Construção de Compiladores
Aula 4 - MEPA

Bruno Müller Junior

Departamento de Informática
UFPR

18 de Agosto de 2014

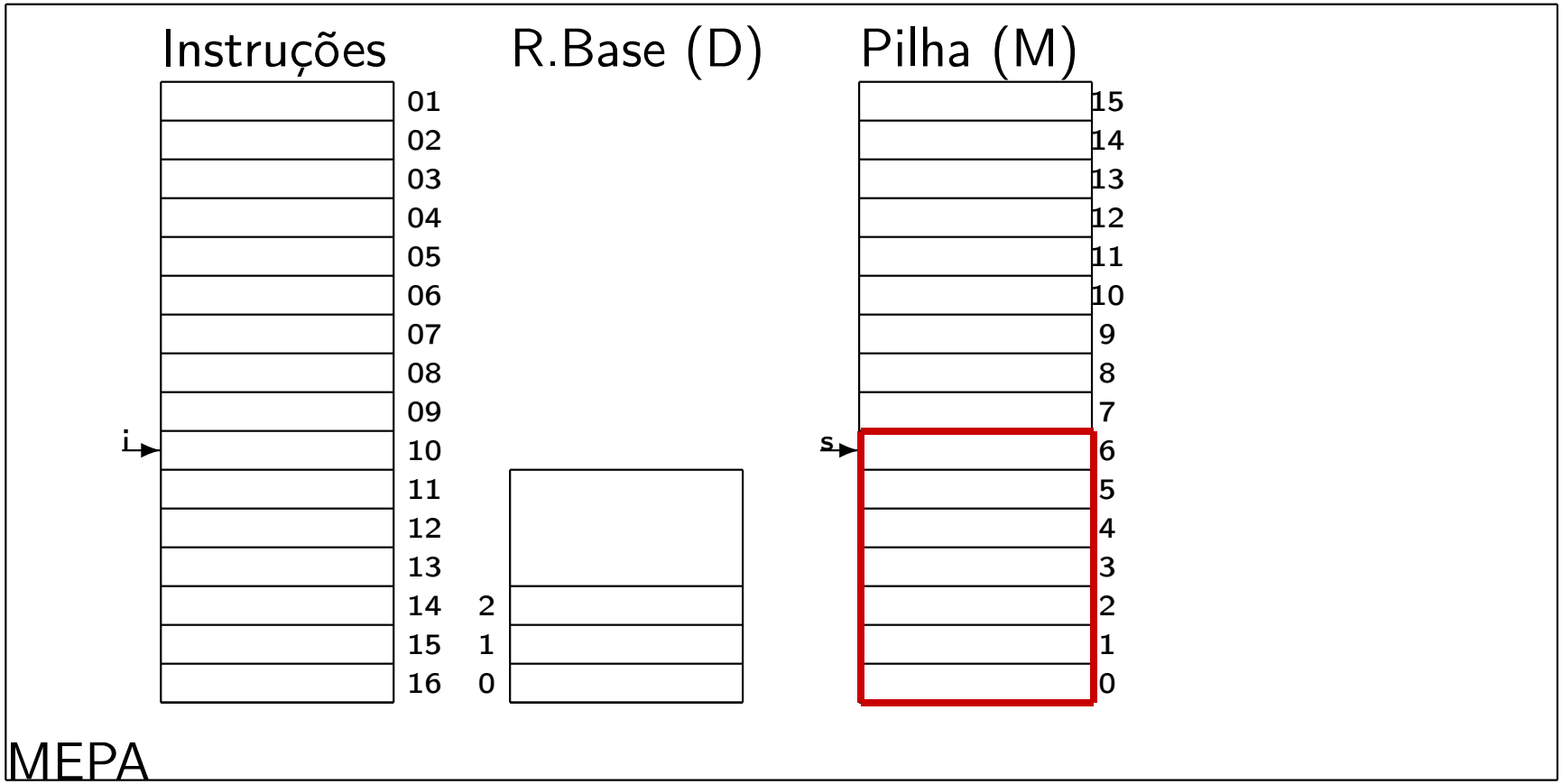
Introdução

- O bison pode gerar código assembly diretamente nos nós "executáveis", porém por vezes é complicado de entender.
- A MEPA (Máquina de Execução de Pascal) é uma linguagem intermediária que simplifica a geração de código.
- Características
  - Máquina que usa uma pilha para cálculos (como na notação posfixa: `abc+-` e demais operações.
  - Memória (M)
  - Registradores de base (D)
  - Contador de instruções (i)
  - apontador da pilha (s) (*stack pointer*).

