Interpretação e Compilação de Linguagens de Programação

Mestrado Integrado em Engenharia Informática
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

2016-2017

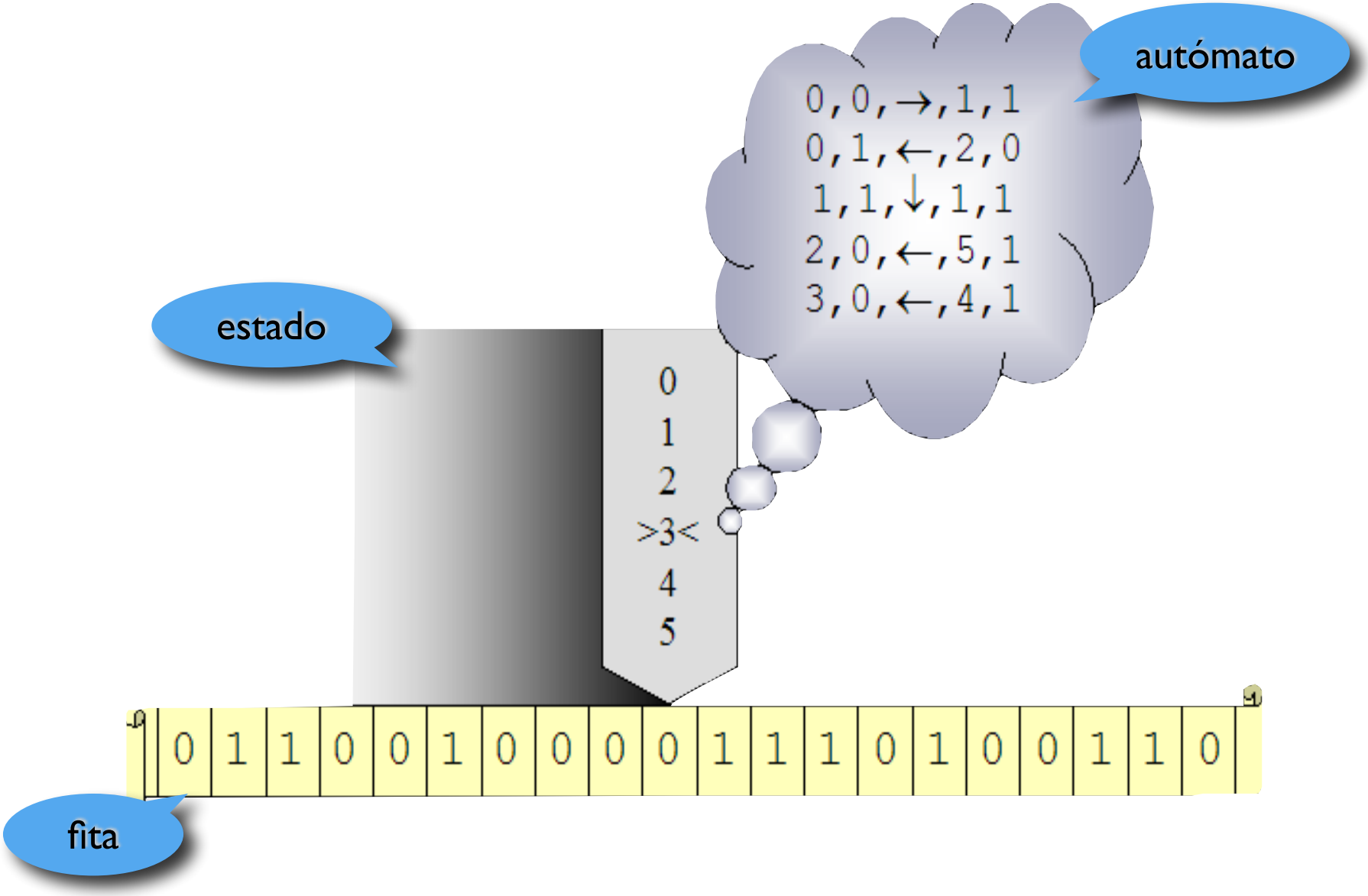
João Costa Seco (joao.seco@di.fct.unl.pt)

