At the end of the rainbow

Ignacio Burgueño
Lua Workshop 2015
Stockholm, Sweden

About Me

Ignacio Burgueño

Developer at inConcert

@iburgueno

https://github.com/ignacio

What will be talking about?

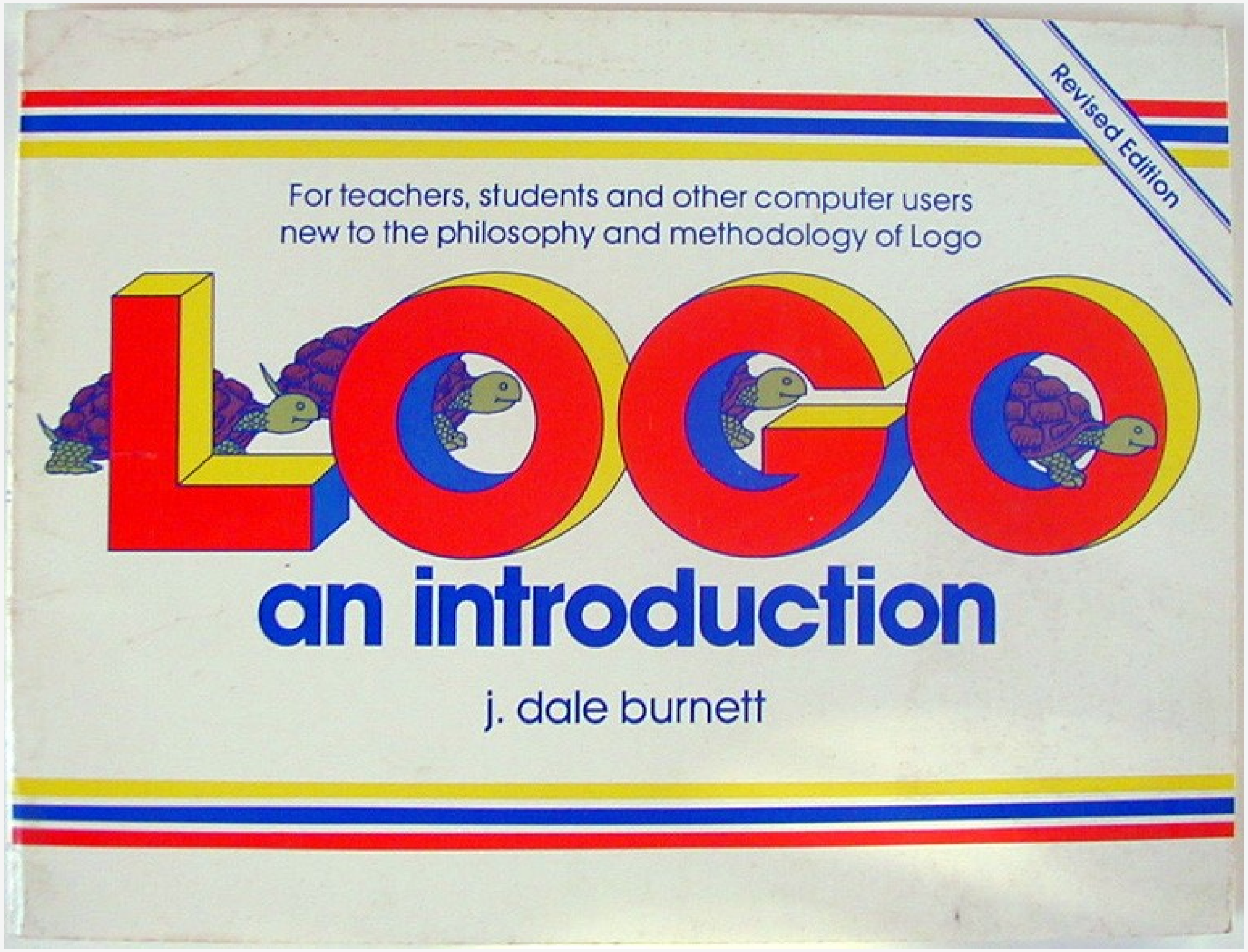
- Writing a ZX Spectrum emulator in Lua 5.3
- Fun with Bitwise operators

A bit of history

apple II

MO5

ORIC



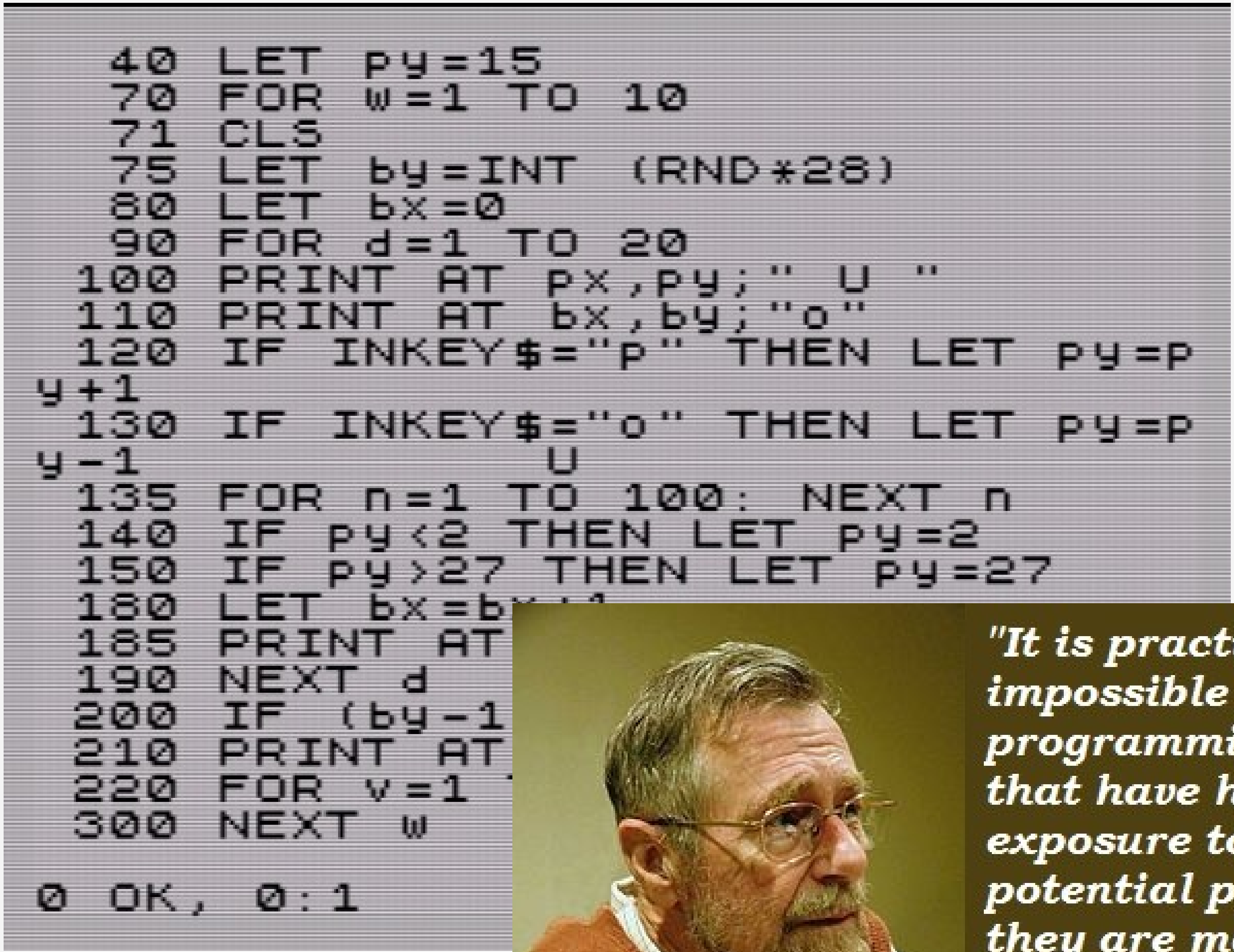
Revised Edition

For teachers, students and other computer users new to the philosophy and methodology of Logo