Introdução

- Modelo Esquemático da MEPA.
- A pilha cresce para cima.
- Da posição 0 (zero) até o topo (s), a pilha contém valores válidos. De $s + 1$ para cima, são valores desconhecidos ou inválidos.

Modelo Esquemático



Instruções

- As instruções da MEPA implementam o modelo de execução.
- Todas elas são descritas com um mnemônico de quatro letras precedidas ou não de um rótulo seguido do símbolo dois pontos (:)
- Estas instruções serão apresentadas gradualmente, mas todas podem ser encontradas no livro do Tomasz (somente modelo básico).
- Exemplos:

Instrução	Ação	Significado

CRTC k	`s:=s+1;`
	`M[s]:=k`
	`i:=i+1`
		Carrega Constante

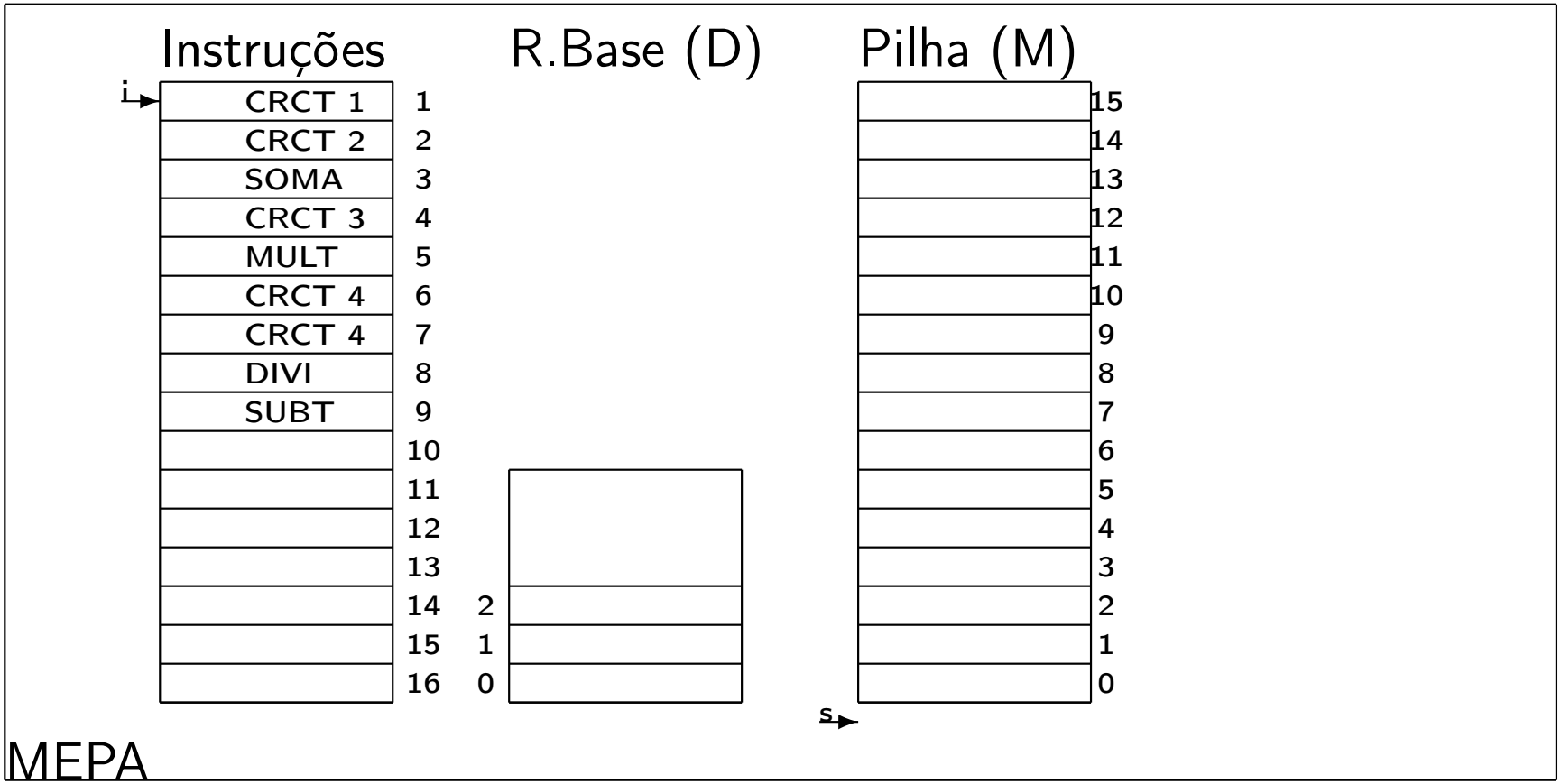
SOMA	`M[s-1]:=M[s-1]+M[s];`
	`s:=s-1;`
	`i:=i+1`
		Soma