Lecture 03
Máquinas virtuais e compilação

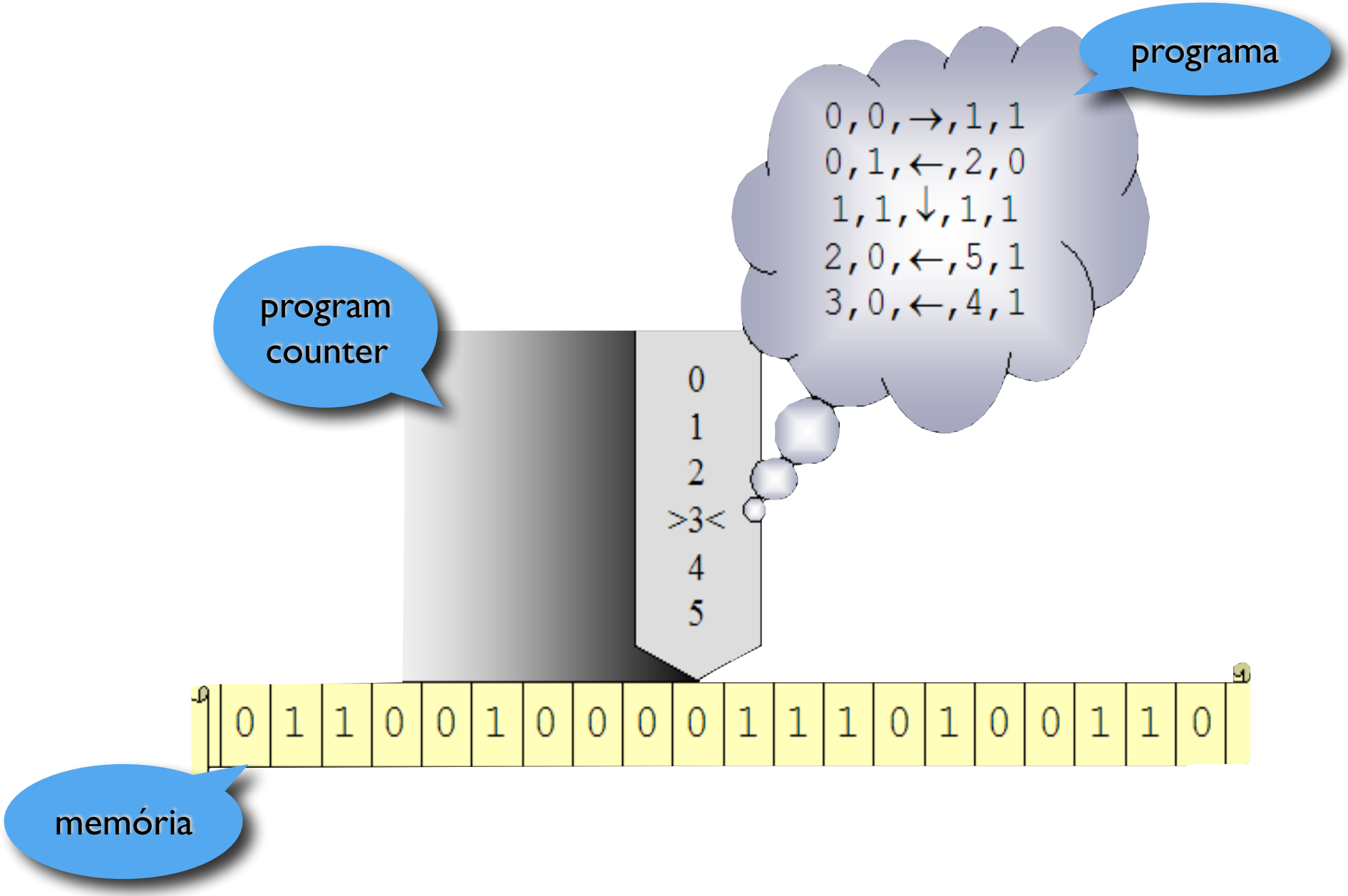
Ambientes de execução: Máquinas Virtuais (software processors)

- Máquina de Turing (Turing, 1931)
  - A primeira …
- SECD (Landin, 1962)
  - Stack, Environment, Code, Dump
- P-code machine (Wirth 1972)
  - Máquina de Pilha, Pascal p-code compiler
- JVM (Sun, 1995)
  - Multi-threaded, tipificada, dedicada à linguagem Java
- CLR (Microsoft, 2000)
  - Multi-threaded, tipificada, Multi linguagem
- LLVM (Vikram Adve and Chris Lattner, 2000)
  - Intermediate language, compiler infrastructure

