MEPA Interpreter

(version 5.0)

Tomasz Kowaltowski
INSTITUTE OF COMPUTING
UNICAMP

June 2016

Introduction

MEPA (*Máquina de Execução para PAscal*, in Portuguese) is a virtual machine developed in order to simplify code generation for a PASCAL subset (*Simple Pascal*) used in a course on implementation of programming languages. The subset and the rationale for MEPA development are described in the reference at the end of this document and in some other course handouts.

There are many ways of implementing a virtual machine like MEPA. A natural and efficient way is through macro-assembly: each MEPA instruction can be defined by a macro-instruction (sequence of machine instructions) of the target computer. This approach would result in a quite realistic code generation process. Another approach, particularly convenient for teaching purposes, is an implementation through a *simulator* or *interpreter*, with the inclusion of debugging facilities.

This document describes a MEPA interpreter implemented in PYTHON 3, running under most operating systems.

Program usage

The interpreter is invoked through a command line with the following syntax:

```
[python3] mepa.py
       [-h | -help] [-c | --copyright][-s | --silent]
       [--messfile filename (stderr)][--programsize cardinal (500)]
       [--stacksize cardinal (500)][--displaysize cardinal (10)]
       [--limit cardinal (10000)][--infile filename (stdin)]
       [--outfile filename (stdout)][--progfile filename (stdin)]
       [--debug] [--nocheck] [--step]
```

All parameters are optional, and their order is irrelevant. Each parameter is followed by its default value between parentheses. The default value of boolean parameters `--debug`, `--nocheck`, `--silent` and `--step` is *false*.

The meaning of the parameters is:

`--help:`	Print a help message with a short description.
`--copyright:`	Print a copyright message.
`--silent:`	Disable printing of any messages except errors.
`--messfile:`	Name of the file for messages and errors.
`--programsize:`	Maxumum number of instructions of a MEPA program.
`--stacksize:`	Maximum execution stack size.
`--displaysize:`	Maximum display size (number of base registers).
`--limit:`	Maximum number of instructions to be executed.
`--infile:`	Input data file name (see the instruction `READ`).
`--outfile:`	Output file name (see the instruction `PRNT`).
`--progfile:`	MEPA program file name.
`--debug:`	Turn on debugging mode.
`--nocheck:`	Turn off consistency checks for stack pointers and function execution levels; should be turned on whenever `--debug` is used.
`--step:`	Turn on step-by-step execution.

MEPA architecture

- `P`: program memory
- `M`: stack memory
- `D`: base registers (display)
- `i`: current instruction pointer (in P)
- `s`: stack pointer (in M)