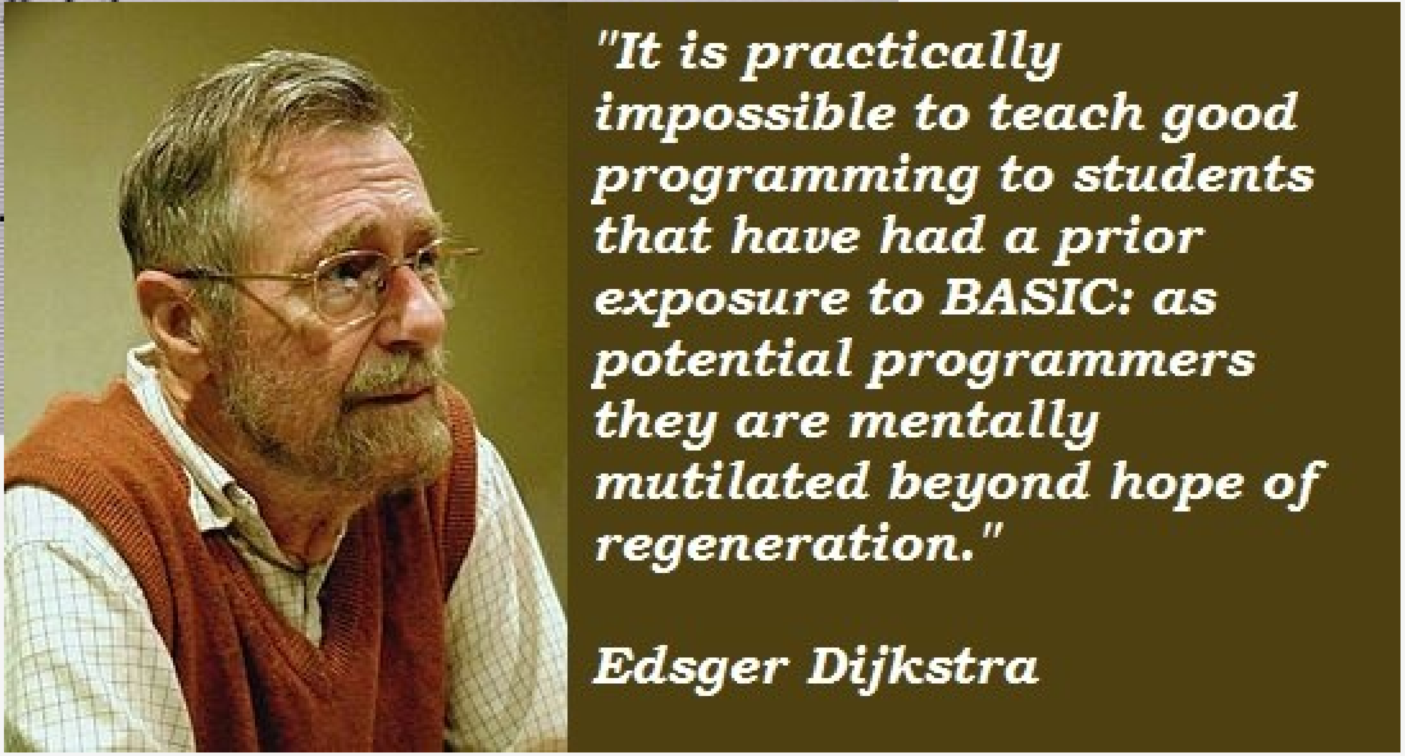
LOGO

an introduction

j. dale burnett



```
   40 LET py=15
   70 FOR w=1 TO 10
   71 CLS
   75 LET by=INT (RND*28)
   80 LET bx=0
   90 FOR d=1 TO 20
  100 PRINT AT px,py;" U "
  110 PRINT AT bx,by;"o"
  120 IF INKEY$="p" THEN LET py=p
y+1
  130 IF INKEY$="o" THEN LET py=p
y-1               U
  135 FOR n=1 TO 100: NEXT n
  140 IF py<2 THEN LET py=2
  150 IF py>27 THEN LET py=27
  180 LET bx=b
  185 PRINT AT
  190 NEXT d
  200 IF (by-1
  210 PRINT AT
  220 FOR v=1 
  300 NEXT w

0 OK, 0:1
```



"It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration."

Edsger Dijkstra

TK 90X
color computer
MICRODIGITAL
CAPS SHIFT
BREAK SPACE
SYMBOL SHIFT
ENTER

A Microdigital lança no Brasil o micro pessoal de maior sucesso no mundo.

A partir de agora a história dos micros pessoais vai ser contada em duas partes: antes e depois do TK 90X.

O TK 90X é, simplesmente, o único micro pessoal lançado no Brasil que merece a classificação de "software machine": um caso raro de micro que pela sua facilidade de uso, grandes recursos e preço acessível recebeu a atenção dos criadores de programas e periféricos em todo o mundo.

Para você ter uma idéia, existem mais de 2 mil programas, 70 livros, 30 periféricos e inúmeras revistas de usuários disponíveis para ele internacionalmente.

E aqui o TK 90X já sai com mais de 100 programas, enquanto outros estão em fase final de desenvolvimento para lhe dar mais opções para trabalhar, aprender ou se divertir que com qualquer outro micro.

O TK 90X tem duas versões de memória (de 16 ou 48 K), imagem de alta resolução gráfica com 8 cores, carregamento rápido de programas (controlável pelo próprio monitor), som pela TV, letras maiúsculas e minúsculas e ainda uma exclusividade: acentuação em português.

Faça o seu programa: peça já uma demonstração do novo TK 90X.