Máquina de Turing (Turing, 1931)

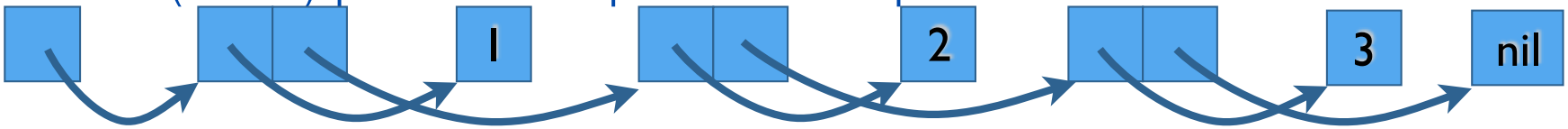


Máquina de Turing (Turing, 1931)



SECD - machine (Landin, 1962)

- A primeira desenhada para implementar o cálculo lambda
- Tem quatro registos a apontar para listas ligadas
  - S : stack
  - E : Environment
  - C : Code
  - D : Dump
- Cada posição da memória pode ter um átomo (12) ou uma lista (um par com dois endereços (o do primeiro elemento e o da lista que se segue)
- A lista (1 2 3) pode ser representada por:



- As instruções: nil, ldc, ld, sel, join, ap, ret, dum, rap

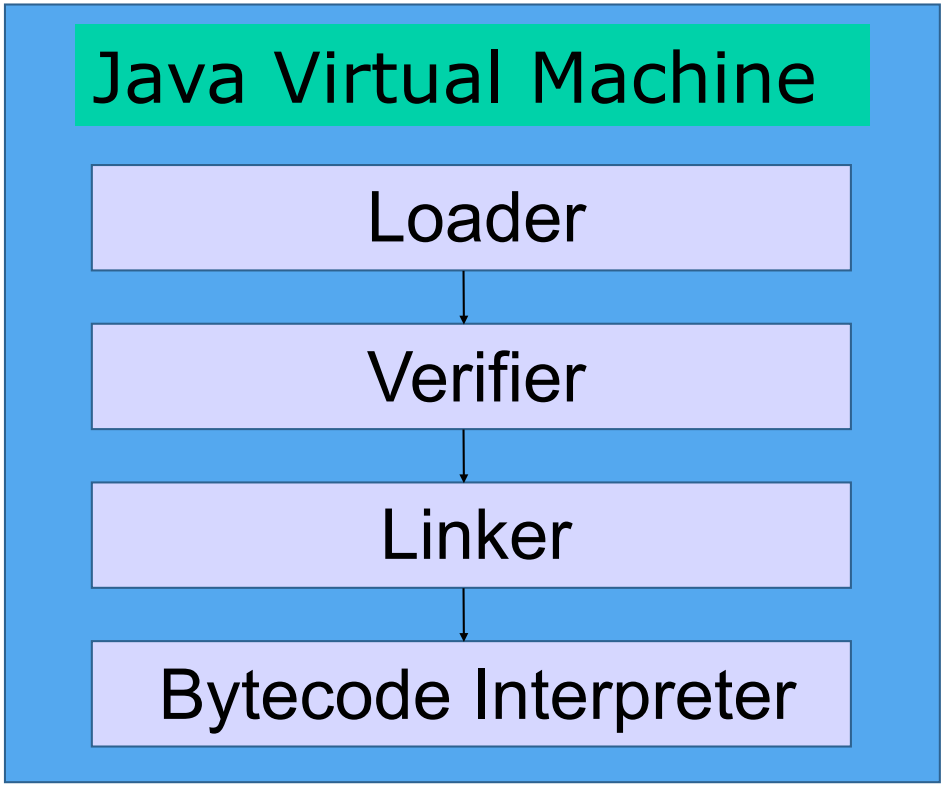
P-code machine (Wirth 1972)

- É uma máquina de pilha o que quer dizer que as instruções da máquina vão buscar os operandos à pilha e colocam o seu resultado de volta na pilha.

