

**Component Based MMIX Simulator using Multiple  
Programming Paradigms**

A dissertation submitted in partial fulfilment of the requirements for the  
MSc in Advanced Computing Technologies

by Stephen Edmans

Department of Computer Science and Information Systems  
Birkbeck College, University of London

September 2015

This report is substantially the result of my own work except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service. I have read and understood the sections on plagiarism in the Programme Handbook and the College website.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

# Abstract

There are currently over 2,500<sup>1</sup> different programming languages, with more created every year. These programming languages can get grouped together in numerous different ways. This makes the decision of what language to use when starting a new project extremely difficult.

There are several ways in which we can reach this decision; choose the language that your team knows best; choose the language that makes the most sense to implement the critical part of your system; choose a simple general purpose language; choose a language that has got an active community. There is no acknowledged best approach to take.

Another approach would be to split your application up into separate components and using a different programming language for each component. This allows us choose the most appropriate programming language for each component.

The purpose of this project is to examine this approach. The application that we will create will be a simulator for an artificial machine language. The artificial machine language that we will use is called MMIX, it was developed by Donald Knuth as part of his seminal work *The Art of Computer Programming*[Knu11].

---

<sup>1</sup>From the language list[Kin]

# Contents

<b>Abstract</b>	<b>2</b>
<b>Contents</b>	<b>3</b>
<b>Acknowledgements</b>	<b>6</b>
<b>1 Introduction</b>	<b>7</b>
<b>2 Assembler</b>	<b>8</b>
2.1 Introduction . . . . .	8
2.2 Executable . . . . .	8
2.3 Lexer . . . . .	8
2.4 Parser . . . . .	8
2.5 Code Generation . . . . .	8
2.5.1 Symbol Table . . . . .	8
2.5.2 Automatically Assigned Registers . . . . .	8
2.5.3 Local Symbols . . . . .	8
2.5.4 Handling Operands . . . . .	8
2.5.5 Assembler Directives . . . . .	8
2.5.6 Generating the Output . . . . .	8
2.6 Component Testing . . . . .	8
<b>3 Graphical User Interface</b>	<b>9</b>
3.1 Introduction . . . . .	9
3.2 User Interface Design . . . . .	10
3.2.1 Console Panel . . . . .	10
3.2.2 Controls Panel . . . . .	10
3.2.3 Main State Panel . . . . .	10
3.2.4 Memory Panel . . . . .	10
3.2.5 Registers Panel . . . . .	10
3.3 Asynchronous UI Programming with Actors . . . . .	10
3.4 Communication . . . . .	10
3.5 Component Testing . . . . .	10

<b>4</b>	<b>Virtual Machine</b>	<b>11</b>
4.1	Introduction . . . . .	11
4.2	Memory . . . . .	11
4.3	Registers . . . . .	12
4.4	Central Processing Unit . . . . .	14
4.5	Calling the Operating System . . . . .	14
4.6	Communication . . . . .	14
4.7	Component Testing . . . . .	14
<b>5</b>	<b>Simulator Application</b>	<b>15</b>
5.1	Introduction . . . . .	15
5.2	Integration Testing . . . . .	15
5.2.1	Generate Prime Numbers Sample Application . . . . .	15
	<b>Conclusion</b>	<b>16</b>
	<b>References</b>	<b>17</b>
	<b>Appendices</b>	
<b>A</b>	<b>Source Code</b>	<b>18</b>
A.1	Assembler . . . . .	18
A.2	Graphical User Interface . . . . .	18
A.3	Virtual Machine . . . . .	18
<b>B</b>	<b>Intermediate Assembler Representations</b>	<b>19</b>
B.1	Definitions . . . . .	19
B.2	Test Application . . . . .	19
B.2.1	Sample Test MMIXAL Code . . . . .	19
B.2.2	Parsed Sample File . . . . .	21

# List of Figures

3.1	GUI Sample Screenshot . . . . .	9
4.1	Special Registers . . . . .	13

# Acknowledgements

# Chapter 1

## Introduction

All definition of an MMIX computer come directly from either [Knu11] or [Knu]