MEPA instruction repertoir

`LDCT k` (Load constant)
`s ← s+1; M[s] ← k`

`LDVL m,n` (Load value)
`s ← s+1; M[s] ← M[D[m]+n]`

`LADR m,n` (Load address)
`s ← s+1; M[s] ← D[m]+n`

`STVL m,n` (Store value)
`M[D[m]+n] ← M[s]; s ← s-1`

`LVLI m,n` (Load value indirectly)
`s ← s+1; M[s] ← M[M[D[m]+n]]`

`STVI m,n` (Store value indirectly)
`M[M[D[m]+n]] ← M[s]; s ← s-1`

`ADDD` (Add)
`M[s-1] ← M[s-1]+M[s]; s ← s-1`

`SUBT` (Subtract)
`M[s-1] ← M[s-1]-M[s]; s ← s-1`

`MULT` (Multiply)
`M[s-1] ← M[s-1]*M[s]; s ← s-1`

`DIVI` (Divide)
`M[s-1] ← M[s-1]` **div** `M[s]; s ← s-1`

`NEGT` (Negate)
`M[s] ← -M[s]`

`LAND` (Logical and)
`M[s-1] ← M[s-1]` **and** `M[s]; s ← s-1`

`LORR` (Logical or)
`M[s-1] ← M[s-1]` **or** `M[s]; s ← s-1`

`LNOT` (Logical not)
`M[s] ←` **not** `M[s]`

`LESS` (Less)
`M[s-1] ← M[s-1]<M[s]; s ← s-1`

`GRTR` (Greater)
`M[s-1] ← M[s-1]>M[s]; s ← s-1`

`EQUA` (Equal)
`M[s-1] ← M[s-1]=M[s]; s ← s-1`

`DIFF` (Different)
`M[s-1] ← M[s-1]<>M[s]; s ← s-1`

`LEQU` (Less or equal)
`M[s-1] ← M[s-1]<=M[s]; s ← s-1`

`GEQU` (Greater or equal)
`M[s-1] ← M[s-1]>=M[s]; s ← s-1`

`JUMP p` (Jump)
`i ← p`

`JMPF p` (Jump on false)
if not `M[s]` **then** `i ← p` **else** `i ← i+1; s ← s-1`

`NOOP` (No operation)

`STOP` (Stop)
`"stop execution"`

```
READ        (Read)
            s ← s+1; M[s] ← "next input value"

PRNT        (Print)
            "print M[s]"; s ← s-1

ALOC  n     (Allocate memory)
            s ← s+n

DLOC  n     (Deallocate memory)
            s ← s-n

MAIN        (Start program execution)
            s ← -1; D[0] ← 0

ENLB  j,n   (Enter label)
            s ← D[j]+n-1

LGAD  p,k   (Load generalized address)
            M[s+1] ← p; M[s+2] ← D[k];
            M[s+3] ← k; s ← s+3

CFUN  p,k   (Call function)
            M[s+1] ← i+1; M[s+2] ← D[k];
            M[s+3] ← k; s ← s+3; i ← p

CPFN  m,n,k (Call parameter function)
            M[s+1] ← i+1; M[s+2] ← D[k];
            M[s+3] ← k; s ← s+3; i ← M[D[m]+n];
            temp ← M[D[m]+n+2]; D[temp] ← M[D[m]+n+1];
            while temp>=2 do
              {D[temp-1] ← M[D[temp]-1]; temp ← temp-1}

ENFN  k     (Enter function)
            s ← s+1; M[s] ← D[k-1]; D[k] ← s+1

RTRN  n     (Return from function)
            temp ← M[s-1]; D[temp] ← M[s-2];
            i ← M[s-3]; s ← s-(n+4);
            while temp>=2 do
              {D[temp-1] ← M[D[temp]-1]; temp ← temp-1}
```

Additional instructions

Debugging

The interpreter recognizes three additional instructions which help debugging MEPA programs; they can be inserted at any place in the program:

`DUMP`	Dumps the description of the machine state at the point of execution of this instruction.
`STEP 0`	Turns off step-by-step mode.
`STEP 1`	Turns on step-by-step mode.
`DBUG 0`	Turns off debugging mode.
`DBUG 1`	Turns on debugging mode.

Whenever MEPA is in the debugging mode (including its activation by the command option '`--debug`'), the execution of each instruction will output the following values:

- address of the following instruction (register `i`)
- stack top index (register `s`)
- the following instruction to be executed
- relevant values for the instruction

Values which represent integers stored in memory are accompanied (between parentheses) by information about their types following the convention:

- 0: integer operand
- 1: lexical level
- 2: stack address (pointer)
- 3: program address

Indexing

The interpreter recognizes four instructions which simplify code generation for arrays. Some of these instructions are superfluous and could be replaced by sequences of regular MEPA instructions, except for the fact that it would require usage of the `--nocheck` option.

```
INDX  k     (Index)
            M[s-1]← M[s-1]+M[s]*k
            s ← s-1

CONT        (Load contents)
            M[s] ← M[M[s]]

LDMV  k     (Load multiple value)
            temp1 ← M[s]
            for temp2 from 0 to k-1 do
               M[s+temp2] ← M[temp1+temp2]
            s ← s+k-1

STMV  k     (Store multiple value)
            for temp from 0 to k-1 do
               M[M[s-k]+temp] ← M[s-k+temp+1]
            s ← s-k-1
```

Portuguese version

The interpreter can be executed with programs which use Portuguese instruction codes as described in the reference text with the command:

```
[python3] mepa_pt.py  ...
```

The following are alphabetically ordered equivalence tables for the instruction codes.