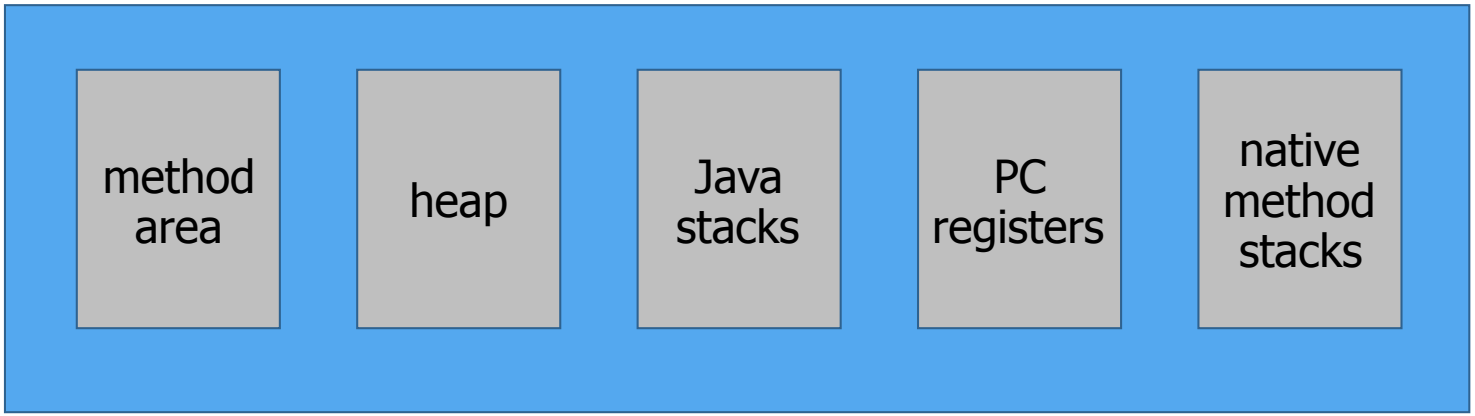
- Máquina simples de implementar apenas com uma pilha partilhada entre informação de controlo e dados.

```
procedure interpret;
   const stacksize = 500;

   var
      p,b,t: integer; {program-, base-, topstack-registers}
      i: instruction; {instruction register}
      s: array [1..stacksize] of integer; {datastore}

   function base(l: integer): integer;
      var b1: integer;
   begin
      b1 := b; {find base l levels down}
      while l > 0 do
         begin b1 := s[b1]; l := l - 1
         end;
      base := b1
   end {base};

begin
   writeln(' start pl/0');
   t := 0; b := 1; p := 0;
   s[1] := 0; s[2] := 0; s[3] := 0;
   repeat
      i := code[p]; p := p + 1;
      with i do
      case f of
      lit: begin t := t + 1; s[t] := a end;
      opr: case a of {operator}
           0: begin {return}
                 t := b - 1; p := s[t + 3]; b := s[t + 2];
              end;
           1: s[t] := -s[t];
           2: begin t := t - 1; s[t] := s[t] + s[t + 1] end;
```

Java Virtual Machine (Sun, 1995)

