SBVCONC: Construção de Compiladores

Aula 10: CVM (CPRL *Virtual Machine*)

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

CVM (CPRL *Virtual Machine*)

- CVM:
  - É um computador hipotético projetado para simplificar a fase de geração de código do compilador da CPRL;
  - Usa uma arquitetura baseada em pilhas, ou seja, a maioria das instruções esperam ou armazenam valores na pilha, ou ambos;
  - Possui quatro registradores de propósito específico internos, mas nenhum registrador de propósito geral;
- A memória é organizada em bytes de 8 bits. Cada byte é endereçável diretamente;
- Uma palavra é um agrupamento lógico de 4 bytes consecutivos na memória. O endereço de uma palavra é o endereço de seu primeiro byte (o mais abaixo).

Representando Tipos Primitivos

- Valores booleanos são representados em um único byte:
  - 0 significa falso;
  - Qualquer valor diferente de 0 é interpretado como verdadeiro;
- Valores do tipo caractere (Unicode) usam 2 bytes:
  - Plano 0 ou Plano Multilíngue Básico (*Basic Multilingual Plane*);
  - Pontos de código variam de U+0000 a U+FFFF;
- Valores inteiros usam uma palavra:
  - 32-bits em complemento de 2.

Instruções da CVM

- Cada código de operação (*operation code* → opcode) das instruções da CVM ocupa um byte na memória;
- Algumas instruções recebem um ou dois argumentos, os quais estão sempre localizados nas palavras que seguem imediatamente as instruções na memória;
- Dependendo do opcode, um argumento pode ser:
  - Um único byte;
  - Dois bytes (exemplo: um caractere);
  - Quatro bytes (exemplo: um inteiro ou um endereço de memória);
  - Múltiplos bytes (exemplo: um literal de string).
- A maioria das instruções obtêm seus operandos da pilha;
- Em geral, os operandos são removidos da pilha sempre que uma instrução é executada e quaisquer resultados gerados são inseridos no topo da pilha.

Instruções da CVM
Exemplos

- `ADD`: remove dois inteiros do topo da pilha e empilha a soma deles de volta;
- `INC`: adiciona 1 ao inteiro que está no topo da pilha;
- `LOADW` (*load word*): carrega/empilha uma palavra (quatro bytes consecutivos) na pilha. O endereço do primeiro byte da palavra é obtido removendo-o do topo da pilha;
- `CMP` (*compare*): remove dois inteiros do topo da pilha e os compara. Empilha um byte que representa -1, 0 ou 1 de volta à pilha dependendo se o primeiro inteiro é menor que, igual a, ou maior que o segundo inteiro, respectivamente.

Registradores

- Quatro registradores internos de 32-bits (não há registradores de propósito geral)
  - PC (*program counter* ou *instruction pointer*): mantém o endereço da próxima instrução que será executada;
  - SP (*stack pointer*): mantém o endereço do topo da pilha. A pilha cresce a partir dos endereços mais baixos da memória em direção aos endereços mais altos;
  - SB (*stack base*): mantém o endereço da parte inferior (início) da pilha. Quando um programa é carregado, SB é inicializado com o endereço do primeiro byte livre da memória;
  - BP (*base pointer*): mantém o endereço base do subprograma que está sendo executado no momento.

Endereçamento Relativo Usando os Registradores SB e BP

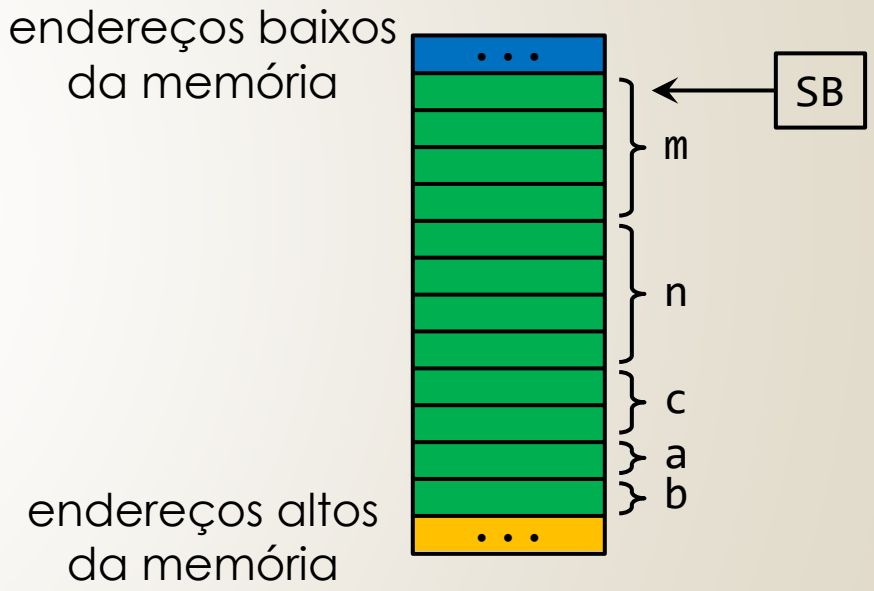
- Variáveis declaradas no escopo de programa são endereçadas em relação ao registrador SB;
- Variáveis declaradas no escopo de subprograma são endereçadas em relação ao registrador BP;
- Exemplo: se SB tiver o valor 112 e uma variável com escopo de programa, identificada por x, tem o endereço relativo igual à 8, então o endereço real de x é [SB] + relAddr(x), ou seja, 120;
- Durante a preparação para a geração de código, o compilador precisa determinar o endereço relativo de todas as variáveis;
- Para os programas que não possuem subprogramas, tanto SB quanto BP apontarão para a mesma localização de memória.

Endereçamento Relativo
Exemplo

- Suponha que um programa contenha as seguintes declarações:

```
var m, n : Integer;
var c : Char;
var a, b : Boolean;
```

- O endereço relativo das variáveis são:
  - `m` tem endereço relativo 0
  - `n` tem endereço relativo 4
  - `c` tem endereço relativo 8
  - `a` tem endereço relativo 10
  - `b` tem endereço relativo 11
- O comprimento total das variáveis para esse programa é 12.