## Chapter 2

# Assembler

### 2.1 Introduction

### 2.2 Executable

### 2.3 Lexer

### 2.4 Parser

### 2.5 Code Generation

#### 2.5.1 Symbol Table

#### 2.5.2 Automatically Assigned Registers

#### 2.5.3 Local Symbols

#### 2.5.4 Handling Operands

#### 2.5.5 Assembler Directives

#### 2.5.6 Generating the Output

### 2.6 Component Testing

## Chapter 3

# Graphical User Interface

### 3.1 Introduction

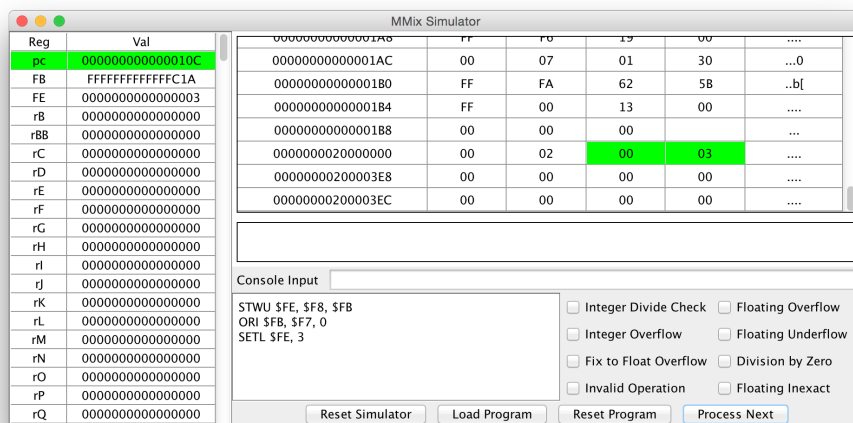


Figure 3.1: GUI Sample Screenshot

## **3.2 User Interface Design**

### **3.2.1 Console Panel**

### **3.2.2 Controls Panel**

### **3.2.3 Main State Panel**

### **3.2.4 Memory Panel**

### **3.2.5 Registers Panel**

## **3.3 Asynchronous User Interface Programming with Actors**

## **3.4 Communication**

## **3.5 Component Testing**

## Chapter 4

# Virtual Machine

### 4.1 Introduction

Architecture of a computer CPU, ALU, Memory, Secondary Storage, IO Devices

The virtual machine we are developing will only consist of a CPU and memory.

The way that memory is organised can be considered a hierarchy, to quote Aho et al[ALSU06]

A memory hierarchy consists of several levels of storage with different speeds and sizes, with the levels closest to the processor being the fastest but smallest... Memory hierarchies are found in all machines. A processor usually has a small number of registers consisting of hundreds of bytes, several levels of caches containing kilobytes to megabytes, physical memory containing megabytes to gigabytes, and finally secondary storage that contains gigabytes and beyond.

For this project we will only be considering the physical memory and the registers.

### 4.2 Memory

Wyde

$$M_2[0] = M_2[1] = M[0]M[1]$$

Tetra

$$M_4[4k] = M_4[4k + 1] = \dots = M_4[4k + 3] = M[4k]M[4k + 1]\dots M[4k + 3]$$

Octa

$$M_8[8k] = M_8[8k + 1] = \dots = M_8[8k + 7] = M[8k]M[8k + 1]\dots M[8k + 7]$$

### 4.3 Registers

An MMIX computer contains two distinct types of registers, 256 general purpose registers and 32 special purpose registers. A complete list of the special registers can be found in Figure 4.1

rA Arithmetic Status Register

least significant byte contains eight event bits. DVWIOUZX

Register	Description
D	Integer Divide Check
V	Integer Overflow
W	Float-to-Fix Overflow
I	Invalid Operation
O	Floating Overflow
U	Floating Underflow
Z	Division by Zero
X	Floating Inexact

The next least significant byte contains eight “enable” bits with the same name DVWIOUZX and the same meanings.

When an exceptional condition occurs, there are two cases: If the corresponding enable bit is 0, the corresponding event bit is set to 1; but if the corresponding enable bit is 1, MMIX interrupts its current instruction stream and execute a special “exception handler”. Thus, the event bits record exceptions that have not been “tripped”.