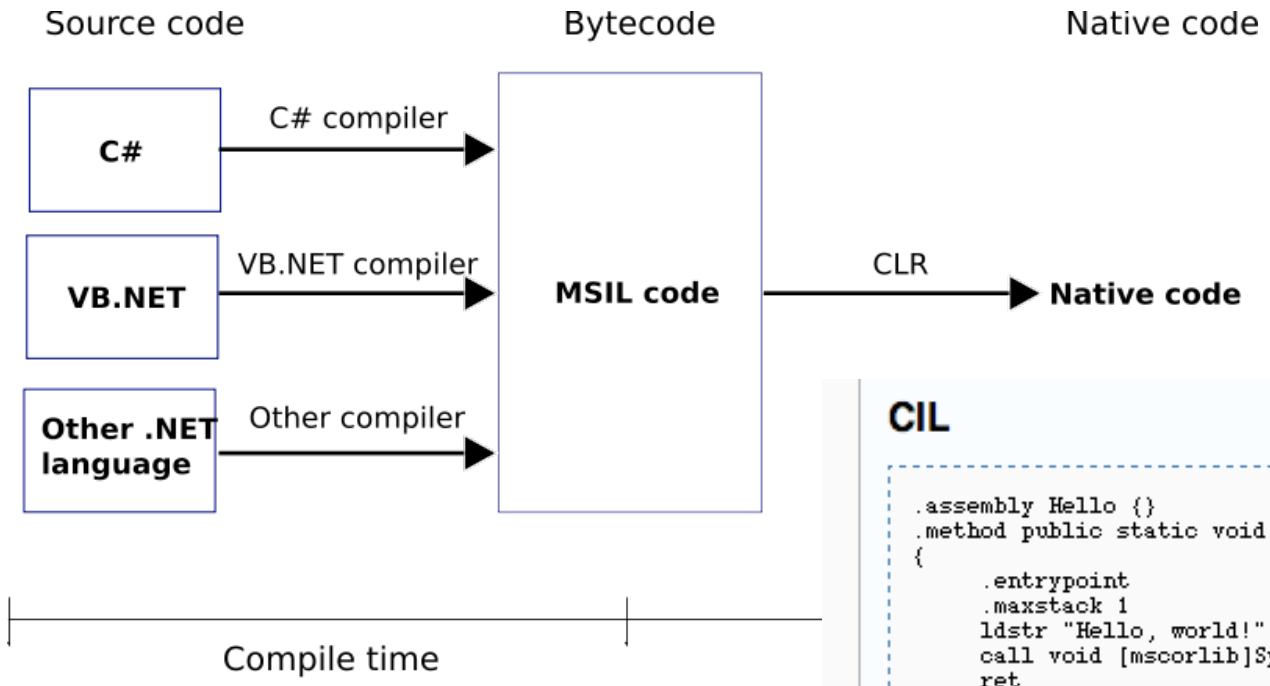



```
// Bytecode stream: 03 3b 84
// 00 01 1a 05 68 3b a7 ff f9
// Disassembly:
iconst_0 // 03
istore_0 // 3b
iinc 0, 1 // 84 00 01
iload_0 // 1a
iconst_2 // 05
imul // 68
istore_0 // 3b
goto -7 // a7 ff f9
```



Common Language Runtime (Microsoft, 2000)

- Máquina de pilha, base para a plataforma Microsoft .NET
- Desenhada para executar programas de várias linguagens.
- CIL - Common Intermediate Language



CIL

```
.assembly Hello {}
.method public static void Main() cil managed
{
      .entrypoint
      .maxstack 1
      ldstr "Hello, world!"
      call void [mscorlib]System.Console::WriteLine(string)
      ret
}
```

Leituras: Ideias a procurar!!

http://en.wikipedia.org/wiki/Turing_machine

http://en.wikipedia.org/wiki/SECD_machine

http://en.wikipedia.org/wiki/P-code_machine

http://en.wikipedia.org/wiki/Common_Language_Runtime

http://en.wikipedia.org/wiki/JVM

LLVM

- Low-level virtual machine, but really
  a compiler infrastructure for designing compiler tools
- Designed to easily define compiler optimisations.
- Provides an intermediate representation and tools.
- University of Illinois, open-source, Apple Xcode.
- clang replaces gcc entirely with LLVM IL.

http://llvm.org

LLVM - Compiler Infrastructure

- Register based intermediate language
- Typed registers and instructions
- Single static assignment
- Defines Basic block graphs (Phi operation)

```
%03 = add i32 %01 %02
%04 = icmp slt i32 %03 0
br i1 %04, label %L0, label %L1
L0:
%05 = add i32 %03 2
br label %end
L1:
%06 = add i32 %03 3
br label %end
end:
%07 = phi i32 [0, %L0], [%03,%L1]
```

```
                                i1
                               i32
                           i1942652
                              half
                             float
                            double
                              fp128
                          x86_fp80
                         ppc_fp128
                        [40 x i32]
              [12 x [10 x float]]
                        [4 x i32]*
                      i32 (i32*)*
                 {i32, i32, i32}
             {float, i32 (i32)*}
```