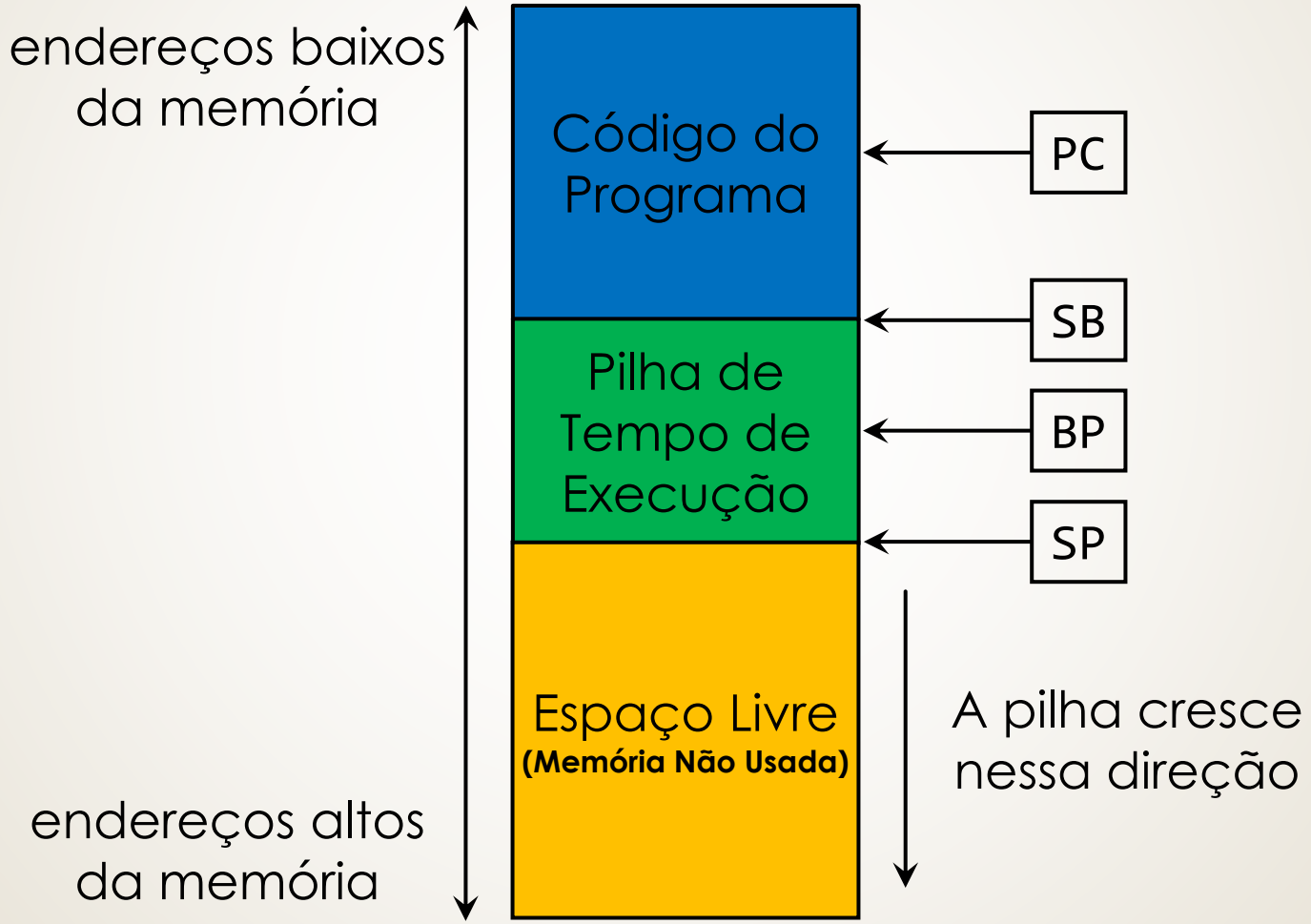


Carregando um Programa

- O código objeto é carregado no início da memória;
- O registrador `PC` é inicializado com 0, o endereço da primeira instrução;
- Os registradores `SB` e `BP` são inicializados com o endereço que segue a última instrução;
- O registrador `SP` é inicializado com `BP-1`;
- A primeira instrução normalmente tem a forma `PROGRAM n`. Quando executada, são alocados `n` bytes no topo da pilha para variáveis globais.

Programa Carregado na Memória

Obs: após a execução de várias instruções



Opcodes LDGADDR e LDLADDR

***Load/Store* Opcodes**
LDGADDR: *load global address*
LDLADDR: *load local address*

- LDGADDR n
  - Carrega endereços globais para variáveis com deslocamento de n
  - Empilha SB+n na pilha;
  - Usado para variáveis declaradas no escopo de programa;
- LDLADDR n
  - Carrega endereços locais para variáveis com deslocamento de n
  - Empilha BP+n na pilha;
  - Usado para variáveis declaradas no escopo de subprograma.

Exemplo em CPRL

```
var m, n : Integer;
var c : Char;
const five := 5;

begin

   m := 7;
   n := five * m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```

Exemplo Desmontado

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STOREW2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: endereço relativo = 0, endereço absoluto = 88
n: endereço relativo = 4, endereço absoluto = 92
c: endereço relativo = 8, endereço absoluto = 96
```

***Load/Store* Opcodes**

- **LDCCH:** *load constant character*
- **LDCINT:** *load constant integer*
- **LDCSTR:** *load constant string*
- **LDGADDR:** *load global address*
- **LOAD2B:** *load two bytes*
- **LOADW:** *load word*
- **STORE2B:** *store two bytes*
- **STOREW:** *store word*

***Arithmetic* Opcodes**

- **MUL:** *multiply*

***Program* Opcodes**

- **HALT:** *halt*
- **PROGRAM:** *program*

***I/O* Opcodes**

- **PUTCH:** *put character*
- **PUTEOL:** *put end-of-line*
- **PUTINT:** *put integer*
- **PUTSTR:** *put string*

Exemplo em CPRL

```
var m, n : Integer;
var c : Char;
const five := 5;

begin

   m := 7;
   n := five * m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```

Exemplo Desmontado

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STORE2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: endereço relativo = 0, endereço absoluto = 88
n: endereço relativo = 4, endereço absoluto = 92
c: endereço relativo = 8, endereço absoluto = 96
```

***Load/Store* Opcodes**

```
 LDCCH: load constant character
 LDCINT: load constant integer
 LDCSTR: load constant string
LDGADDR: load global address
 LOAD2B: load two bytes
  LOADW: load word
STORE2B: store two bytes
 STOREW: store word
```

***Arithmetic* Opcodes**

```
MUL: multiply
```

***Program* Opcodes**

```
   HALT: halt
PROGRAM: program
```

***I/O* Opcodes**

```
 PUTCH: put character
PUTEOL: put end-of-line
PUTINT: put integer
PUTSTR: put string
```

Exemplo em CPRL

```
var m, n : Integer;
var c : Char;
const five := 5;

begin

   m := 7;
   n := five * m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```

Exemplo Desmontado

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STORE2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: endereço relativo = 0, endereço absoluto = 88
n: endereço relativo = 4, endereço absoluto = 92
c: endereço relativo = 8, endereço absoluto = 96
```

***Load/Store* Opcodes**

- **LDCCH:** *load constant character*
- **LDCINT:** *load constant integer*
- **LDCSTR:** *load constant string*
- **LDGADDR:** *load global address*
- **LOAD2B:** *load two bytes*
- **LOADW:** *load word*
- **STORE2B:** *store two bytes*
- **STOREW:** *store word*

***Arithmetic* Opcodes**

- **MUL:** *multiply*

***Program* Opcodes**

- **HALT:** *halt*
- **PROGRAM:** *program*

***I/O* Opcodes**

- **PUTCH:** *put character*
- **PUTEOL:** *put end-of-line*
- **PUTINT:** *put integer*
- **PUTSTR:** *put string*

Exemplo em CPRL

```
var m, n : Integer;
var c : Char;
const five := 5;

begin

   m := 7;
   n := five * m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```

Exemplo Desmontado

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STORE2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: endereço relativo = 0, endereço absoluto = 88
n: endereço relativo = 4, endereço absoluto = 92
c: endereço relativo = 8, endereço absoluto = 96
```

***Load/Store* Opcodes**

- **LDCCH:** *load constant character*
- **LDCINT:** *load constant integer*
- **LDCSTR:** *load constant string*
- **LDGADDR:** *load global address*
- **LOAD2B:** *load two bytes*
- **LOADW:** *load word*
- **STORE2B:** *store two bytes*
- **STOREW:** *store word*

***Arithmetic* Opcodes**

- **MUL:** *multiply*

***Program* Opcodes**

- **HALT:** *halt*
- **PROGRAM:** *program*

***I/O* Opcodes**

- **PUTCH:** *put character*
- **PUTEOL:** *put end-of-line*
- **PUTINT:** *put integer*
- **PUTSTR:** *put string*

Exemplo em CPRL

```
var m, n : Integer;
var c : Char;
const five := 5;

begin

   m := 7;
   n := five * m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```

Exemplo Desmontado

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STORE2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: endereço relativo = 0, endereço absoluto = 88
n: endereço relativo = 4, endereço absoluto = 92
c: endereço relativo = 8, endereço absoluto = 96
```

***Load/Store* Opcodes**

- **LDCCH:** *load constant character*
- **LDCINT:** *load constant integer*
- **LDCSTR:** *load constant string*
- **LDGADDR:** *load global address*
- **LOAD2B:** *load two bytes*
- **LOADW:** *load word*
- **STORE2B:** *store two bytes*
- **STOREW:** *store word*

***Arithmetic* Opcodes**

- **MUL:** *multiply*

***Program* Opcodes**

- **HALT:** *halt*
- **PROGRAM:** *program*

***I/O* Opcodes**

- **PUTCH:** *put character*
- **PUTEOL:** *put end-of-line*
- **PUTINT:** *put integer*
- **PUTSTR:** *put string*

Exemplo em CPRL

```
var m, n : Integer;
var c : Char;
const five := 5;

begin

   m := 7;
   n := five * m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```

Exemplo Desmontado

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STORE2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: endereço relativo = 0, endereço absoluto = 88
n: endereço relativo = 4, endereço absoluto = 92
c: endereço relativo = 8, endereço absoluto = 96
```