This leaves six high order bytes. At present, only two of those 48 bits are defined. The two bits corresponding to  $2^{17}$  and  $2^{16}$  in rA specify a rounding mode, as follows: -

00	Round to the nearest
01	Round off
10	Round up
11	Round down

Identifier	Description
rA	Arithmetic Status Register
rB	Bootstrap Register
rC	Continuation Register
rD	Dividend Register
rE	Epsilon Register
rF	Failure Location Register
rG	Global Threshold Register
rH	Himult Register
rI	Interval Counter
rJ	Return-Jump Register
rK	Interrupt Mask Register
rL	Local Threshold Register
rM	Multiplex Mask Register
rN	Serial Number
rO	Register Stack Offset
rP	Prediction Register
rQ	Interrupt Request Register
rR	Remainder Register
rS	Register Stack Pointer
rT	Trap Address Register
rU	Usage Counter
rV	Virtual Translation Register
rW	Where Interrupted Register
rX	Execution Register
rY	Y Operand
rZ	Z Operand
rBB	Bootstrap Register
rTT	Dynamic Trap Address Register
rWW	Where Interrupted Register
rXX	Execution Register
rYY	Y Operand
rZZ	Z Operand

Figure 4.1: Special Registers

**4.4 Central Processing Unit**

**4.5 Calling the Operating System**

**4.6 Communication**

**4.7 Component Testing**

## Chapter 5

# Simulator Application

### 5.1 Introduction

### 5.2 Integration Testing

#### 5.2.1 Generate Prime Numbers Sample Application



# Conclusion

# References

- [akk] Akka toolkit. <<http://akka.io/>>[Access 24 August 2015].
- [ALSU06] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
- [Kin] Bill Kinnersley. The language list. <<http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm>>[Access 7 September 2015].
- [Knu] D.E. Knuth. The art of computer programming fascicle 1 mmix [e-book]. Stanford University: Addison Wesley Available through: Stanford University <<http://www-cs-faculty.stanford.edu/~uno/fasc1.ps.gz>>[Access 7 April 2013].
- [Knu90] D.E. Knuth. *MMIXware A RISC Computer for the Third Millennium*. Springer, 1990.
- [Knu11] D.E. Knuth. *The Art of Computer Programming*, volume 1-4a. 1st ed. Addison Wesley, 2011.
- [Ruc12] Martin Ruckert. Mmix quick reference card. <<http://mmix.cs.hm.edu/doc/mmix-refcard-a4.pdf>>[Access 24 August 2015], 2012.

## Appendix A

# Source Code

A.1 Assembler

A.2 Graphical User Interface

A.3 Virtual Machine

## Appendix B

# Intermediate Assembler Representations

### B.1 Definitions

### B.2 Test Application

#### B.2.1 Sample Test MMIXAL Code

The sample mmixal application I am using to test the system is taken from Fascile 1[Knu]. The complete code listing is

```

L      IS      500
t      IS      $255
n      GREG    0
q      GREG    0
r      GREG    0
jj     GREG    0
kk     GREG    0
pk     GREG    0
mm     IS      kk
      LOC      Data_Segment
PRIME1 WYDE    2
      LOC      PRIME1+2*L
ptop   GREG    @
j0     GREG    PRIME1+2-@
BUF    OCTA    0
      LOC      #100
Main   GREG    @
      SET      n,3
      SET      jj,j0
2H     STWU    n,ptop,jj
      INCL     jj,2
3H     BZ      jj,2F
4H     INCL     n,2
5H     SET      kk,j0
6H     LDWU    pk,ptop,kk
      DIV      q,n,pk
      GET      r,rR
      BZ      r,4B
7H     CMP     t,q,pk
      BNP     t,2B
8H     INCL     kk,2
      JMP     6B
      GREG    @
Title  BYTE    "First Five Hundred Primes"
NewLn  BYTE    #a,0
Blanks BYTE    " ",0
2H     LDA     t,Title
      TRAP    0,Fputs,StdOut
      NEG     mm,2
3H     ADD     mm,mm,j0
      LDA     t,Blanks
      TRAP    0,Fputs,StdOut
2H     LDWU    pk,ptop,mm
0H     GREG    #2030303030000000
      STOU    0B,BUF
      LDA     t,BUF+4
1H     DIV     pk,pk,10
      GET     r,rR
      INCL     r,'0'
      STBU    r,t,0
      SUB     t,t,1
      PBNZ    pk,1B
      LDA     t,BUF
      TRAP    0,Fputs,StdOut
      INCL     mm,2*L/10
      PBN     mm,2B
      LDA     t,NewLn
      TRAP    0,Fputs,StdOut
      CMP     t,mm,2*(L/10-1)
      PBNZ    t,3B
      TRAP    0,Halt,0