http://llvm.org/docs/LangRef.html

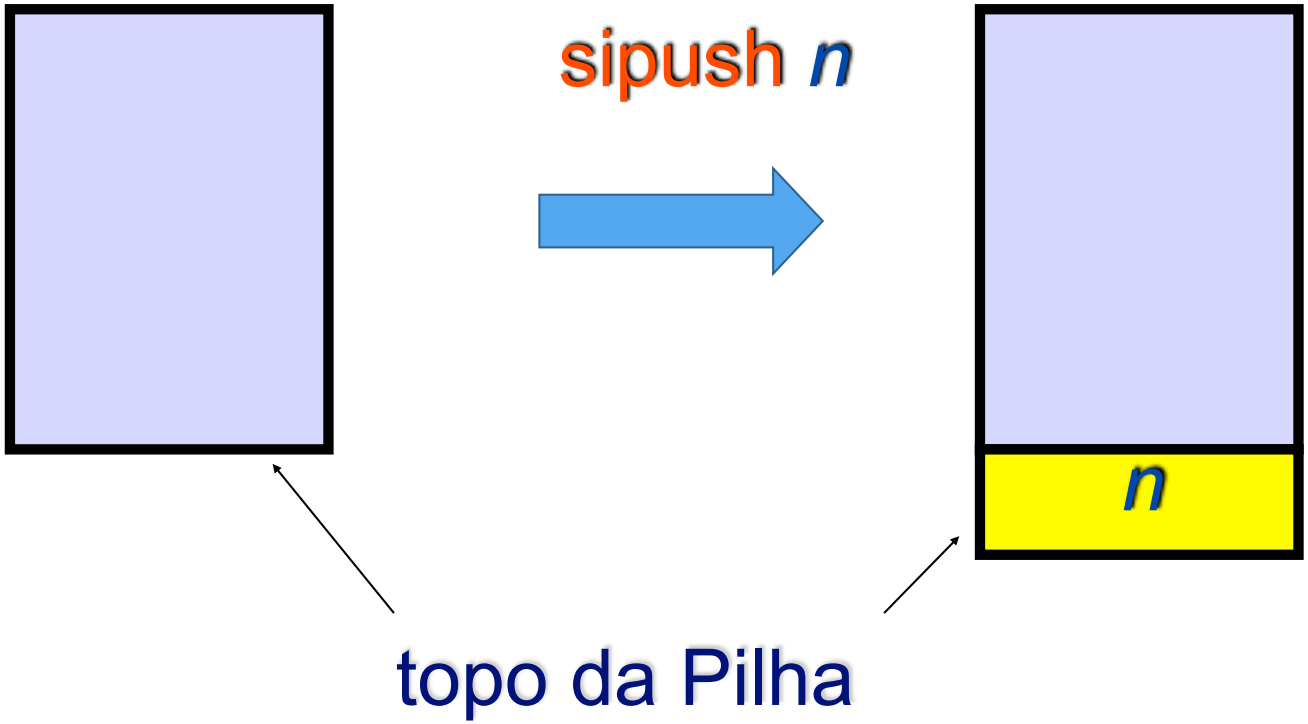
Instruções da JVM

- Máquina de pilha:
  - todas as instruções consomem argumentos do topo da Pilha, e deixam um resultado no topo da Pilha.

- "Primeiras" (5) instruções da JVM:
  - **sipush** n : Carrega o valor n (short integer) no topo da pilha
  - **iadd**: Retira dois valores inteiros do topo da pilha e coloca na pilha o resultado da soma
  - **imul** : idem para a multiplicação
  - **idiv** : idem para a divisão
  - **isub** : idem para a subtracção