***Load/Store* Opcodes**

- **LDCCH:** *load constant character*
- **LDCINT:** *load constant integer*
- **LDCSTR:** *load constant string*
- **LDGADDR:** *load global address*
- **LOAD2B:** *load two bytes*
- **LOADW:** *load word*
- **STORE2B:** *store two bytes*
- **STOREW:** *store word*

***Arithmetic* Opcodes**

- **MUL:** *multiply*

***Program* Opcodes**

- **HALT:** *halt*
- **PROGRAM:** *program*

***I/O* Opcodes**

- **PUTCH:** *put character*
- **PUTEOL:** *put end-of-line*
- **PUTINT:** *put integer*
- **PUTSTR:** *put string*

Exemplo em CPRL

```
var m, n : Integer;
var c : Char;
const five := 5;

begin

   m := 7;
   n := five * m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```

Exemplo Desmontado

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STORE2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: endereço relativo = 0, endereço absoluto = 88
n: endereço relativo = 4, endereço absoluto = 92
c: endereço relativo = 8, endereço absoluto = 96
```

***Load/Store* Opcodes**

- **LDCCH:** *load constant character*
- **LDCINT:** *load constant integer*
- **LDCSTR:** *load constant string*
- **LDGADDR:** *load global address*
- **LOAD2B:** *load two bytes*
- **LOADW:** *load word*
- **STORE2B:** *store two bytes*
- **STOREW:** *store word*

***Arithmetic* Opcodes**

- **MUL:** *multiply*

***Program* Opcodes**

- **HALT:** *halt*
- **PROGRAM:** *program*

***I/O* Opcodes**

- **PUTCH:** *put character*
- **PUTEOL:** *put end-of-line*
- **PUTINT:** *put integer*
- **PUTSTR:** *put string*

Exemplo em CPRL

```
var m, n : Integer;
var c : Char;
const five := 5;

begin

   m := 7;
   n := five * m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```

Exemplo Desmontado

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STORE2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: endereço relativo = 0, endereço absoluto = 88
n: endereço relativo = 4, endereço absoluto = 92
c: endereço relativo = 8, endereço absoluto = 96
```

***Load/Store* Opcodes**

- **LDCCH:** *load constant character*
- **LDCINT:** *load constant integer*
- **LDCSTR:** *load constant string*
- **LDGADDR:** *load global address*
- **LOAD2B:** *load two bytes*
- **LOADW:** *load word*
- **STORE2B:** *store two bytes*
- **STOREW:** *store word*

***Arithmetic* Opcodes**

- **MUL:** *multiply*

***Program* Opcodes**

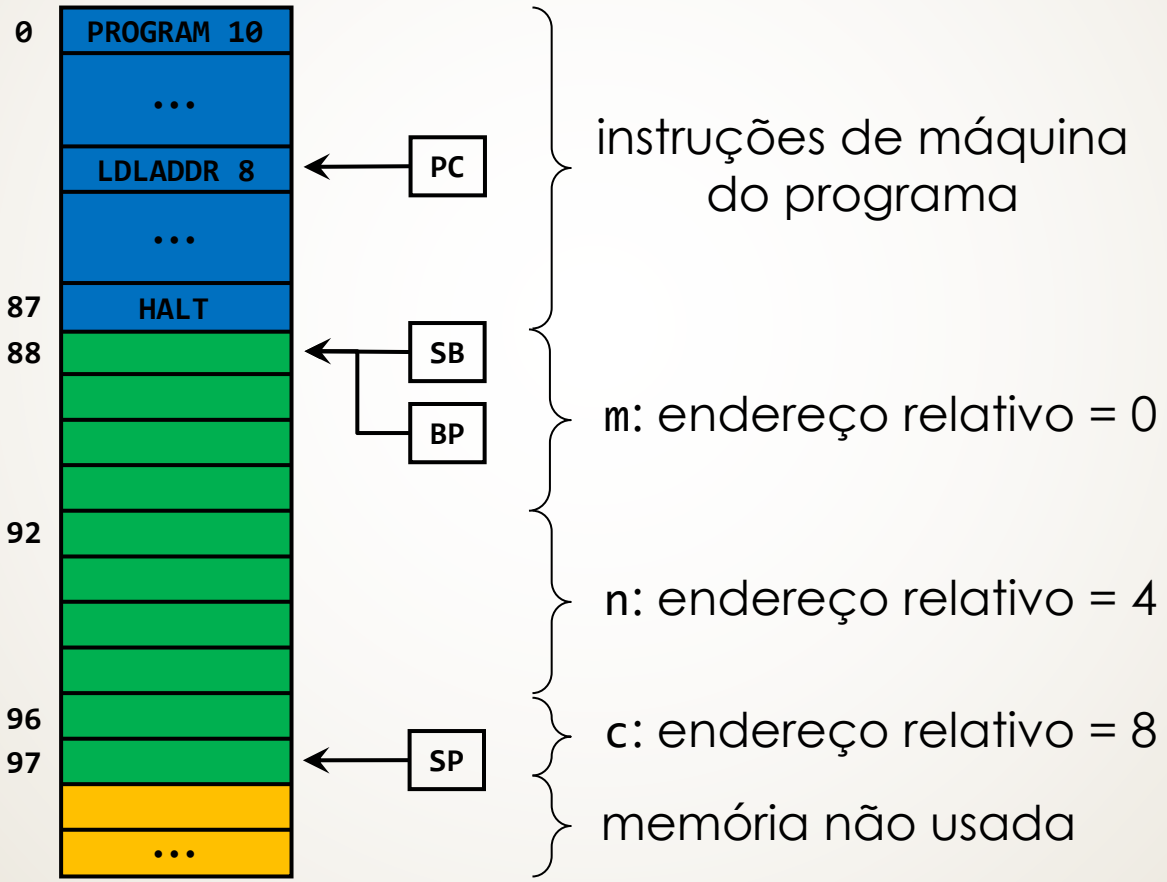
- **HALT:** *halt*
- **PROGRAM:** *program*

***I/O* Opcodes**

- **PUTCH:** *put character*
- **PUTEOL:** *put end-of-line*
- **PUTINT:** *put integer*
- **PUTSTR:** *put string*

Exemplo em CPRL na Memória

Obs: após a execução da primeira instrução



Usando a Pilha para Manter Valores Temporários

- A parte da memória abaixo das instruções da CVM e das variáveis globais é usada como uma pilha de tempo de execução que mantém os registros de ativação (*activation records/stack frames*) dos subprogramas e valores temporários e intermediários;
- Conforme as instruções da máquina são executadas, a pilha aumenta e diminui;
- A pilha de tempo de execução estará vazia tanto no início quanto no fim de cada instrução da CPRL dentro do programa principal.

Usando a Pilha para Manter Valores Temporários Exemplo

- Assuma que:
  - O registrador `SB` tem o valor `100`
  - A variável inteira `x` tem endereço relativo `0`
  - A variável inteira `y` tem valor `5` e endereço relativo `4`
  - A variável inteira `z` tem valor `13` e endereço relativo `8`
- A instrução de atribuição da CPRL:

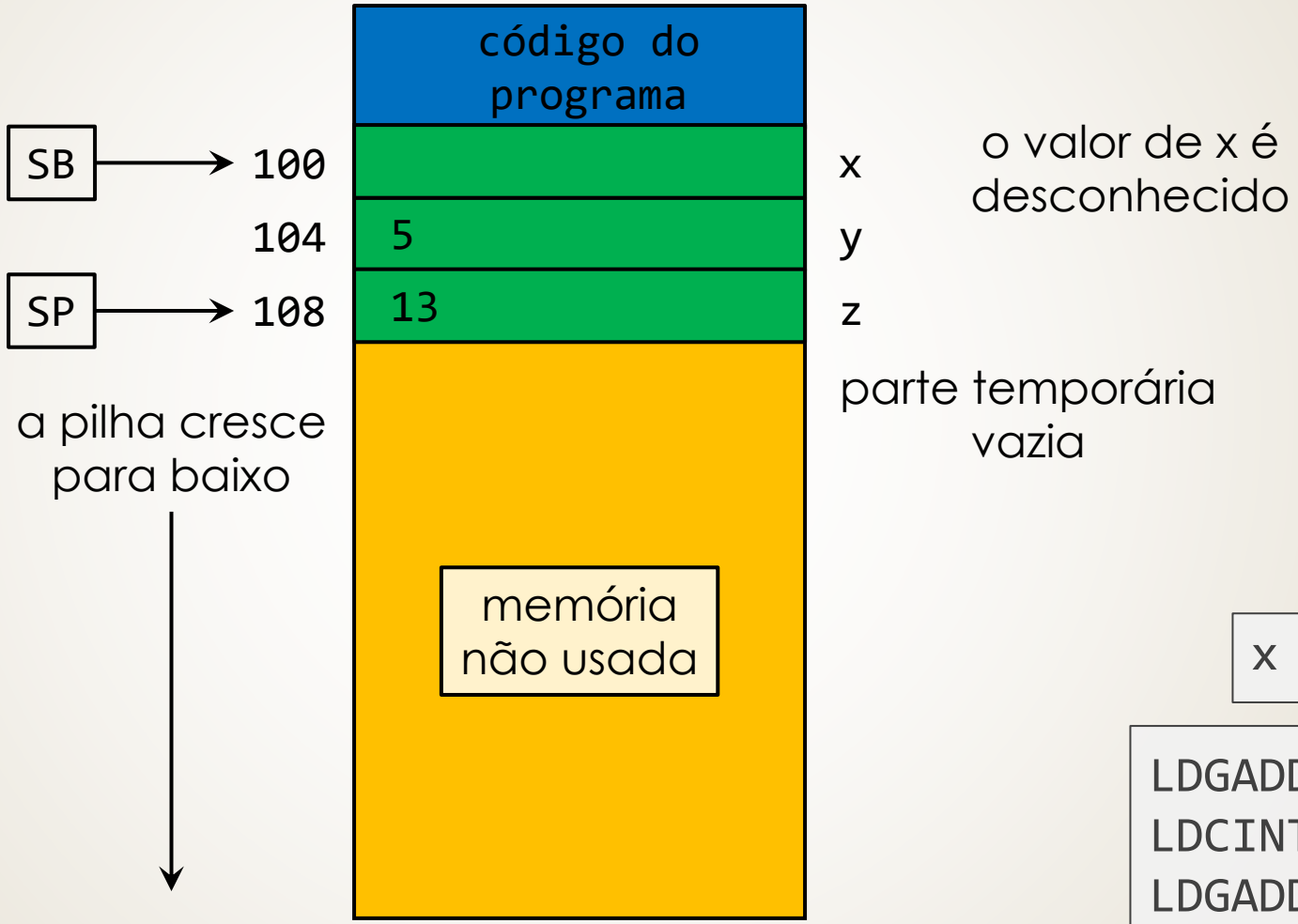
```
x := 2 * y + z;
```

compilará nas seguintes instruções da CVM:

```
LDGADDR 0     LDGADDR 8
LDCINT 2      LOADW
LDGADDR 4     ADD
LOADW         STOREW
MUL
```

Usando a Pilha para Manter Valores Temporários Exemplo

A parte temporária da pilha está vazia no início de uma instrução da CPRL.




```
x := 2 * y + z;
```

```
LDGADDR 0
LDCINT 2
LDGADDR 4
LOADW
MUL
LDGADDR 8
LOADW
ADD
STOREW
```

***Load/Store* Opcodes**

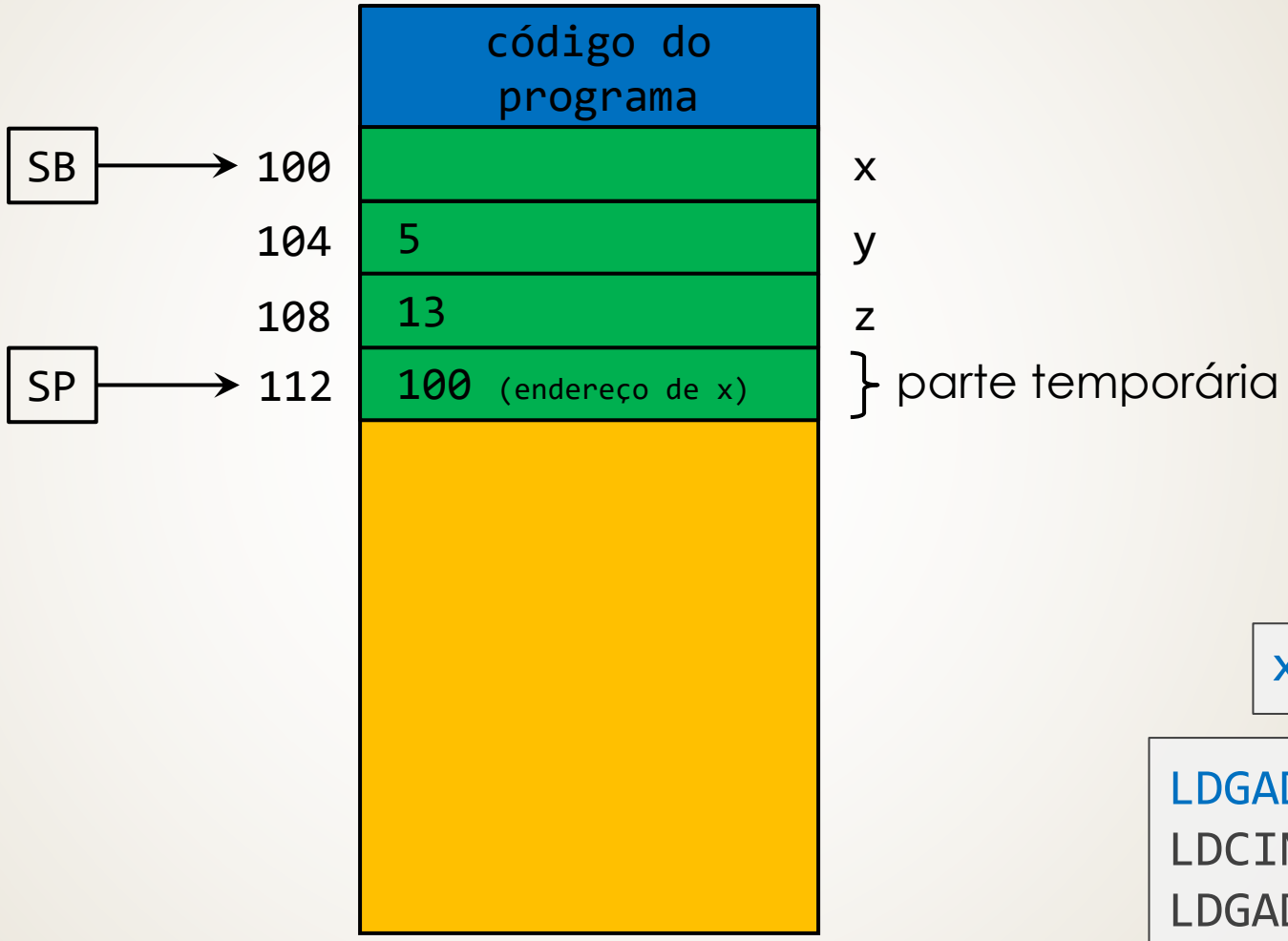
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**

ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários
Exemplo

Após a execução de
`LDGADDR 0`




```
x := 2 * y + z;
```

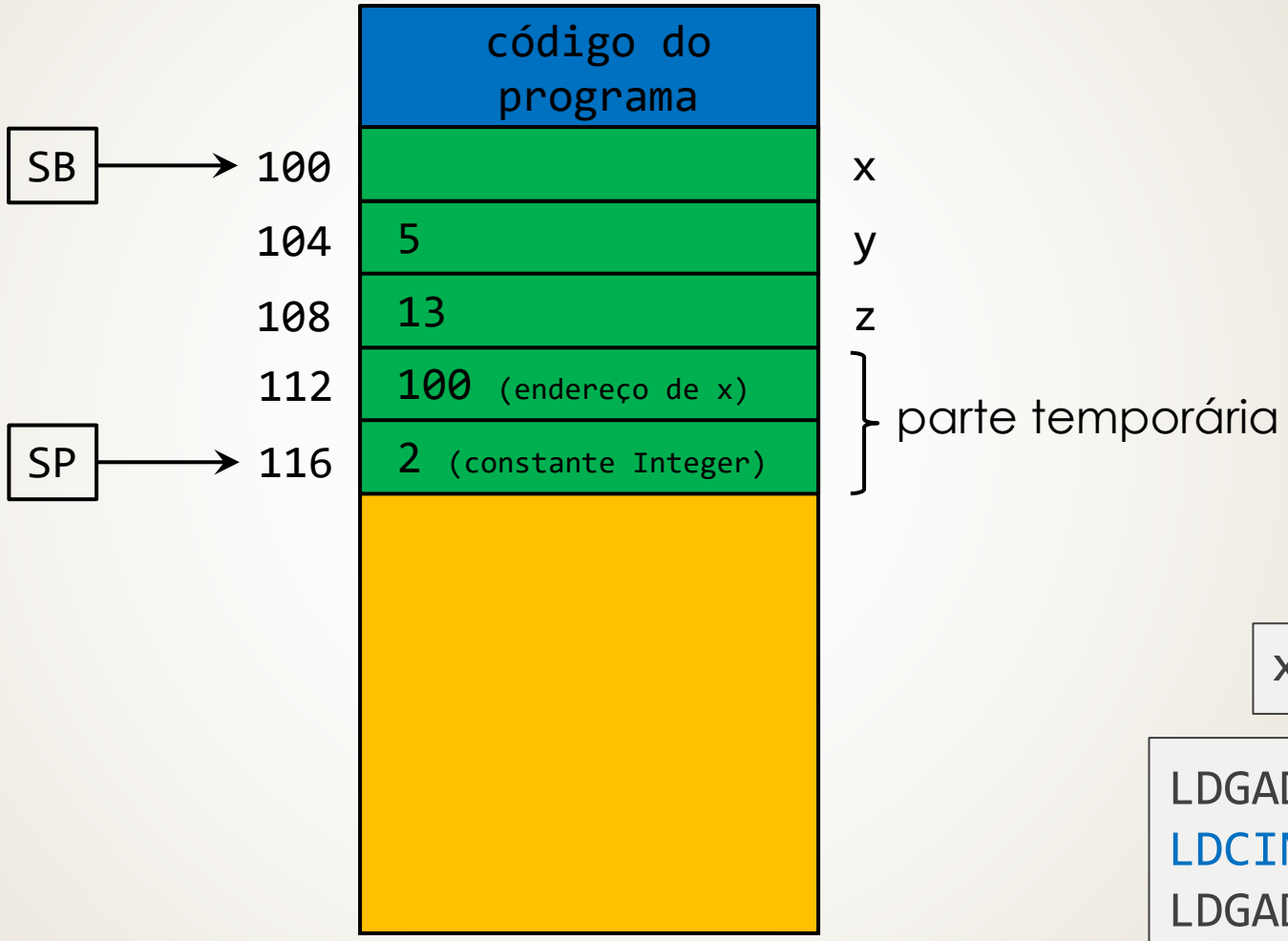
```
LDGADDR 0      LDGADDR 8
LDCINT 2       LOADW
LDGADDR 4      ADD
LOADW          STOREW
MUL
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários
Exemplo

Após a execução de
`LDCINT 2`




```
x := 2 * y + z;
```

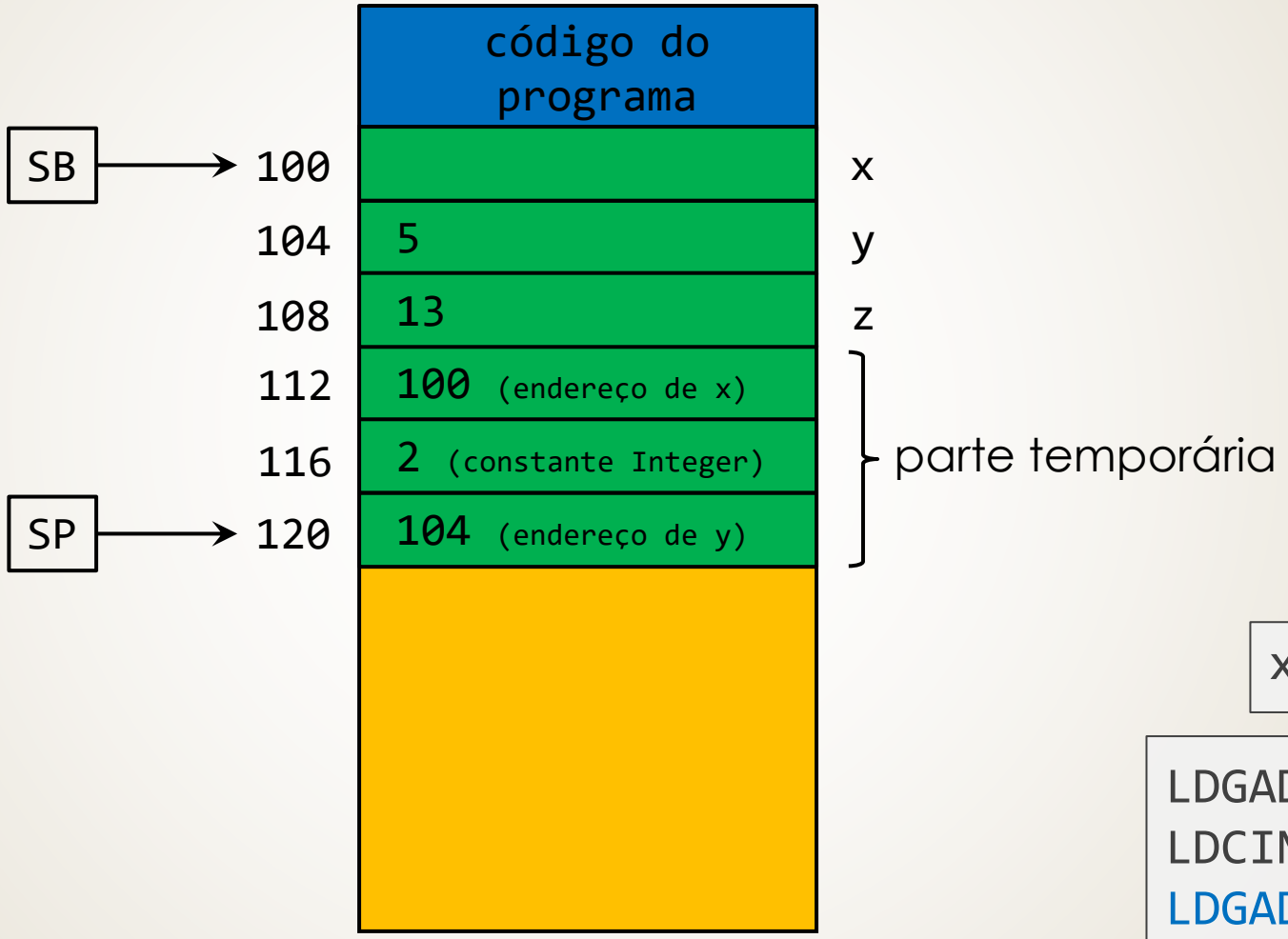
```
LDGADDR 0      LDGADDR 8
LDCINT 2       LOADW
LDGADDR 4      ADD
LOADW          STOREW
MUL
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários
Exemplo