```

## B.2.2 Parsed Sample File

The final version of the parsed source code for the test application is

```
LabelledPILine {
    lppl_id = IsNumber 500, lppl_ident = Id "L", lppl_loc = 0
}
LabelledPILine {
    lppl_id = IsRegister 255, lppl_ident = Id "t", lppl_loc = 0
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\254' (ExpressionNumber
    0)), lppl_ident = Id "n", lppl_loc = 0
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\253' (ExpressionNumber
    0)), lppl_ident = Id "q", lppl_loc = 0
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\252' (ExpressionNumber
    0)), lppl_ident = Id "r", lppl_loc = 0
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\251' (ExpressionNumber
    0)), lppl_ident = Id "jj", lppl_loc = 0
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\250' (ExpressionNumber
    0)), lppl_ident = Id "kk", lppl_loc = 0
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\249' (ExpressionNumber
    0)), lppl_ident = Id "pk", lppl_loc = 0
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\248' (ExpressionNumber
    0)), lppl_ident = Id "mm", lppl_loc = 0
}
PlainPILine {
    ppl_id = LocEx (ExpressionNumber 536870912), ppl_loc =
    536870912
}
LabelledPILine {
    lppl_id = WydeArray "\STX", lppl_ident = Id "PRIME1", lppl_loc
    = 536870912
}
PlainPILine {
    ppl_id = LocEx (ExpressionNumber 536871912), ppl_loc =
    536871912
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\247' (ExpressionNumber
    536871912)), lppl_ident = Id "ptop", lppl_loc = 536871912
}
LabelledPILine {
    lppl_id = GregEx (ExpressionRegister '\246' (ExpressionNumber
    (-998))), lppl_ident = Id "j0", lppl_loc = 536871912
}
LabelledPILine {
    lppl_id = OctaArray "\NUL", lppl_ident = Id "BUF", lppl_loc =
    536871912
}
```

```

PlainPILine {
    ppl_id = LocEx (ExpressionNumber 256), ppl_loc = 256
}
LabelledPILine {
    lppl_id = Set (Expr (ExpressionIdentifier (Id "n")),Expr (
        ExpressionNumber 3)), lppl_ident = Id "Main", lppl_loc =
        256
}
PlainPILine {
    ppl_id = Set (Expr (ExpressionIdentifier (Id "jj")),Expr (
        ExpressionIdentifier (Id "j0"))), ppl_loc = 260
}
LabelledOpCodeLine {
    lpocl_code = 166, lpocl_ops = [Expr (ExpressionIdentifier (Id
        "n")),Expr (ExpressionIdentifier (Id "ptop")),Expr (
        ExpressionIdentifier (Id "jj"))], lpocl_ident = Id "??2H0"
    , lpocl_loc = 264
}
PlainOpCodeLine {
    pocl_code = 231, pocl_ops = [Expr (ExpressionIdentifier (Id "
        jj")),Expr (ExpressionNumber 2)], pocl_loc = 268
}
LabelledOpCodeLine {
    lpocl_code = 66, lpocl_ops = [Expr (ExpressionIdentifier (Id "
        jj")),Ident (Id "??2H1")], lpocl_ident = Id "??3H0",
    lpocl_loc = 272
}
LabelledOpCodeLine {
    lpocl_code = 231, lpocl_ops = [Expr (ExpressionIdentifier (Id
        "n")),Expr (ExpressionNumber 2)], lpocl_ident = Id "??4H0"
    , lpocl_loc = 276
}
LabelledPILine {
    lppl_id = Set (Expr (ExpressionIdentifier (Id "kk")),Expr (
        ExpressionIdentifier (Id "j0"))), lppl_ident = Id "??5H0",
    lppl_loc = 280
}
LabelledOpCodeLine {
    lpocl_code = 134, lpocl_ops = [Expr (ExpressionIdentifier (Id
        "pk")),Expr (ExpressionIdentifier (Id "ptop")),Expr (
        ExpressionIdentifier (Id "kk"))], lpocl_ident = Id "??6H0"
    , lpocl_loc = 284
}
PlainOpCodeLine {
    pocl_code = 28, pocl_ops = [Expr (ExpressionIdentifier (Id "q"
        )),Expr (ExpressionIdentifier (Id "n")),Expr (
        ExpressionIdentifier (Id "pk"))], pocl_loc = 288
}
PlainOpCodeLine {
    pocl_code = 254, pocl_ops = [Expr (ExpressionIdentifier (Id "r
        ")),Expr (ExpressionIdentifier (Id "rR"))], pocl_loc = 292
}
PlainOpCodeLine {
    pocl_code = 66, pocl_ops = [Expr (ExpressionIdentifier (Id "r"
        )),Ident (Id "??4H0")], pocl_loc = 296
}
LabelledOpCodeLine {
    lpocl_code = 48, lpocl_ops = [Expr (ExpressionIdentifier (Id "
        t")),Expr (ExpressionIdentifier (Id "q")),Expr (
        ExpressionIdentifier (Id "pk"))], lpocl_ident = Id "??7H0"
    , lpocl_loc = 300
}
}