English to Portuguese:

ADDD	SOMA
ALOC	AMEM
CONT	CONT
CPFN	CHPP
CFUN	CHPR
DBUG	DBUG
DIFF	CMDG
DIVI	DIVI
DLOC	DMEM
DUMP	DUMP
ENLB	ENRT

ENFN	ENPR
EQUA	CMIG
GEQU	CMAG
GRTR	CMMA
INDX	INDX
JMPF	DSVF
JUMP	DSVS
LADR	CREN
LAND	CONJ
LDCT	CRCT
LEQU	CMEG

LDMV	CRVM
LDVL	CRVL
LESS	CMME
LGAD	CREG
LNOT	NEGA
LORR	DISJ
LVLI	CRVI
MAIN	INPP
MULT	MULT
NEGT	INVR
NOOP	NADA

PRNT	IMPR
READ	LEIT
RTRN	RTPR
STEP	STEP
STMV	ARVM
STOP	PARA
STVI	ARMI
STVL	ARMZ
SUBT	SUBT
END	FIM

Portuguese to English:

AMEM	ALOC
ARMI	STVI
ARMZ	STVL
ARVM	STMV
CHPP	CPFN
CHPR	CFUN
CMAG	GEQU
CMDG	DIFF
CMEG	LEQU
CMIG	EQUA
CMMA	GRTR

CMME	LESS
CONJ	LAND
CONT	CONT
CRCT	LDCT
CREG	LGAD
CREN	LADR
CRVI	LVLI
CRVL	LDVL
CRVM	LDMV
DBUG	DBUG
DISJ	LORR

DIVI	DIVI
DMEM	DLOC
DSVF	JMPF
DSVS	JUMP
DUMP	DUMP
ENPR	ENFN
ENRT	ENLB
IMPR	PRNT
INDX	INDX
INPP	MAIN
INVR	NEGT

LEIT	READ
MULT	MULT
NADA	NOOP
NEGA	LNOT
PARA	STOP
RTPR	RTRN
SOMA	ADDD
STEP	STEP
SUBT	SUBT
FIM	END

Remarks

1. Program lines starting with '`;`' are ignored.
2. Any text following, after spaces, the instruction code and its eventual arguments is ignored.
3. Every label definition must start with a letter and be followed by the caracter '`:`'.
4. The last line of a MEPA program must be the pseudo-instruction `END` marking the *physical* end of the program.

Reference

T. Kowaltowski, *Implementação de Linguagens de Programação*, Guanabara Dois, 1983 (in Portuguese).

Example

```
; Example of a MEPA program
; Several instructions are followed by comments
; This program computes squares of numbers 1 to k
; The value of k is read from the input file
;
               MAIN          program example
               ALOC     2    var i,k:  integer
               READ
               STVL     0,1  read(k)
               LDCT     1
               STVL     0,0  i:=1
        L1:    NOOP          while
               LDVL     0,0
               LDVL     0,1
               LEQU          i<=k
               JMPF     L2   do
               LDVL     0,0
               LDVL     0,0
               MULT
               PRNT          write(i*i)
               LDVL     0,0
               LDCT     1
               ADDD
               STVL     0,0  i:=i+1
               JUMP     L1
        L2:    NOOP
               DLOC     2
               DUMP
               STOP
               END
```

The execution of this program with input value 5 would produce the following output:

```
Mepa Interpreter version 5.0
============================


Options:
--------
    messfile:  stderr
      infile:  ex.in
     outfile:  stdout
    progfile:  ex.mep
 programsize:  500
```

```
   stacksize:  500
 displaysize:  10
       limit:  10000
        help:  False
   copyright:  False
       debug:  False
     nocheck:  False
      silent:  False
        step:  False

1
4
9
16
25


Dump
====
i= 23, s= -1

Display
-------
 0:     0

Memory
------
 0:     6 (0)
 1:     5 (0)
 2:     0 (0)
 3:     5 (0)

Labels
------
L2   :  20
L1   :  6

End dump
========

Executed 85 instructions
```