nVars* variáveis locais)

- **`call`** *<função> nArgs*

  (chamada da função com *nArgs* argumentos já postos na pilha)

- **`return`**

  (termina a execução da sub-rotina e retorna o controle)

Esta sintaxe vai simplificar o posterior desenvolvimento do parser.

Estrutura de Programas em Alto-Nível

Programa em Jack:

```
class Foo {
  static int x1, x2, x3;
  method int f1(int x) {
    var int a, b;
    ...
  }
  method void f2(int x, int y) {
    var int a, b, c;
    ...
  }
  function int f3(int u) {
    var int x;
    ...
  }
}
```

```
class Bar {
  static int y1, y2;
  function void f1(int u, int v)
  {
    ...
  }
  method void f2(int x) {
    var int a1, a2;
    ...
  }
}
```

Em geral

Programa em pseudo-Jack:

```
class Foo {
  static staticsList;
  method f1(argsList) {
    var localsList;
    ...
  }
  method f2(argsList) {
    var localsList;
    ...
  }
  function f3(argsList) {
    var localsList;
    ...
  }
}
```

```
class Bar {
  static staticsList;
  function f1(argsList){
    ...
  }
  method f2(argsList) {
    var localsList;
    ...
  }
}
```

Funcionamento das chamadas VM

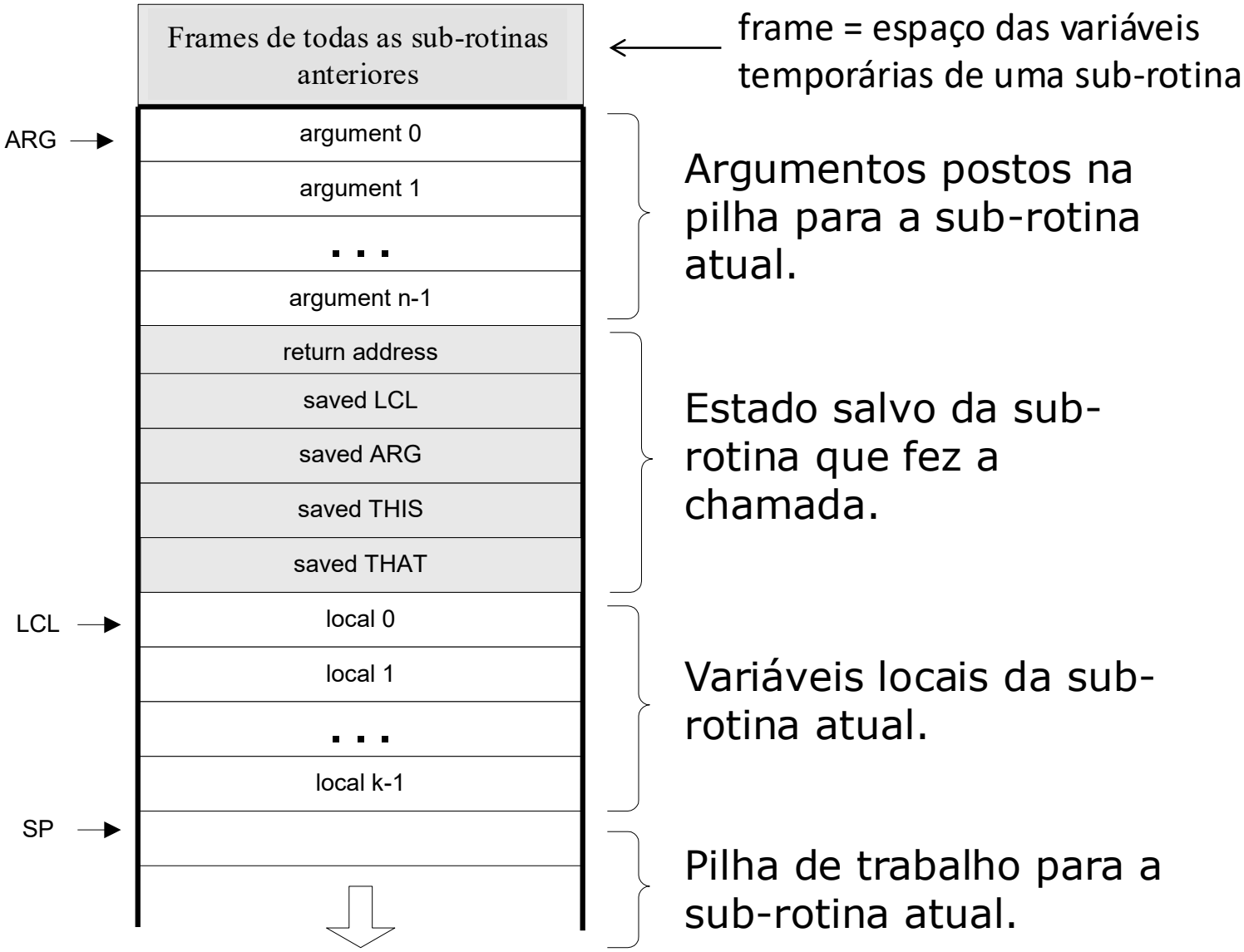
Quando a função f chama a função g, a VM deve:

- Guardar o endereço de retorno;
- Salvar os segmentos virtuais de f;
- Alocar e inicializar (zerar) variáveis locais necessárias para g;
- Definir os indicadores de segmento locais e argumentos de g;
- Transferir o controle para g.

Quando g termina e o controle retorna a f, a VM deve:

- Limpara qualquer lixo de g;
- Restaurar os segmentos virtuais de f;
- Transferir controle de volta para f (saltar para endereço salvo).

Funcionamento da Pilha de Execução



Observações

- Só a função atual está sendo executado, as outras funções estão à espera;
- A função atual só vê o topo da pilha, seus argumentos e variáveis locais, o restante da pilha mantém os estados congelados de todas as funções anteriores;
- A implementação da Máquina Virtual é que controla a alocação e uso dos registradores de ponteiros.

Bootstrap

O *bootstrap* é o processo de inicializar um computador. O termo derivado da expressão em inglês "pulling yourself up by your own bootstraps".



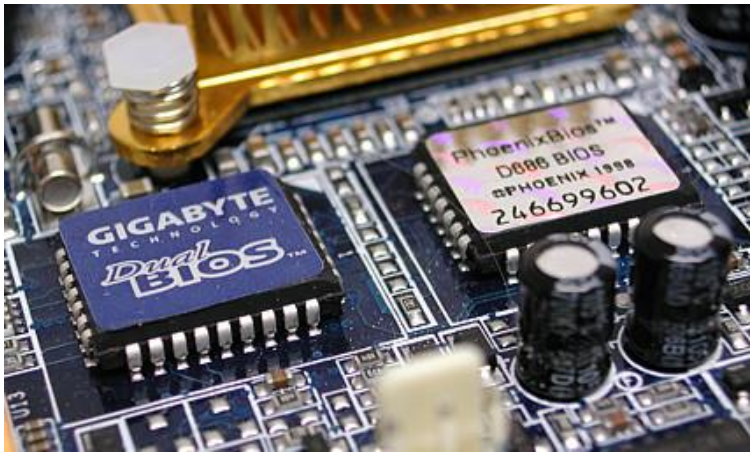
Basicamente quando um computador é ligado pela primeira vez ou reiniciado, ele começa a carregar certos programas em uma certa ordem para deixar o computador operacional.

Durante o processo de start-up, testes de diagnóstico são realizados, como POST (power-on self-test) normalmente carregando o sistema operacional como final do processo.

Boot no PC

Em CPUs x86 a primeira instrução executada é localizada no endereço de memória física FFFF0h em processadores de 16 bits e FFFFFFF0h em processadores de 32 e 64 bits. Esse endereço em geral aponta para o programa de BIOS da ROM.

A rotina de inicialização da BIOS gera uma interrupção INT 19 que lê o setor de inicialização do disco da posição de memória 0000:7C00.





Insper

Insper

www.insper.edu.br