Após a execução de
LDGADDR 4




```
x := 2 * y + z;
```

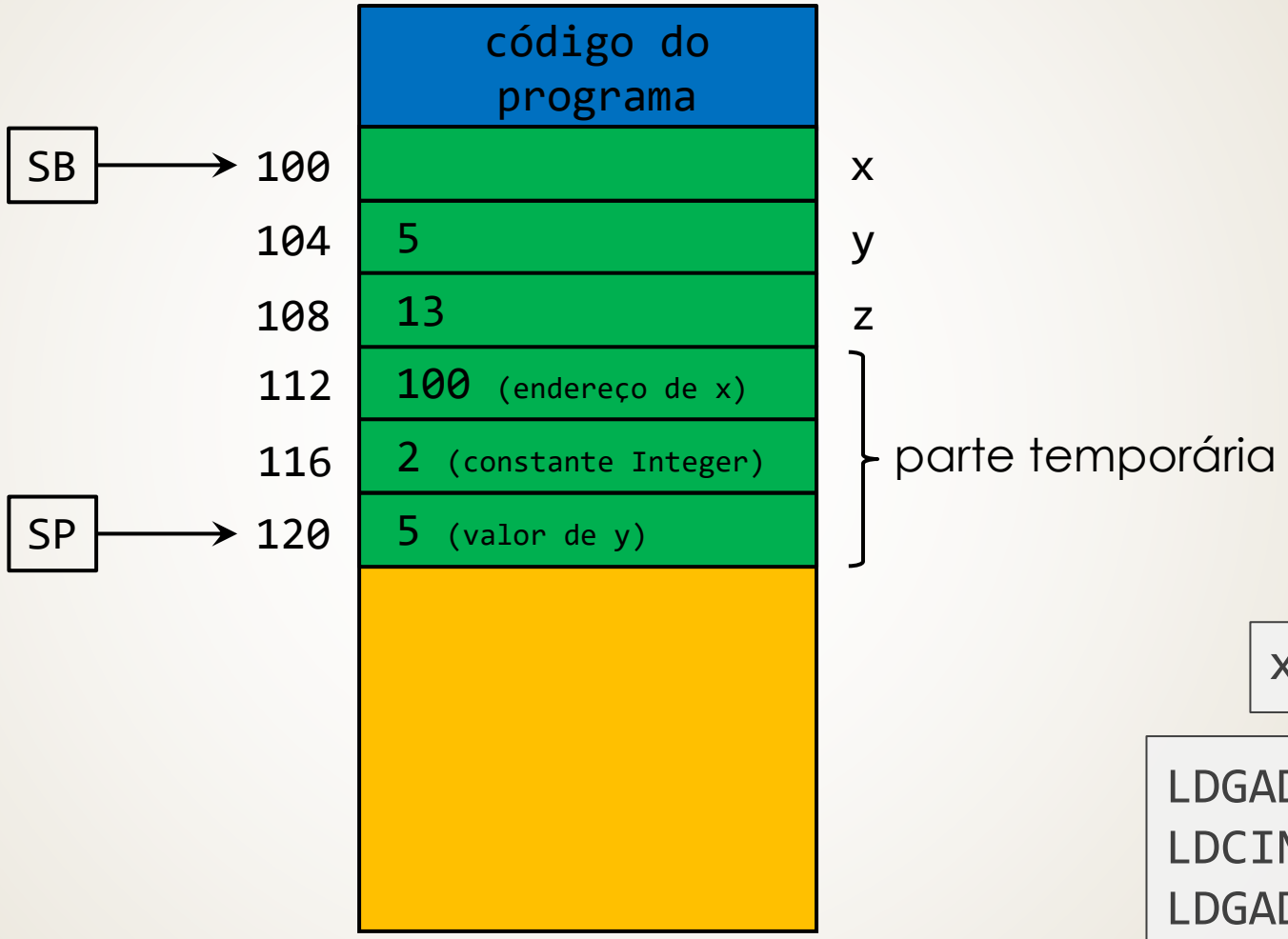
```
LDGADDR 0
LDCINT 2
LDGADDR 4
LOADW
MUL
LDGADDR 8
LOADW
ADD
STOREW
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários
Exemplo

Após a execução de **LOADW**




```
x := 2 * 5 + z;
```

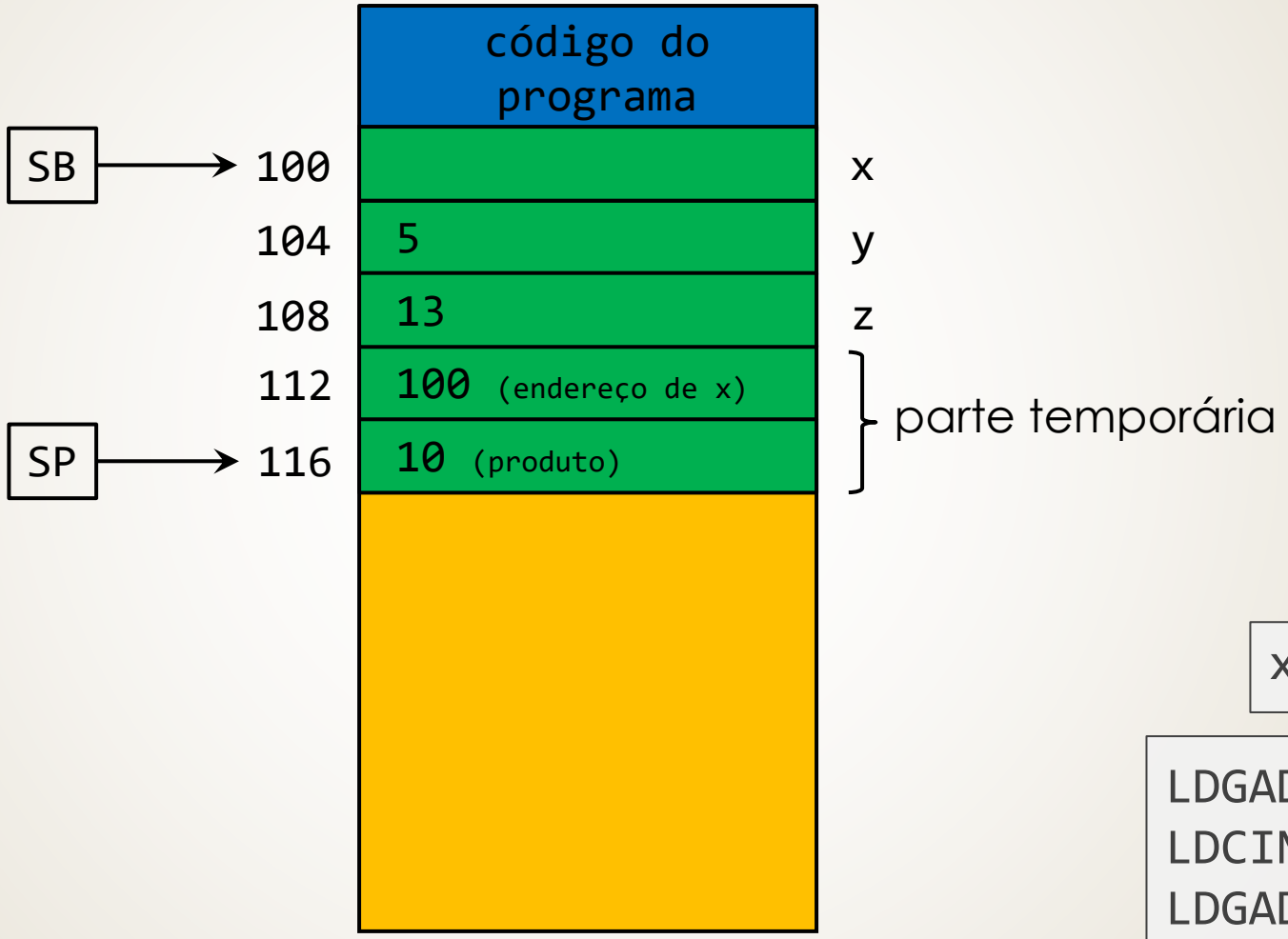
```
LDGADDR 0      LDGADDR 8
LDCINT 2       LOADW
LDGADDR 4      ADD
LOADW          STOREW
MUL
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários
Exemplo