Preço de lançamento*

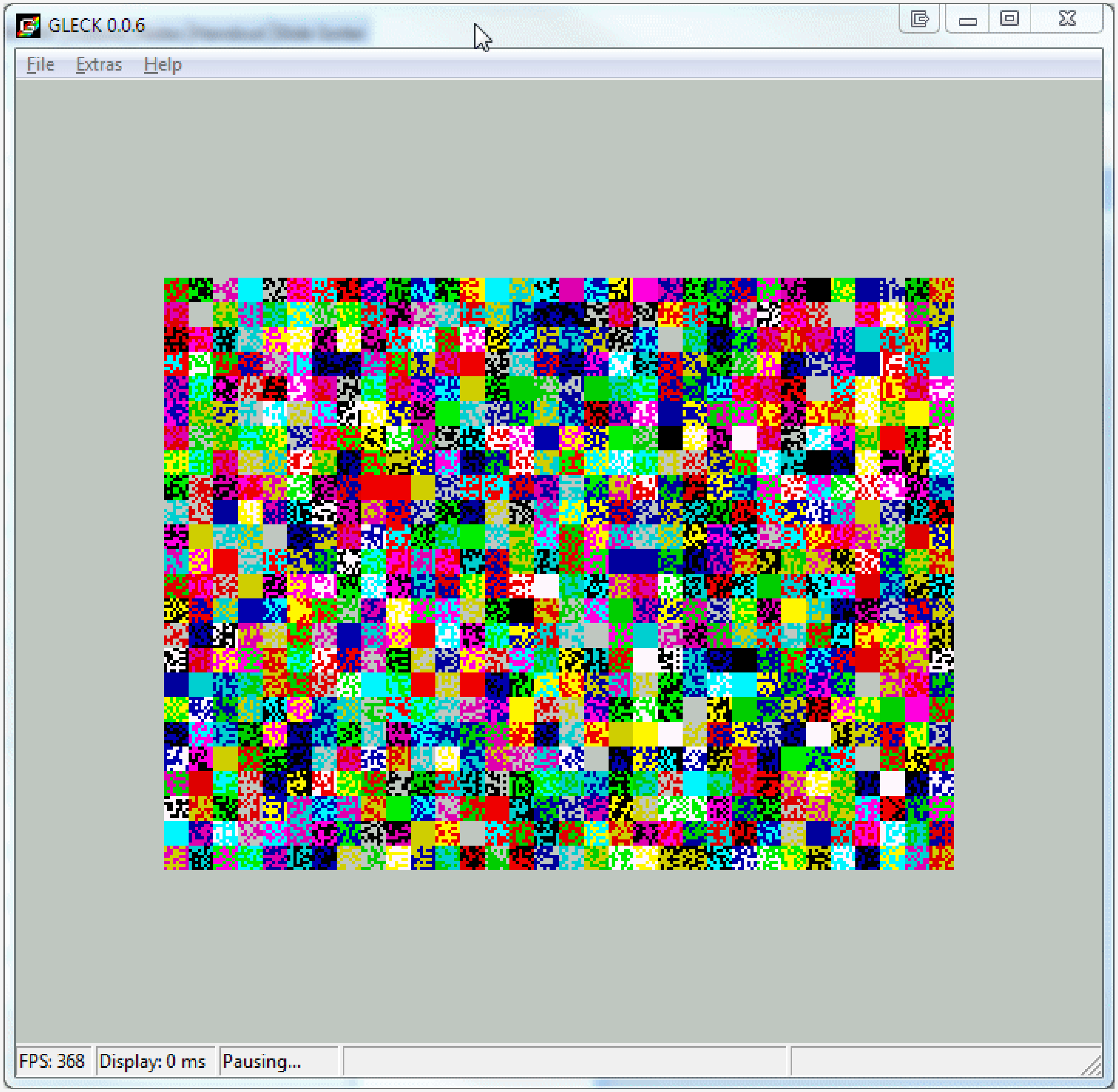
16 K - Cr$ 1.499.850 ● 48 K - Cr$ 1.749.850