SUBT	`M[s-1]:=M[s-1]-M[s];`
	`s:=s-1;`
	`i:=i+1`
		Subtrai

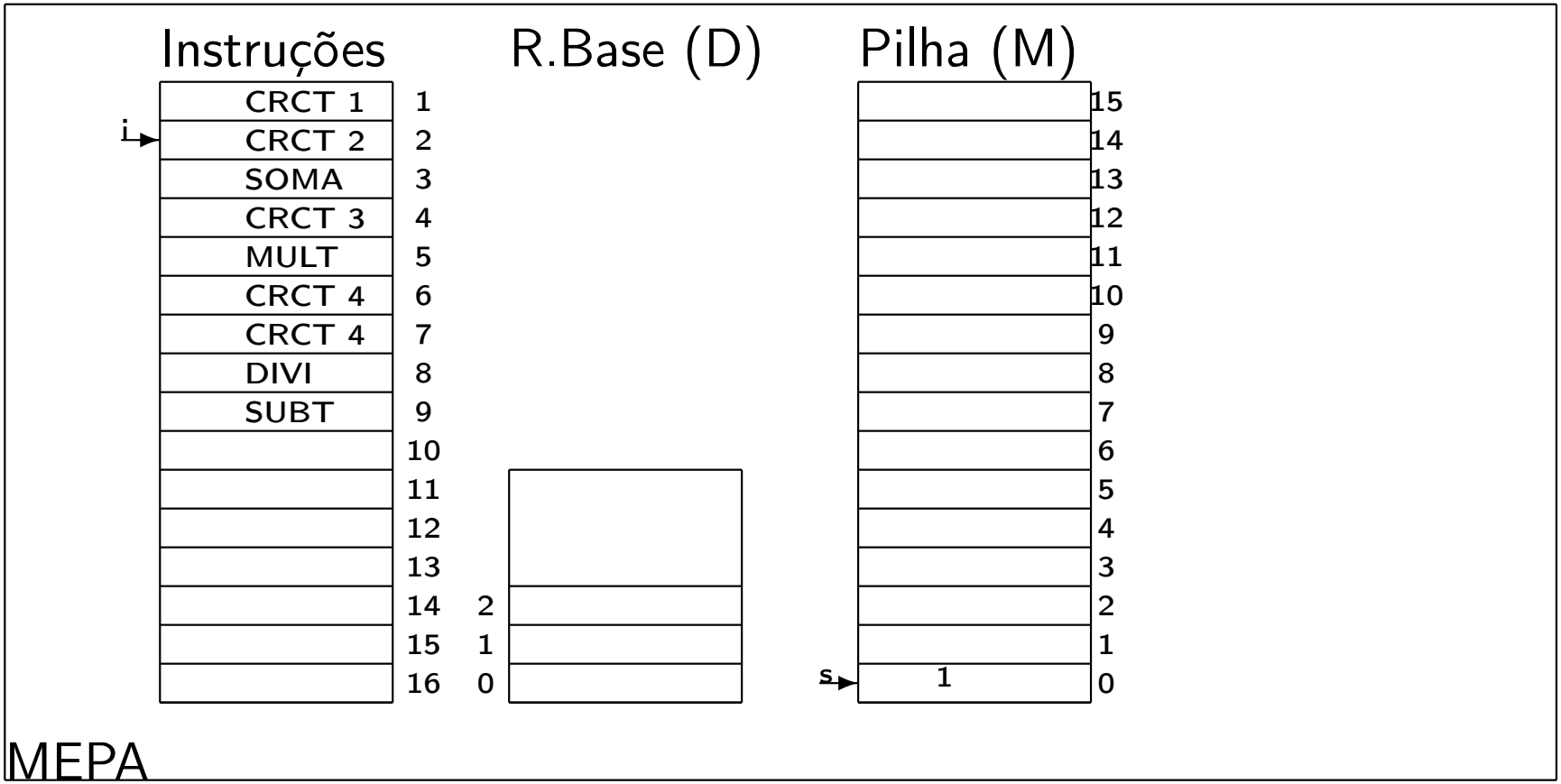
MULT	...	multiplica
DIVI	...	divide

Exemplo de Tradução	