```

```

PlainOpCodeLine {
    pocl_code = 76, pocl_ops = [Expr (ExpressionIdentifier (Id "t"
    )),Ident (Id "??2H0")], pocl_loc = 304
}
LabelledOpCodeLine {
    lpocl_code = 231, lpocl_ops = [Expr (ExpressionIdentifier (Id
    "kk")),Expr (ExpressionNumber 2)], lpocl_ident = Id "??8H0
    ", lpocl_loc = 308
}
PlainOpCodeLine {
    pocl_code = 240, pocl_ops = [Ident (Id "??6H0")], pocl_loc =
    312
}
PlainPILine {
    ppl_id = GregEx (ExpressionRegister '\245' ExpressionAT),
    ppl_loc = 316
}
LabelledPILine {
    lppl_id = ByteArray "First_Five_Hundred_Primes", lppl_ident =
    Id "Title", lppl_loc = 316
}
LabelledPILine {
    lppl_id = ByteArray "\n\NUL", lppl_ident = Id "NewLn",
    lppl_loc = 341
}
LabelledPILine {
    lppl_id = ByteArray "\u\u\u\NUL", lppl_ident = Id "Blanks",
    lppl_loc = 343
}
LabelledOpCodeLine {
    lpocl_code = 34, lpocl_ops = [Expr (ExpressionIdentifier (Id "
    t")),Expr (ExpressionIdentifier (Id "Title"))],
    lpocl_ident = Id "??2H1", lpocl_loc = 347
}
PlainOpCodeLine {
    pocl_code = 0, pocl_ops = [Expr (ExpressionNumber 0),
    PseudoCode 7,PseudoCode 1], pocl_loc = 351
}
PlainOpCodeLine {
    pocl_code = 52, pocl_ops = [Expr (ExpressionIdentifier (Id "mm
    ")),Expr (ExpressionNumber 2)], pocl_loc = 355
}
LabelledOpCodeLine {
    lpocl_code = 32, lpocl_ops = [Expr (ExpressionIdentifier (Id "
    mm")),Expr (ExpressionIdentifier (Id "mm")),Expr (
    ExpressionIdentifier (Id "j0"))], lpocl_ident = Id "??3H1"
    , lpocl_loc = 359
}
PlainOpCodeLine {
    pocl_code = 34, pocl_ops = [Expr (ExpressionIdentifier (Id "t"
    )),Expr (ExpressionIdentifier (Id "Blanks"))], pocl_loc =
    363
}
PlainOpCodeLine {
    pocl_code = 0, pocl_ops = [Expr (ExpressionNumber 0),
    PseudoCode 7,PseudoCode 1], pocl_loc = 367
}
LabelledOpCodeLine {
    lpocl_code = 134, lpocl_ops = [Expr (ExpressionIdentifier (Id
    "pk")),Expr (ExpressionIdentifier (Id "ptop")),Expr (
    ExpressionIdentifier (Id "mm"))], lpocl_ident = Id "??2H2"
    , lpocl_loc = 371
}