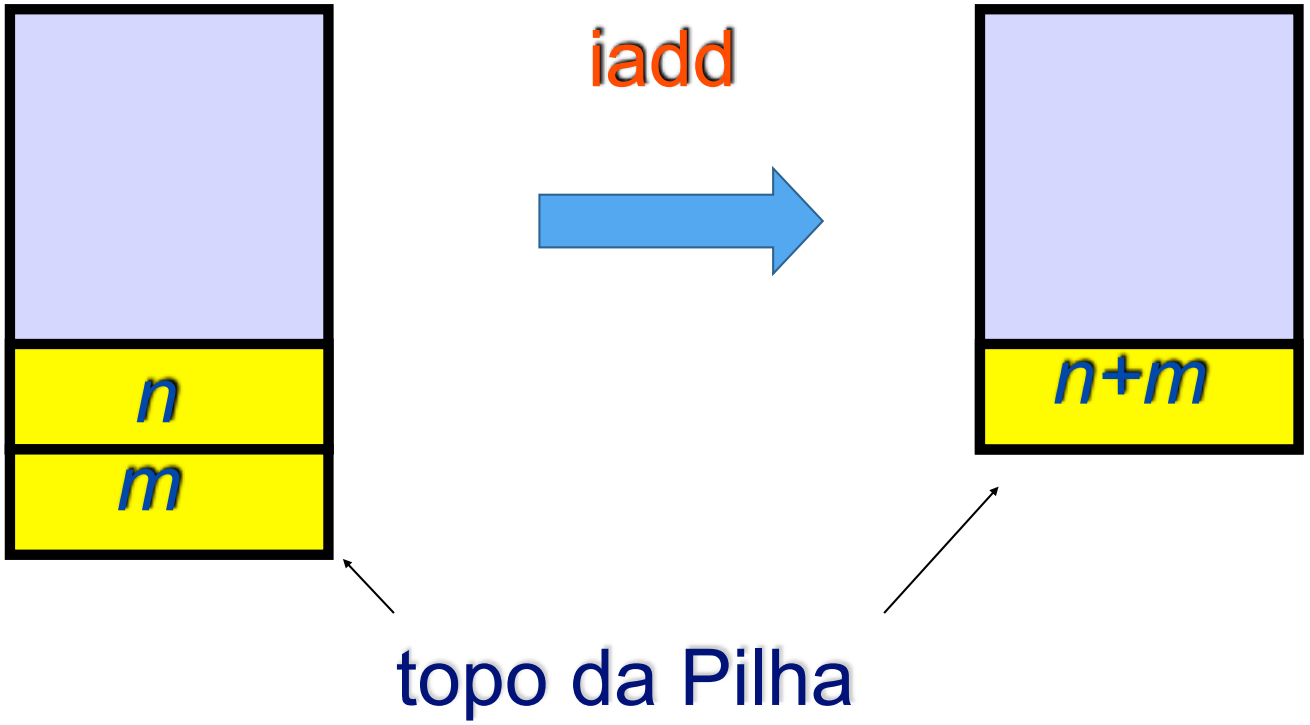
Common Language Runtime

- "Primeiras" (5) instruções: sipush *n*, iadd, imul, idiv, isub.
- Load Constant (sipush *n*)