Expressão	Código MEPA equivalente
(1+2)*3-4/4	CRCT 1 CRCT 2 SOMA CRCT 3 MULT CRCT 4 CRCT 4 DIVI SUBT

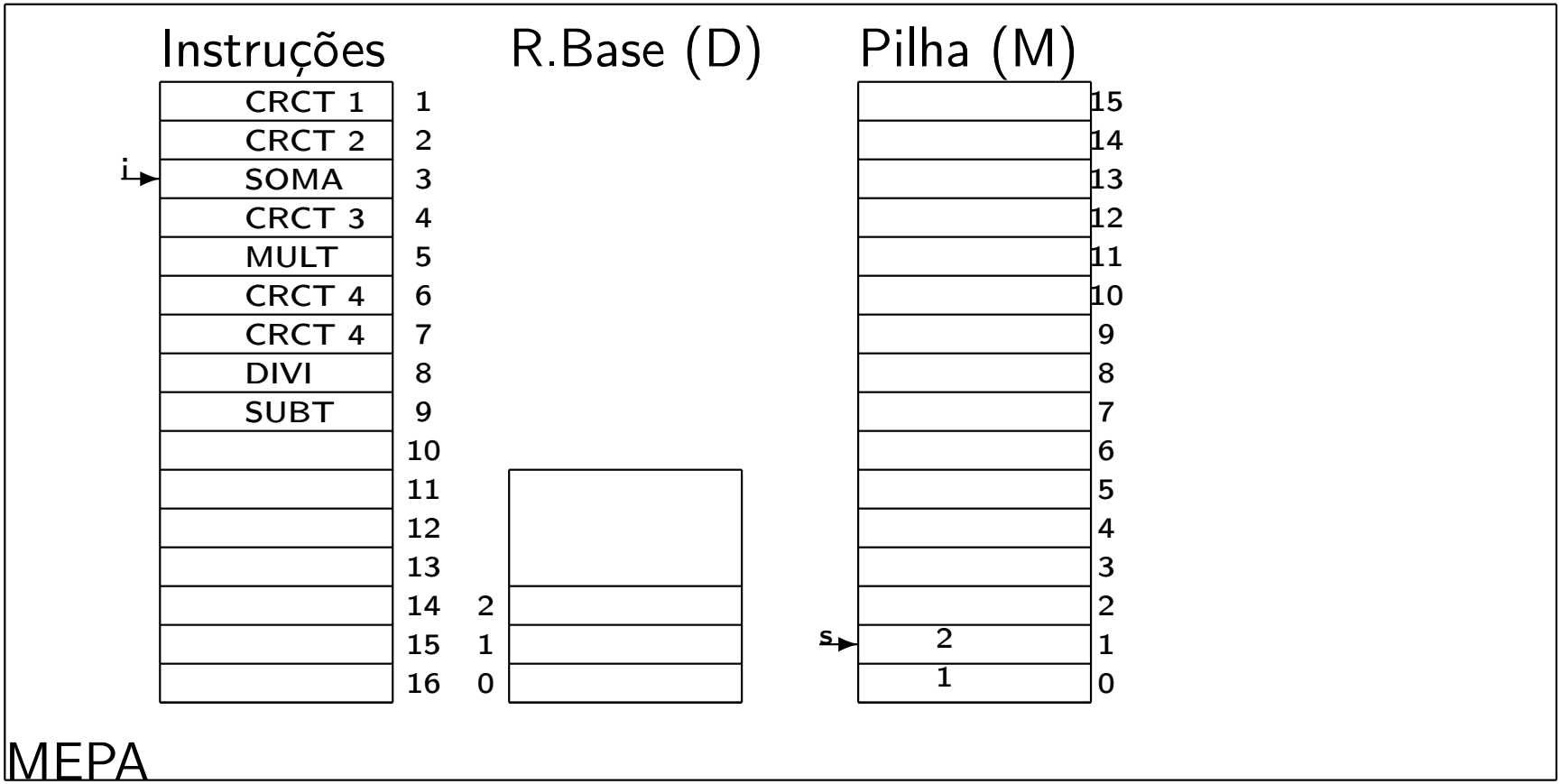
Execuçao (1)