MICRODIGITAL

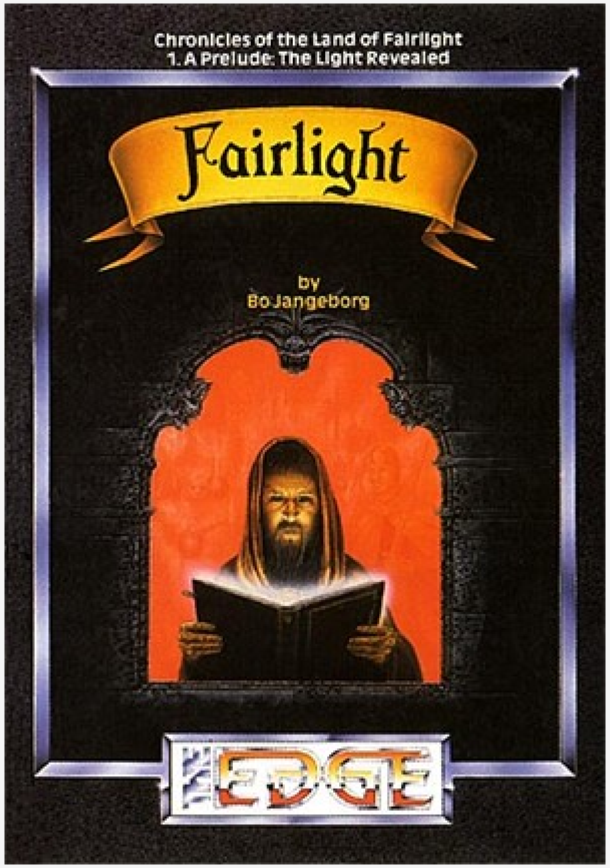
Chegou o micro cheio de programas

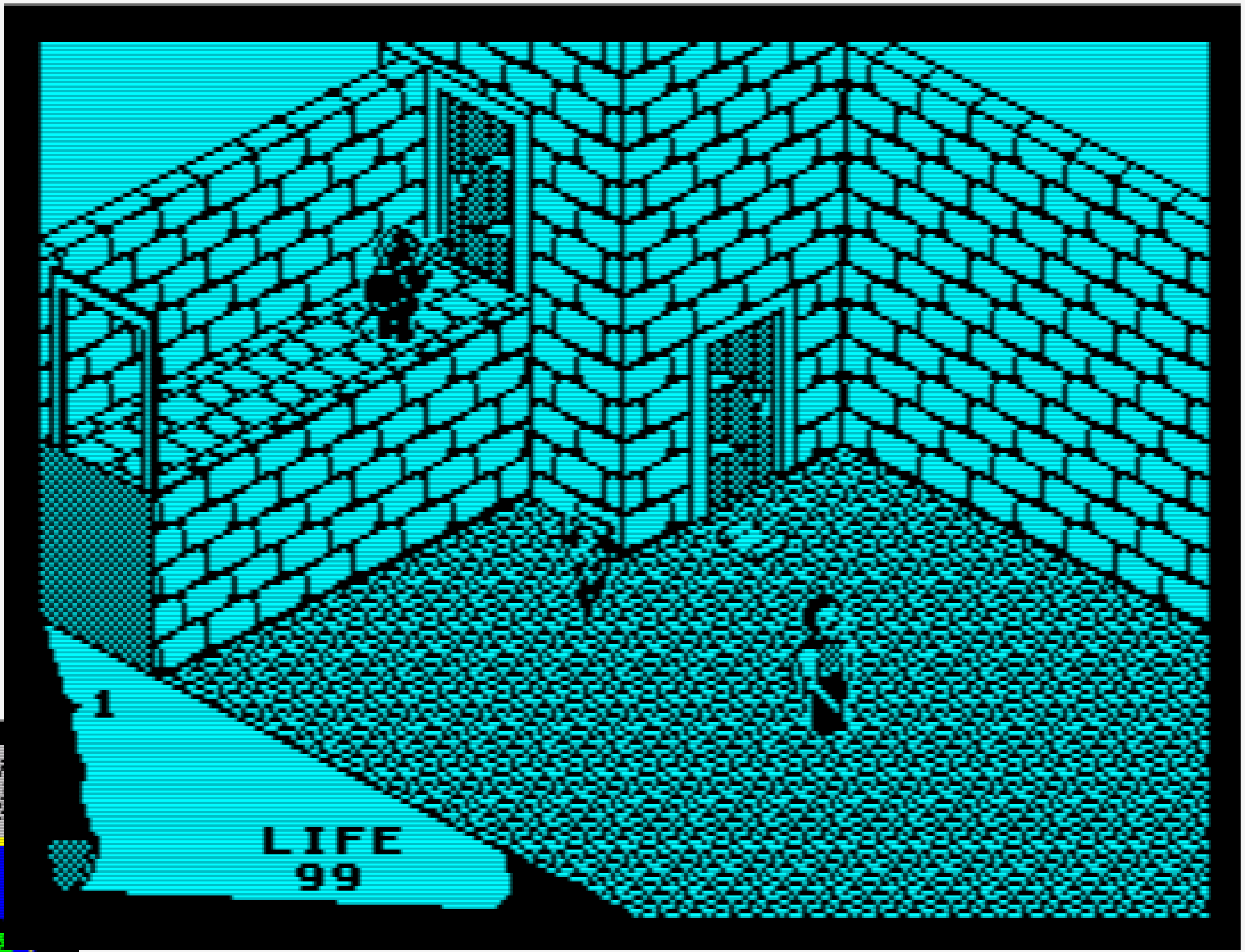
TK 90X

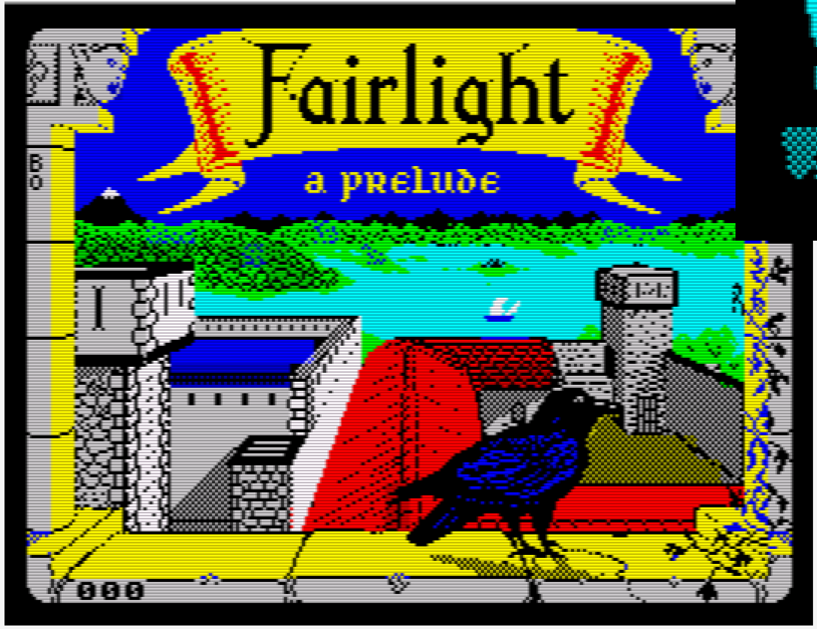
*Sujeito a alteração sem prévio aviso.



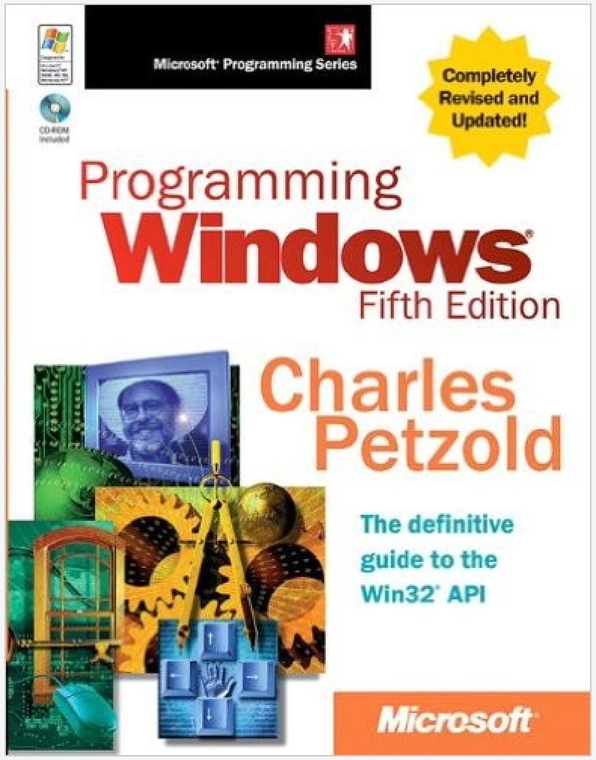
FPS: 368 Display: 0 ms Pausing...


Chronicles of the Land of Fairlight
1. A Prelude: The Light Revealed
Fairlight
by
Bo Jangeborg
EDGE

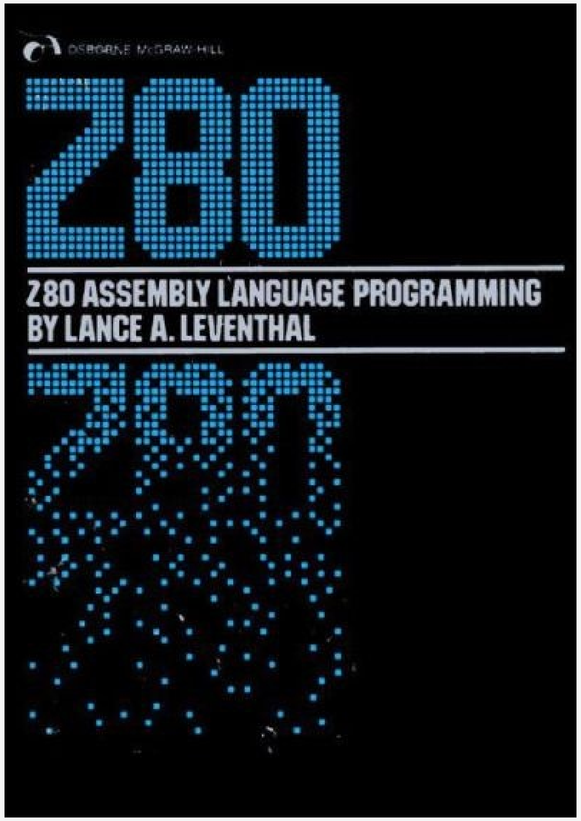




Fairlight
Bo Jangeborg, 1985


Microsoft Programming Series
Completely Revised and Updated!
Programming Windows
Fifth Edition
Charles Petzold
The definitive guide to the Win32 API
Microsoft