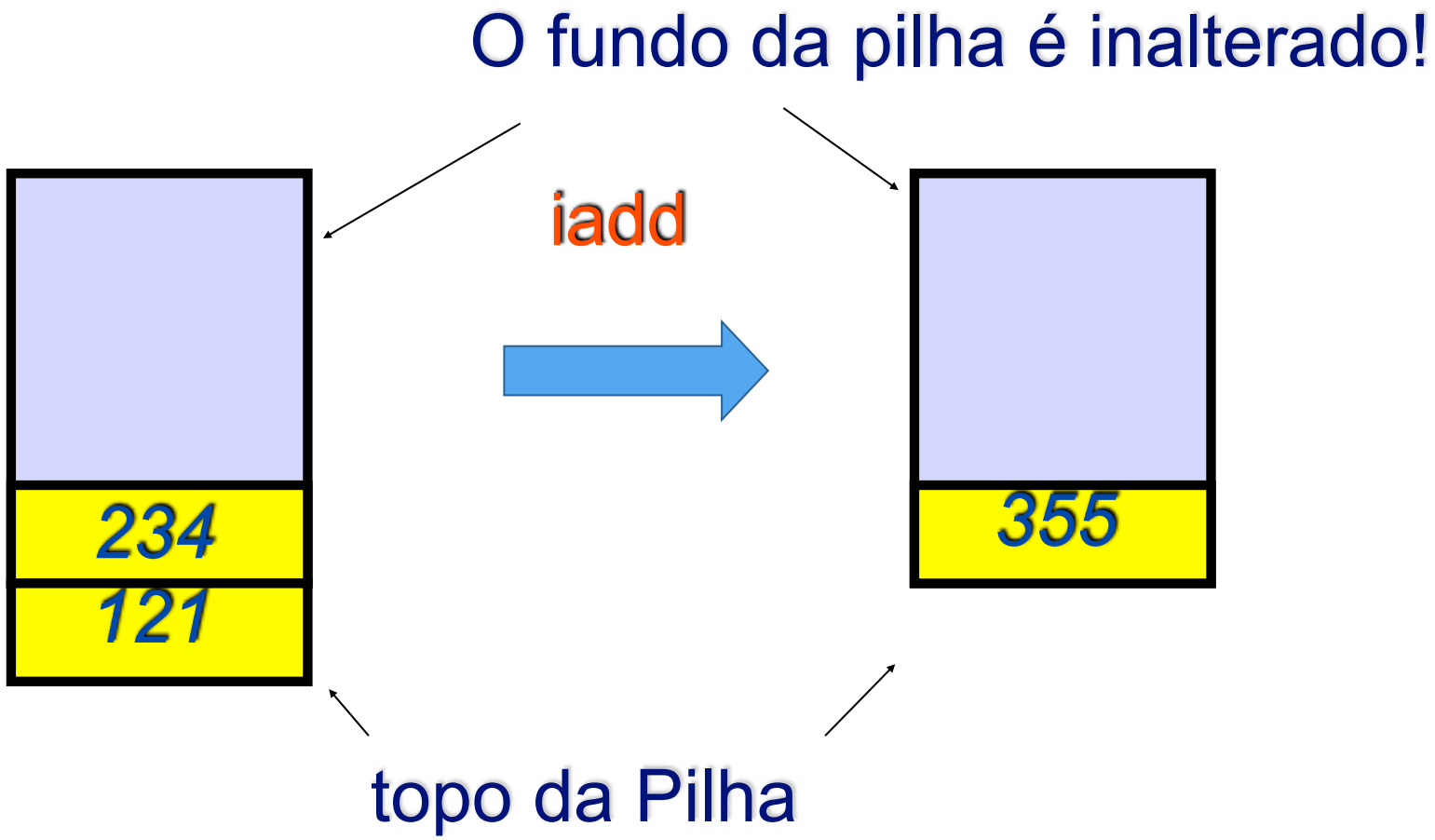
Common Language Runtime

- "Primeiras" (5) instruções: sipush *n*, iadd, imul, idiv, isub.
- Add (iadd)



Common Language Runtime

- "Primeiras" (5) instruções: sipush *n*, iadd, imul, idiv, isub.
- Add (iadd)



Common Language Runtime

- "Primeiras" (5) instruções: sipush ***n***, iadd, imul, idiv, isub.
- Comp("2+2*(7-2)")
- Comp(add(num(2),mul(num(2),sub(num(7),num(2))))) =

Common Language Runtime

- "Primeiras" (5) instruções: sipush *n*, iadd, imul, idiv, isub.
- Comp("2+2*(7-2)")
- Comp(add(num(2),mul(num(2),sub(num(7),num(2))))) =

```
sipush 2
sipush 2
sipush 7
sipush 2
isub
imul
iadd
```

Compilador de CALC

- Algoritmo comp(E) para traduzir uma expressão E qualquer de CALC numa sequência de instruções CLR

$$comp : CALC \rightarrow CodeSeq$$

se E é da forma **num**(n):	comp(E) ≜ < **`sipush n`** >
se E é da forma **add**(E',E''):	s1 = comp(E'); s2 = comp(E'');
	comp(E) ≜ s1 @ s2 @ < **`iadd`** >
se E é da forma **mul**(E',E''):	v1 = comp(E'); v2 = comp(E'');
	comp(E) ≜ s1 @ s2 @ < **`imul`** >
se E é da forma **sub**(E',E''):	v1 = comp(E'); v2 = comp(E'');
	comp(E) ≜ s1 @ s2 @ < **`isub`** >
se E é da forma **div**(E',E''):	v1 = comp(E'); v2 = comp(E'');
	comp(E) ≜ s1 @ s2 @ < **`idiv`** >

Compilador de CALC

- Algoritmo comp(E) para traduzir uma expressão E qualquer de CALC numa sequência de instruções CLR

$$comp : CALC \rightarrow CodeSeq$$

comp(**num**(n)) ≜ < **sipush n** >

comp(**add**(E',E'')) ≜ comp(E') @ comp(E'') @ < **iadd** >

comp(**mul**(E',E'')) ≜ comp(E') @ comp(E'') @ < **imul** >

comp(**sub**(E',E'')) ≜ comp(E') @ comp(E'') @ < **isub** >

comp(**div**(E',E'')) ≜ comp(E') @ comp(E'') @ < **idiv** >

Correcção do Compilador

- Algoritmo comp(E) para traduzir uma expressão E qualquer da linguagem CALC numa sequência de instruções da linguagem CIL (CLR)

$$comp : CALC \rightarrow CodeSeq$$

- **Propriedade de Correcção:** Quando a sequência de instruções comp(E) é executada num estado da máquina virtual em que a pilha está no estado *p*, quando termina deixa sempre a máquina no estado push(*v*,*p*), em que *v* é o valor da expressão E.

Demo