Instruções
R.Base (D)
Pilha (M)
i
CRCT 1
CRCT 2
SOMA
CRCT 3
MULT
CRCT 4
CRCT 4
DIVI
SUBT
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
2
1
0
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
s
MEPA

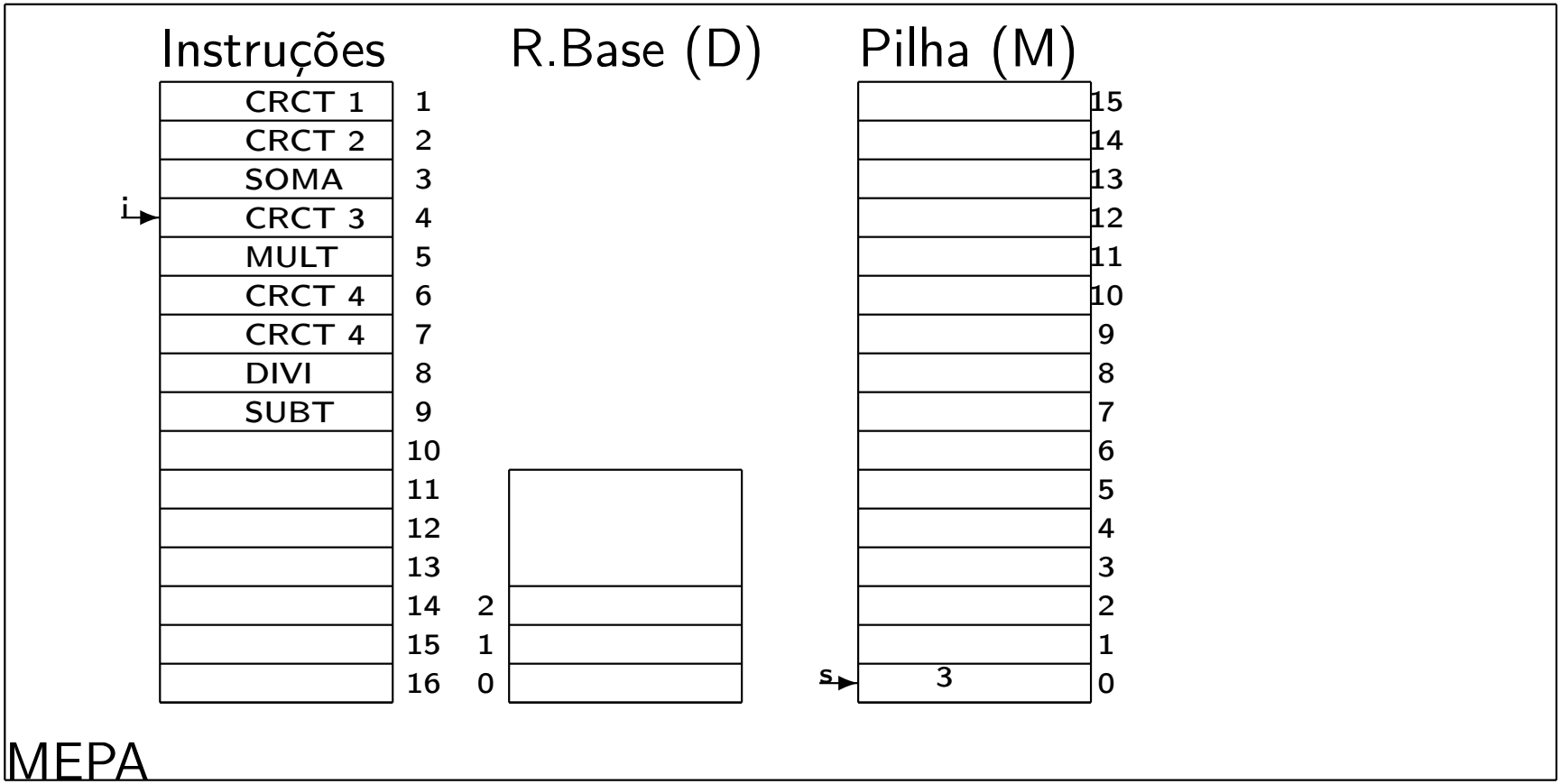
Execuçao (2)