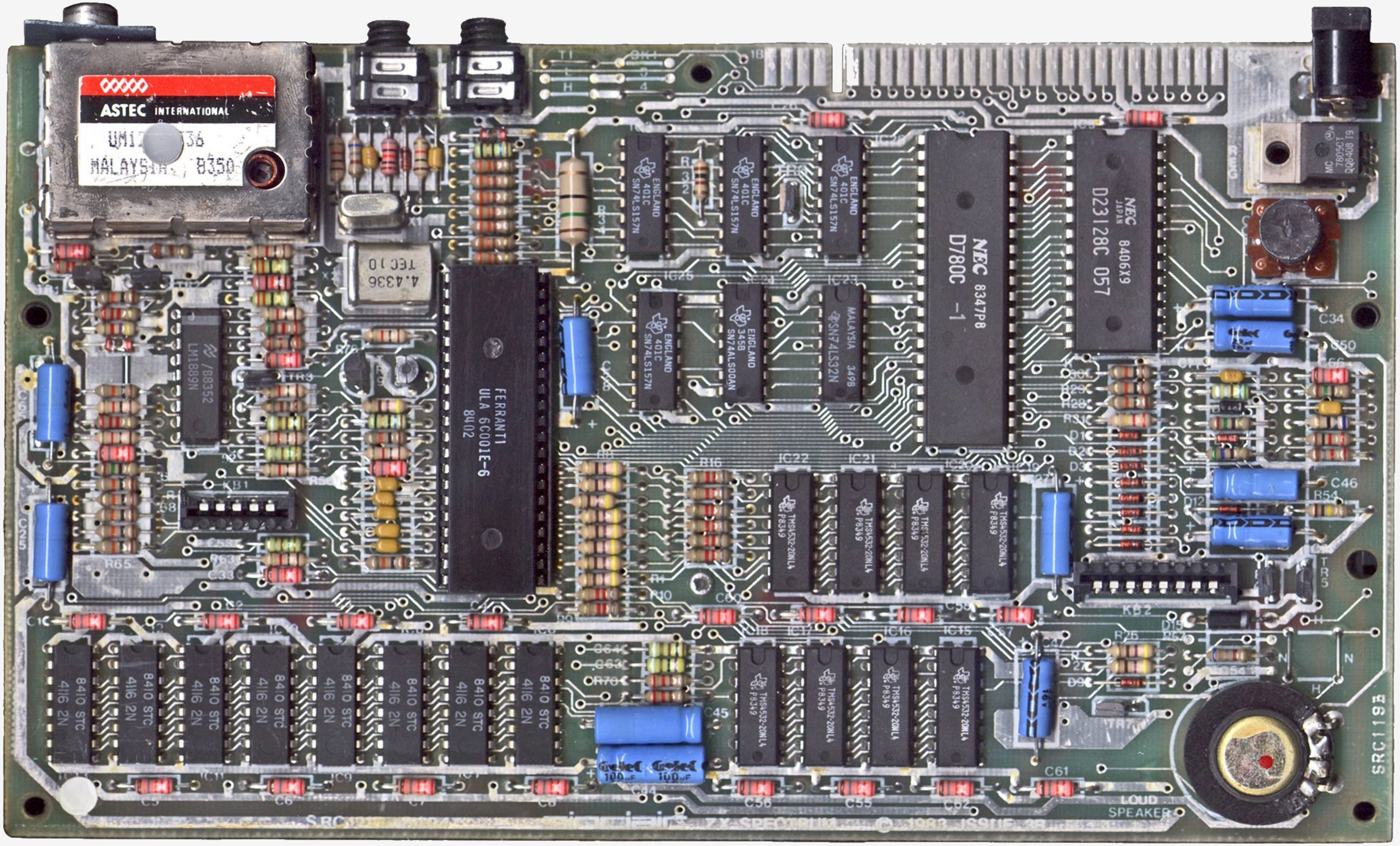
GLECK 0.0.6 - FAIRL48.SNA
File
Extras
Help
Fairlight
a prelude
FPS: 390
Display: 0 ms
Spectrum 48K


OSBORNE McGRAW-HILL
Z80
Z80 ASSEMBLY LANGUAGE PROGRAMMING
BY LANCE A. LEVENTHAL

What is needed?

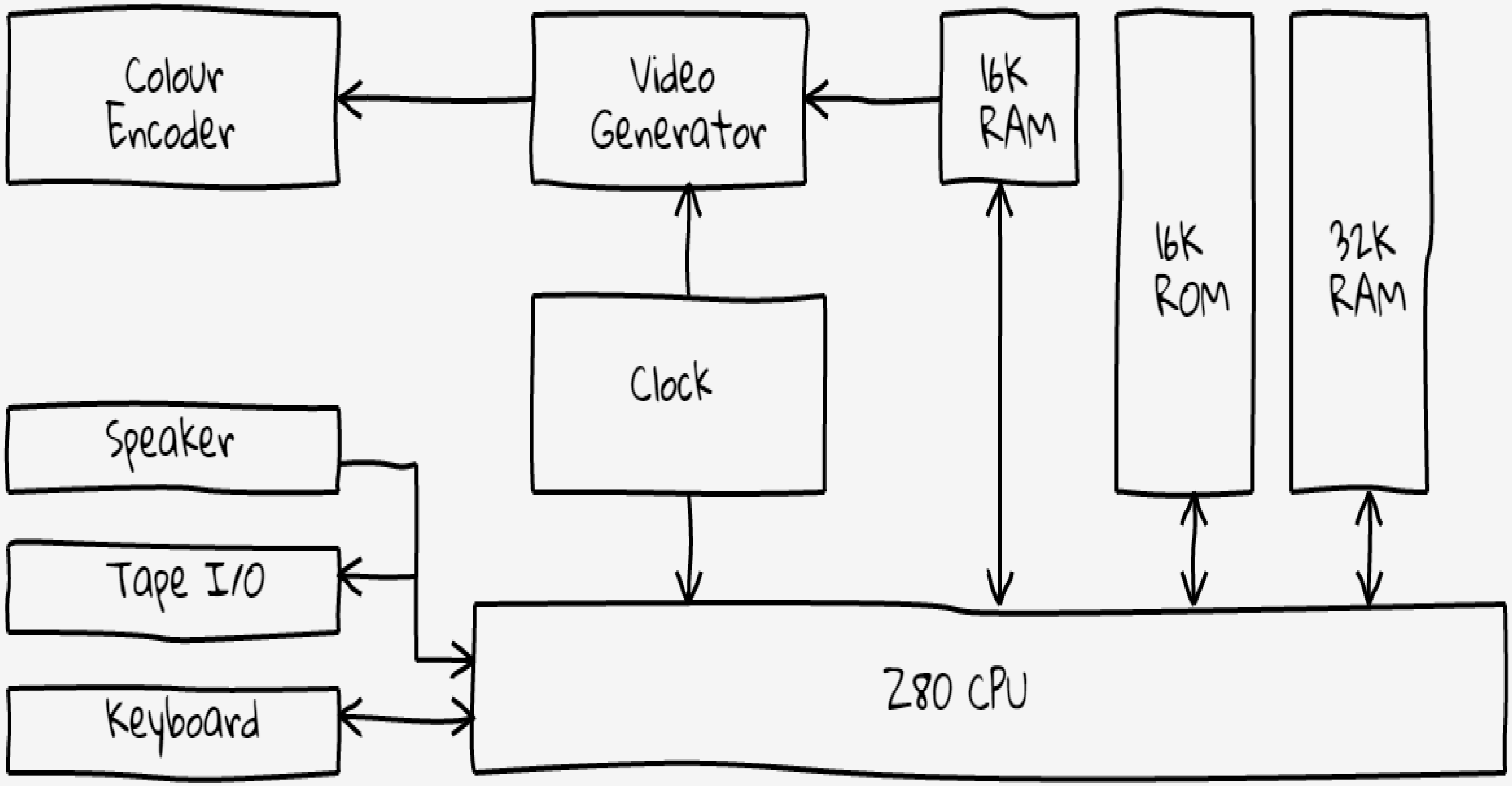
- Replace hardware parts with software
- Which parts?