```



```

}
LabelledPILLine {
    lppl_id = GregEx (ExpressionRegister '\244' (ExpressionNumber
        2319406791617675264)), lppl_ident = Id "??0H0", lppl_loc =
        375
}
PlainOpCodeLine {
    pocl_code = 174, pocl_ops = [Ident (Id "??0H0"),Expr (
        ExpressionIdentifier (Id "BUF"))], pocl_loc = 375
}
PlainOpCodeLine {
    pocl_code = 34, pocl_ops = [Expr (ExpressionIdentifier (Id "t"
        )),Expr (ExpressionNumber 536870924)], pocl_loc = 379
}
LabelledOpCodeLine {
    lpocl_code = 28, lpocl_ops = [Expr (ExpressionIdentifier (Id "
        pk")),Expr (ExpressionIdentifier (Id "pk")),Expr (
        ExpressionNumber 10)], lpocl_ident = Id "??1H0", lpocl_loc
        = 383
}
PlainOpCodeLine {
    pocl_code = 254, pocl_ops = [Expr (ExpressionIdentifier (Id "r
        ")),Expr (ExpressionIdentifier (Id "rR"))], pocl_loc = 387
}
PlainOpCodeLine {
    pocl_code = 231, pocl_ops = [Expr (ExpressionIdentifier (Id "r
        ")),Expr (ExpressionNumber 48)], pocl_loc = 391
}
PlainOpCodeLine {
    pocl_code = 162, pocl_ops = [Expr (ExpressionIdentifier (Id "r
        ")),Expr (ExpressionIdentifier (Id "t")),Expr (
        ExpressionNumber 0)], pocl_loc = 395
}
PlainOpCodeLine {
    pocl_code = 36, pocl_ops = [Expr (ExpressionIdentifier (Id "t"
        )),Expr (ExpressionIdentifier (Id "t")),Expr (
        ExpressionNumber 1)], pocl_loc = 399
}
PlainOpCodeLine {
    pocl_code = 90, pocl_ops = [Expr (ExpressionIdentifier (Id "pk
        ")),Ident (Id "??1H0)], pocl_loc = 403
}
PlainOpCodeLine {
    pocl_code = 34, pocl_ops = [Expr (ExpressionIdentifier (Id "t"
        )),Expr (ExpressionIdentifier (Id "BUF"))], pocl_loc = 407
}
PlainOpCodeLine {
    pocl_code = 0, pocl_ops = [Expr (ExpressionNumber 0),
        PseudoCode 7,PseudoCode 1], pocl_loc = 411
}
PlainOpCodeLine {
    pocl_code = 231, pocl_ops = [Expr (ExpressionIdentifier (Id "
        mm")),Expr (ExpressionNumber 100)], pocl_loc = 415
}
PlainOpCodeLine {
    pocl_code = 80, pocl_ops = [Expr (ExpressionIdentifier (Id "mm
        ")),Ident (Id "??2H2)], pocl_loc = 419
}
PlainOpCodeLine {
    pocl_code = 34, pocl_ops = [Expr (ExpressionIdentifier (Id "t"
        )),Expr (ExpressionIdentifier (Id "NewLn"))], pocl_loc =
        423
}

```

```

}
PlainOpCodeLine {
    pocl_code = 0, pocl_ops = [Expr (ExpressionNumber 0),
        PseudoCode 7,PseudoCode 1], pocl_loc = 427
}
PlainOpCodeLine {
    pocl_code = 48, pocl_ops = [Expr (ExpressionIdentifier (Id "t"
        )),Expr (ExpressionIdentifier (Id "mm")),Expr (
        ExpressionNumber 98)], pocl_loc = 431
}
PlainOpCodeLine {
    pocl_code = 90, pocl_ops = [Expr (ExpressionIdentifier (Id "t"
        )),Ident (Id "??3H1")], pocl_loc = 435
}
PlainOpCodeLine {
    pocl_code = 0, pocl_ops = [Expr (ExpressionNumber 0),
        PseudoCode 0,Expr (ExpressionNumber 0)], pocl_loc = 439
}

```