Após a execução de MUL




```
x := 10 + z;
```

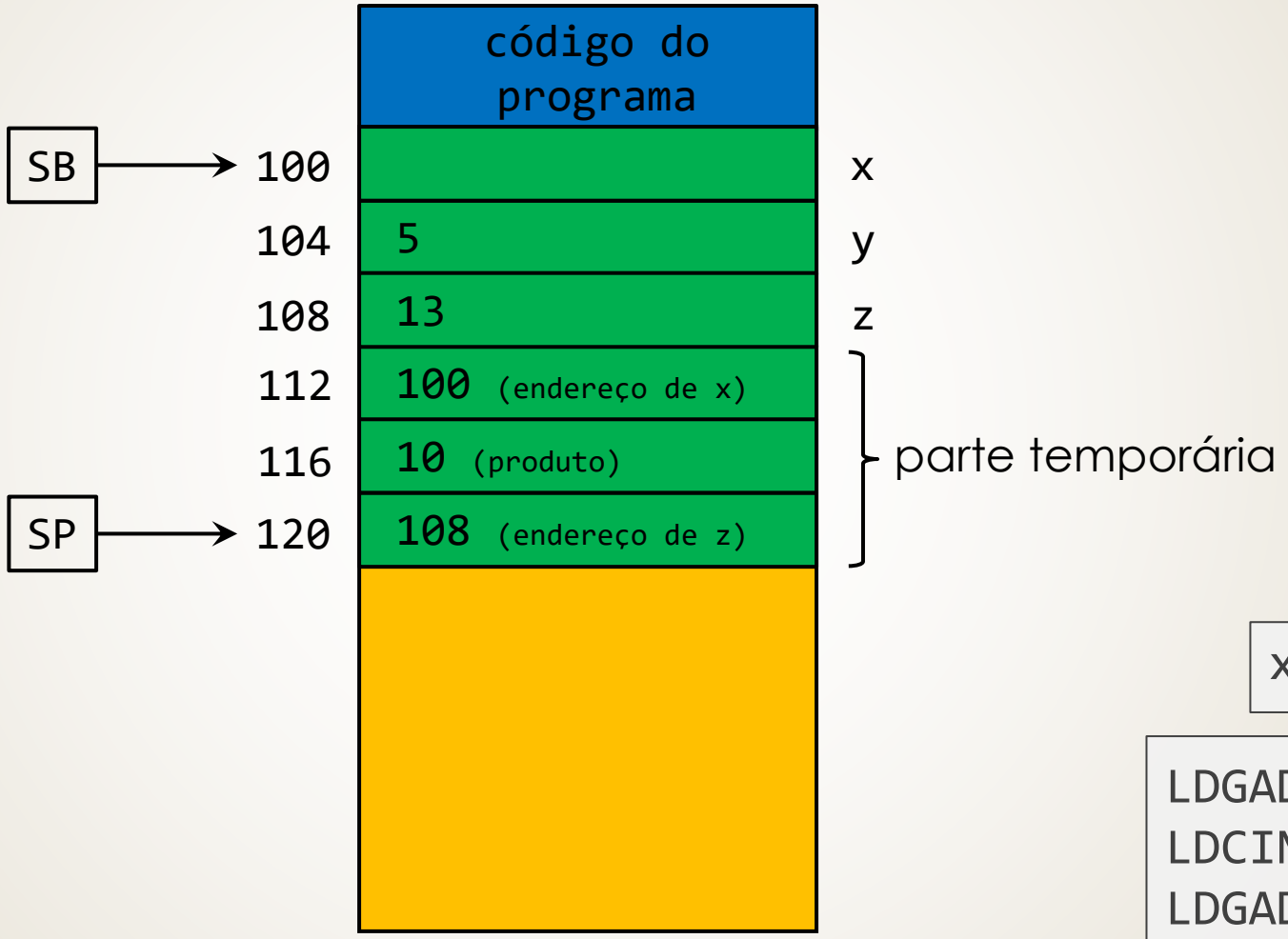
```
LDGADDR 0      LDGADDR 8
LDCINT 2       LOADW
LDGADDR 4      ADD
LOADW          STOREW
MUL
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários
Exemplo

Após a execução de
LDGADDR 8




```
x := 10 + z;
```

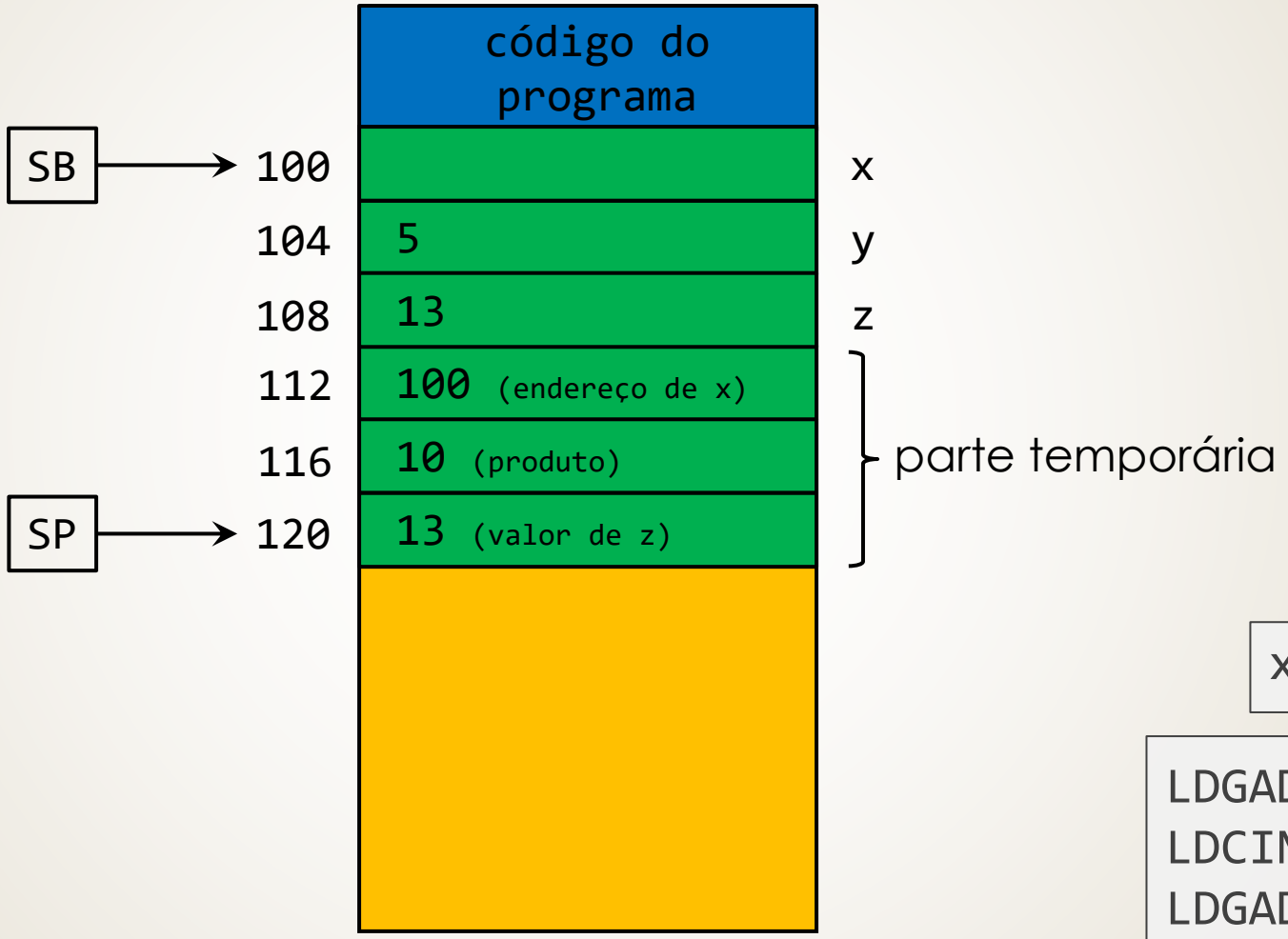
```
LDGADDR 0        LDGADDR 8
LDCINT 2         LOADW
LDGADDR 4        ADD
LOADW            STOREW
MUL
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários
Exemplo