Architecture



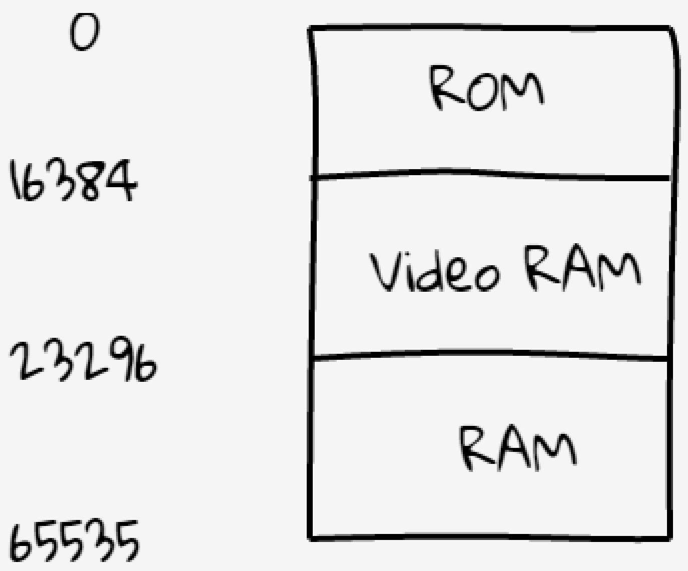
https://commons.wikimedia.org/wiki/File:ZXspectrum_mb.jpg

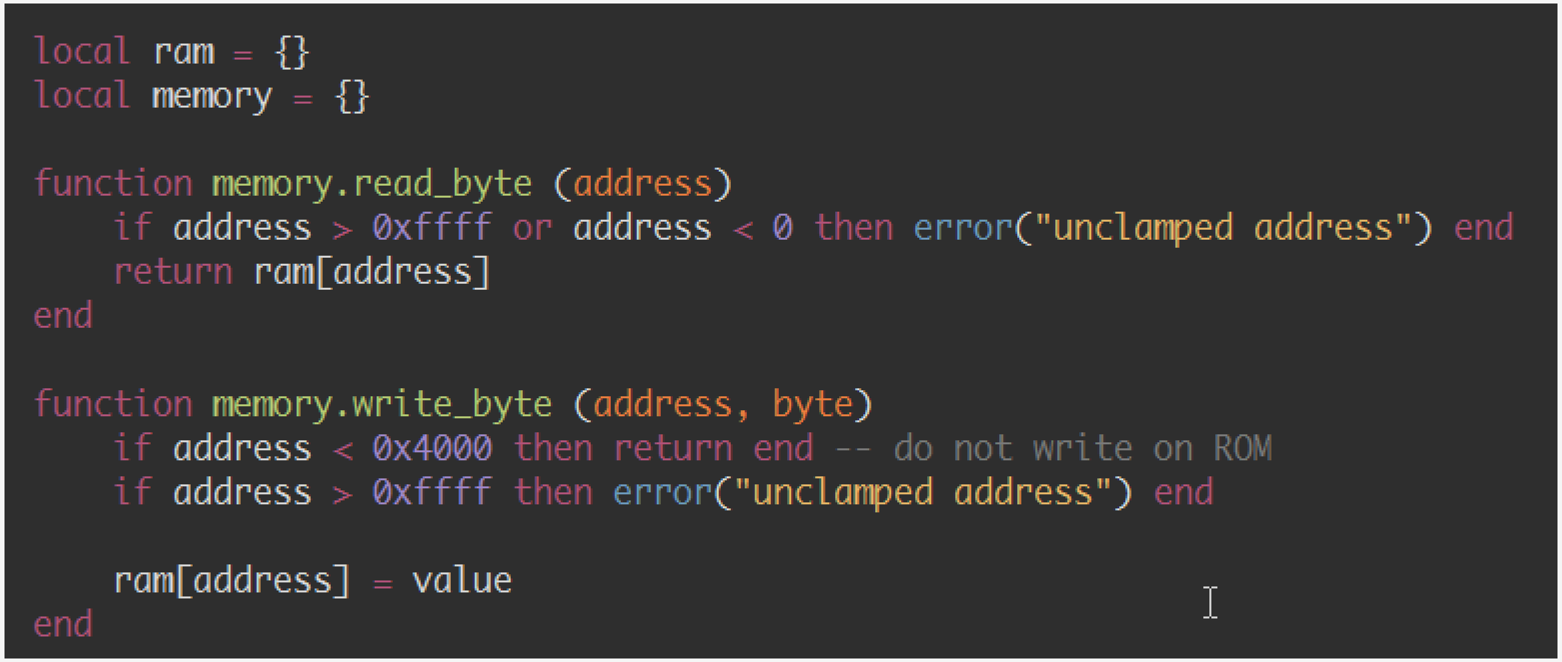
Architecture



"ZX Spectrum block diagram" taken from the book “The ZX Spectrum ULA” by Chris Smith