Instruções
R.Base (D)
Pilha (M)
CRCT 1
CRCT 2
SOMA
CRCT 3
MULT
CRCT 4
CRCT 4
DIVI
SUBT
i
s
1
MEPA

Execuçao (3)


Instruções
R.Base (D)
Pilha (M)
CRCT 1
CRCT 2
SOMA
CRCT 3
MULT
CRCT 4
CRCT 4
DIVI
SUBT
i
s
MEPA

Execuçao (4)


Instruções
R.Base (D)
Pilha (M)
CRCT 1
CRCT 2
SOMA
CRCT 3
MULT
CRCT 4
CRCT 4
DIVI
SUBT
i
s
3
MEPA

- Objetivo: Explicar a tradução Pascal $\Rightarrow$ MEPA
- A cada aula, serão explicados os seguintes tópicos:
  - Construção a ser implementada;
  - Esquema de tradução;
  - Regras gramaticais a serem usadas;
  - Adaptação para bison;
- Como cada aula é construída sobre as construções já conhecidas nas aulas anteriores, cada aluno **DEVE** implementar cada aula o quanto antes.
- Regras gramaticais: apêndice B do livro do Tomasz (notação a ser corrigida).

- Regra 1:

  program <identificador> (<lista de identificadores>);
  <bloco>.

- O que está escrito em vermelho são tokens;
- O que está entre "<" e ">" são regras;
- Considere que a regra <bloco> está vazia. Assim, uma entrada válida para esta regra é:

  program exemplo (input, output); .

- O compilador é um programa que verifica se uma entrada está de acordo com as regras sintáticas e semânticas de uma gramática e se estiver gera código "executável" (que nesta disciplina é o código MEPA).
- As regras sintáticas do bison ajudam nesta geração de código. Um exemplo é quando usar as instruções `INPP` e `PARA`.

Instrução	Ação	Significado

INPP	`s:=-1;`
	`D[0]:=0;`
	`i:=i+1`
		Inicia Programa Pascal

PARA		Finaliza

- Ao encontrar o token `program`, o bison deve imprimir a instrução `INPP`, e ao encontrar o ponto final, imprimir `PARA`.

- Uma sugestão de implementação é:

```
programa :{
            geraCodigo (NULL, "INPP");
          }
            PROGRAM IDENT
            ABRE_PARENTESES lista_idents FECHA_PARENTESES PONTO_E_VIRGULA
            bloco PONTO
          {
            finalizaCompilador();
            geraCodigo (NULL, "PARA");
          }
;
```

- onde a função `geraCodigo(rotulo, comando MEPA)` tem dois parâmetros: o rótulo e o comando MEPA a ser gerado no arquivo de saída.

- O apêndice 1 contém todas as regras que usaremos na disciplina (não usaremos as que tem um (*) ao final.
- Porém, há um problema: elas estão num formato não aceito pelo bison, em especial:
- { e } para indicar repetição;
- [e] para indicar opção;
- A seguir, será explicado como converter as regras que contém estes símbolos em regras equivalentes, apropriadas ao bison.

- Exemplo de regra que indica repetição

```
10. <lista de identificadores> ::= <identificador> {, <identificador>}
```

- Esta regra reconhece como válidas entradas como `a, g1,b`.
- Formato geral: $A ::= \beta\{, \alpha\}$
- Equivalente em bison: $A ::= A\alpha|\beta$
- ou seja:

```
10. <lista de identificadores> ::= <lista de identificadores> <identificador>
                                 | <identificador>
```

- Exemplo de regra opcional:

```
2. <bloco> ::= [<parte de declaração de rótulos>] ...
```

- <parte de declaração de rótulos> é opcional.
- Formato A ::= [B]
- Como B é opcional, a sua regra deve ser acrescida de um "ou" vazio, ou seja:

```
B ::= --copiar regras já existentes --
    | // acrescentando a regra vazia
```

- Isto funciona porque nestes no Pascal simplificado, o primeiro token de B indicará se a regra deve ser usada ou não.

- Baixe o arquivo `Projeto.tar.bz2`
- Este arquivo contém o início do compilador (arquivos flex, bison, header (.h) e subrotinas C (.c)).
- Compile (`make`) e execute o compilador com entradas simples.
- Acrescente a geração de código para `INPP` e `PARA`.