Após a execução de **LOADW**




```
x := 10 + 13;
```

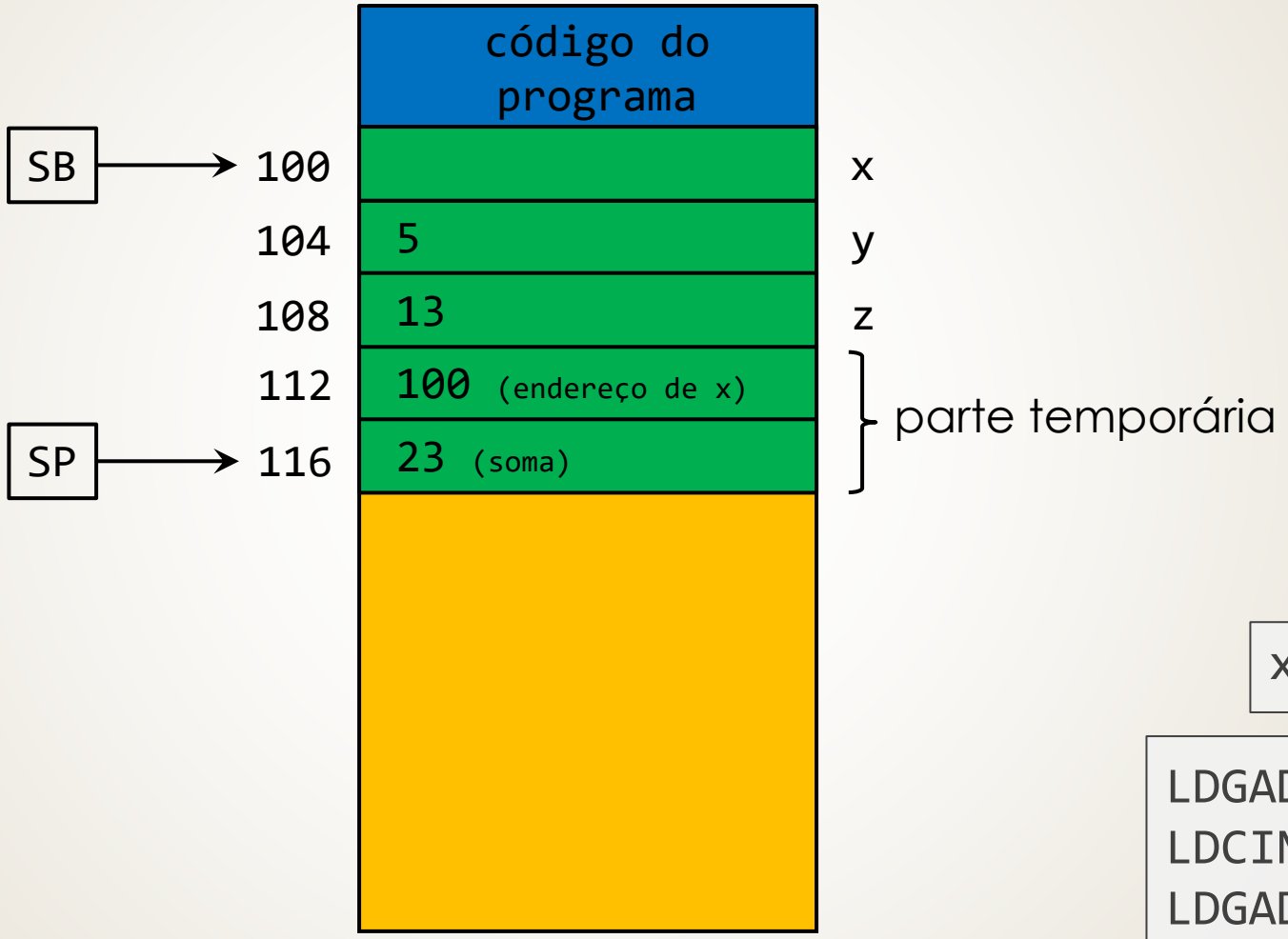
```
LDGADDR 0      LDGADDR 8
LDCINT 2       LOADW
LDGADDR 4      ADD
LOADW          STOREW
MUL
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários
Exemplo

Após a execução de **ADD**




```
x := 23;
```

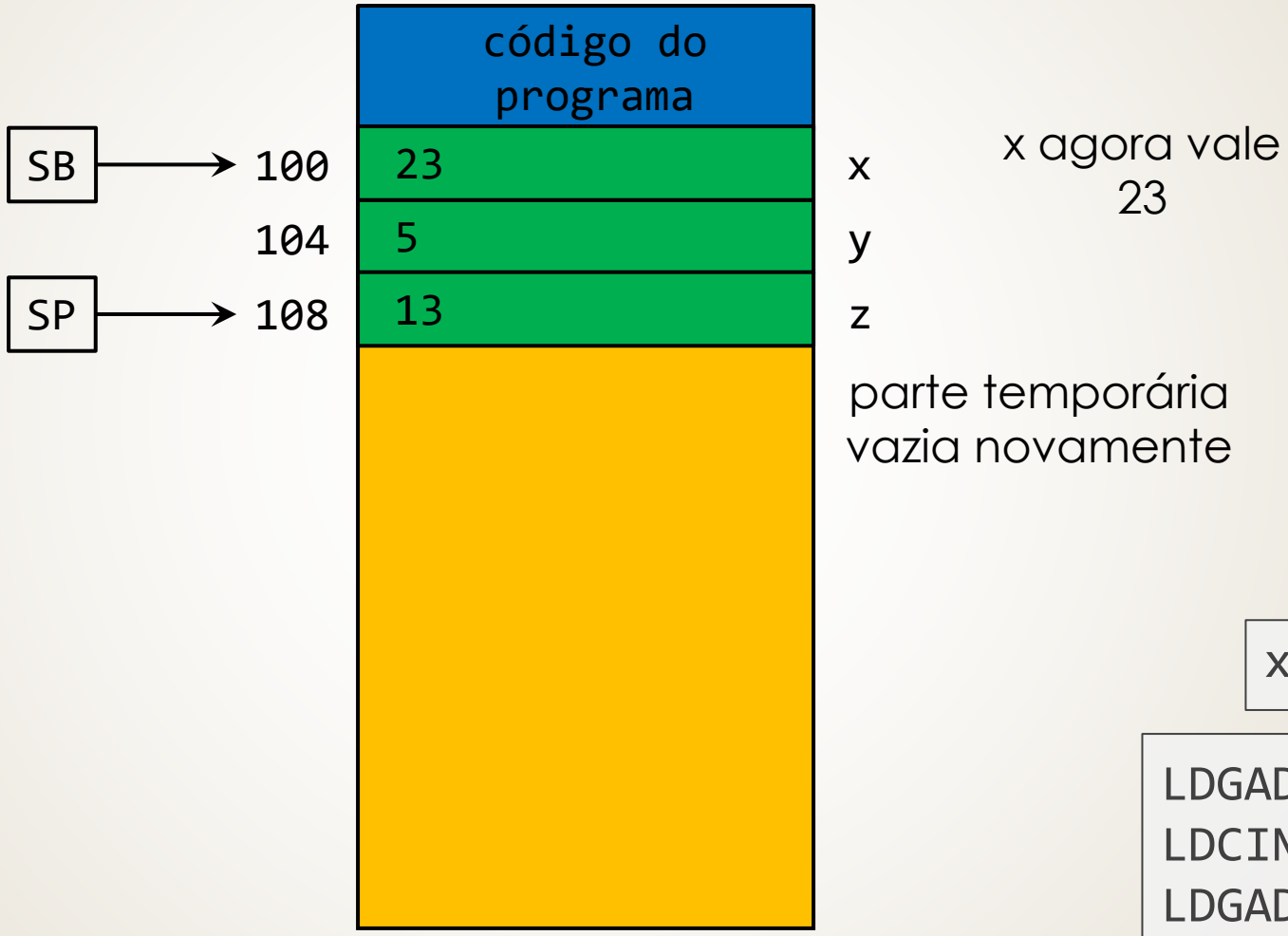
```
LDGADDR 0      LDGADDR 8
LDCINT 2       LOADW
LDGADDR 4      ADD
LOADW          STOREW
MUL
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Usando a Pilha para Manter Valores Temporários Exemplo

Após a execução de `STOREW`




```
x := 23;
```

```
LDGADDR 0      LDGADDR 8
LDCINT 2       LOADW
LDGADDR 4      ADD
LOADW          STOREW
MUL
```

***Load/Store* Opcodes**
LDCINT: *load constant integer*
LDGADDR: *load global address*
LOADW: *load word*
STOREW: *store word*

***Arithmetic* Opcodes**
ADD: *add*
MUL: *multiply*

Bibliografia

MOORE JR., J. I. **Introduction to Compiler Design: an Object Oriented Approach Using Java**. 2. ed. [s.l.]:SoftMoore Consulting, 2020. 284 p.

AHO, A. V.; LAM, M. S.; SETHI, R. ULLMAN, J. D. **Compiladores: Princípios, Técnicas e Ferramentas**. 2. ed. São Paulo: Pearson, 2008. 634 p.

COOPER, K. D.; TORCZON, L. **Construindo Compiladores**. 2. ed. Rio de Janeiro: Campus Elsevier, 2014. 656 p.

JOSÉ NETO, J. **Introdução à Compilação**. São Paulo: Elsevier, 2016. 307 p.

SANTOS, P. R.; LANGOLOIS, T. **Compiladores: da teoria à prática**. Rio de Janeiro: LTC, 2018. 341 p.