Memory

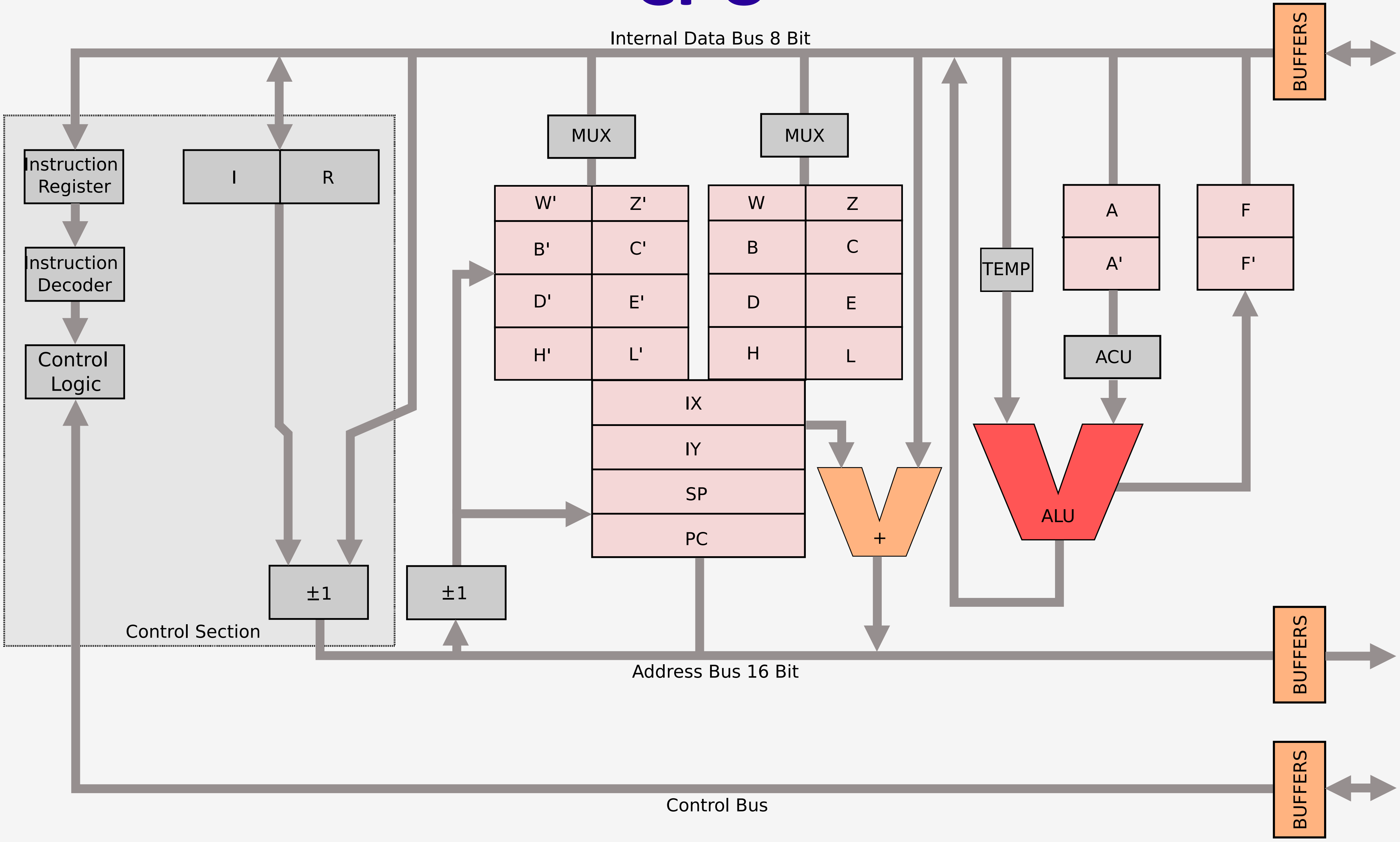




```
local ram = {}
local memory = {}

function memory.read_byte (address)
    if address > 0xffff or address < 0 then error("unclamped address") end
    return ram[address]
end

function memory.write_byte (address, byte)
    if address < 0x4000 then return end -- do not write on ROM
    if address > 0xffff then error("unclamped address") end

    ram[address] = value
end
```

Z80 Architecture

CPU



CPU

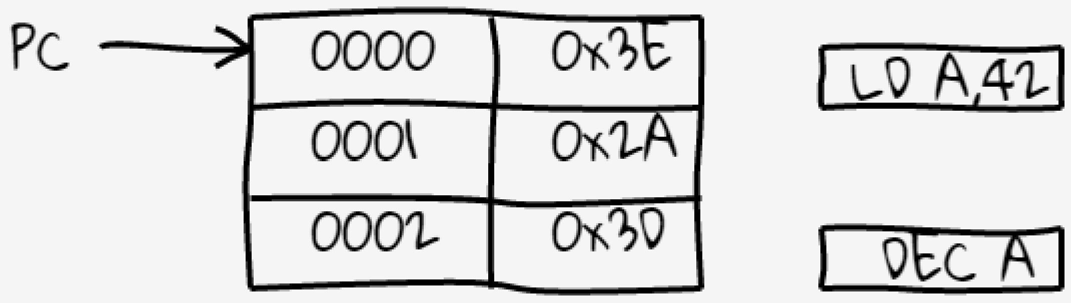
```lua
local machine = {}

machine.ram = {}
machine.z80 = {
    -- main registers
    A = 0, F = 0, H = 0, L = 0, B = 0, C = 0, D = 0, E = 0, IX = 0, IY = 0,
    -- shadow registers
    Ap = 0, Fp = 0, Hp = 0, Lp = 0, Bp = 0, Cp = 0, Dp = 0, Ep = 0,
    -- others
    PC = 0, iff1 = 0, iff2 = 0, I = 0,
    int_mode = 0, SP = 0, R = 0, tstates = 0, halted = false
}

machine.memory = {
    read_byte = function (address)
        if address > 0xffff or address < 0 then error("unclamped address") end
        return machine.ram[address]
    end,
    write_byte = function (address, byte)
        if address < 0x4000 then return end -- do not write on ROM
        if address > 0xffff then error("unclamped address") end

        ram[address] = value
    end
}
```

CPU – Instruction Set

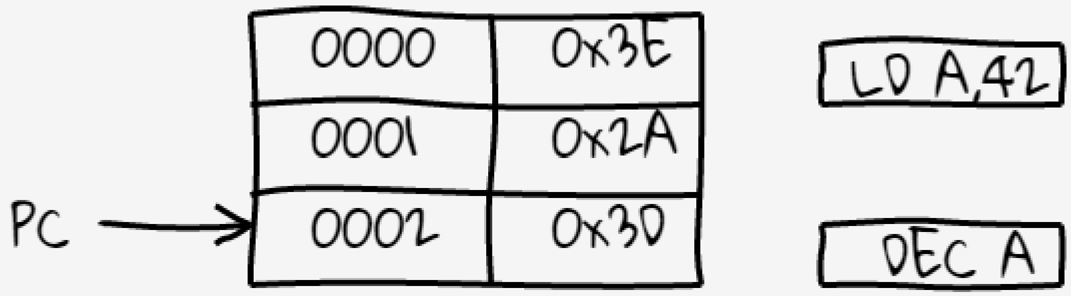
- 8-bit arithmetic and logic operations
- 16-bit arithmetic
- 8-bit load
- 16-bit load
- Bit set, reset, and test
- Call, return, and restart
- Exchange, block transfer, and search
- General purpose arithmetic and CPU control
- Input and output
- Jump
- Rotate and shift

$\boxed{\text{LD A,42} \longrightarrow \text{0x3e 0x2a}}$

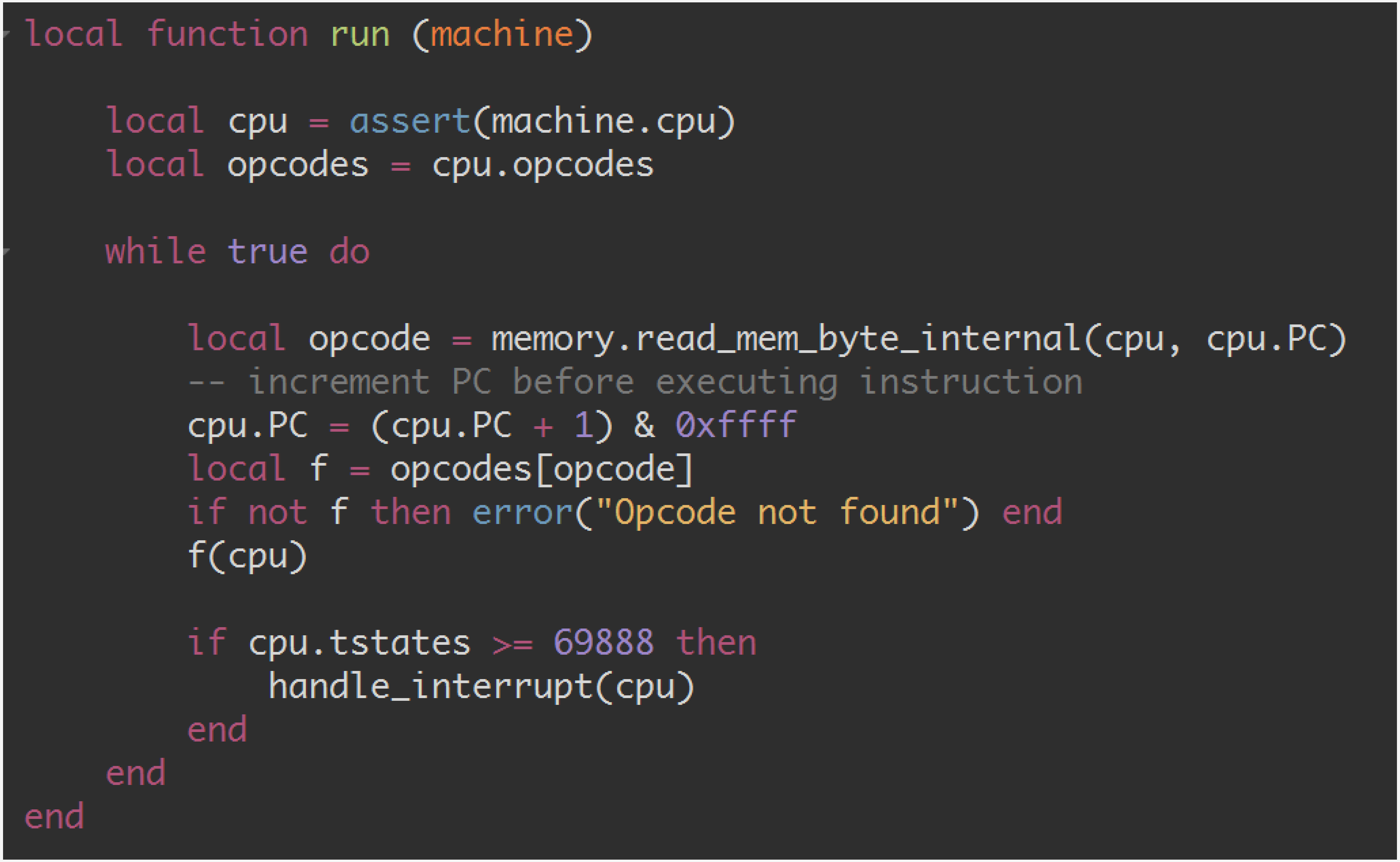
CPU – Fetch/Execute Cycle



CPU – Fetch/Execute Cycle

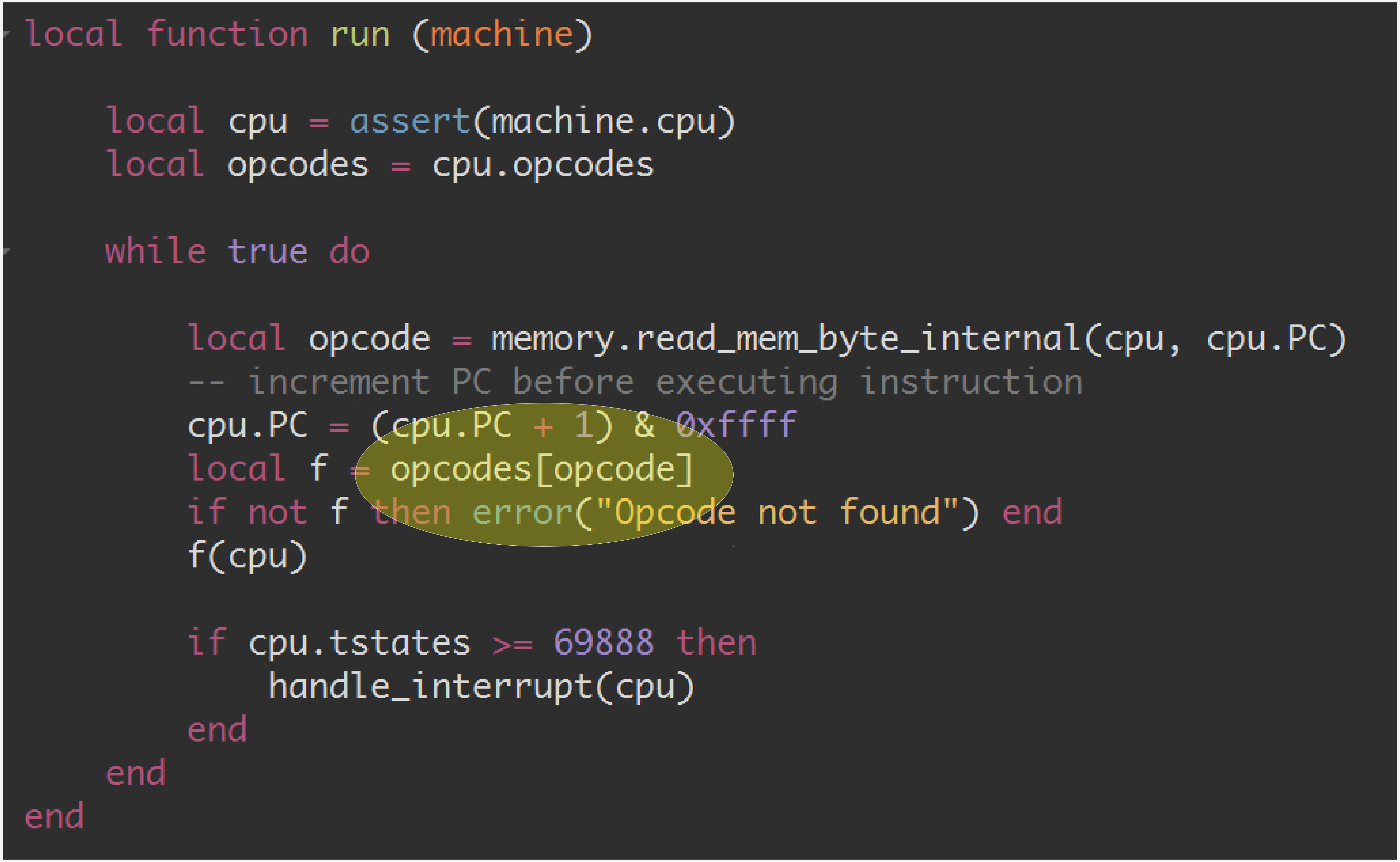


CPU – Fetch/Execute Cycle



```
local function run (machine)

    local cpu = assert(machine.cpu)
    local opcodes = cpu.opcodes

    while true do

        local opcode = memory.read_mem_byte_internal(cpu, cpu.PC)
        -- increment PC before executing instruction
        cpu.PC = (cpu.PC + 1) & 0xffff
        local f = opcodes[opcode]
        if not f then error("Opcode not found") end
        f(cpu)

        if cpu.tstates >= 69888 then
            handle_interrupt(cpu)
        end
    end
end
```

CPU – Fetch/Execute Cycle



```
local function run (machine)

    local cpu = assert(machine.cpu)
    local opcodes = cpu.opcodes

    while true do

        local opcode = memory.read_mem_byte_internal(cpu, cpu.PC)
        -- increment PC before executing instruction
        cpu.PC = (cpu.PC + 1) & 0xffff
        local f = opcodes[opcode]
        if not f then error("Opcode not found") end
        f(cpu)

        if cpu.tstates >= 69888 then
            handle_interrupt(cpu)
        end
    end
end
```

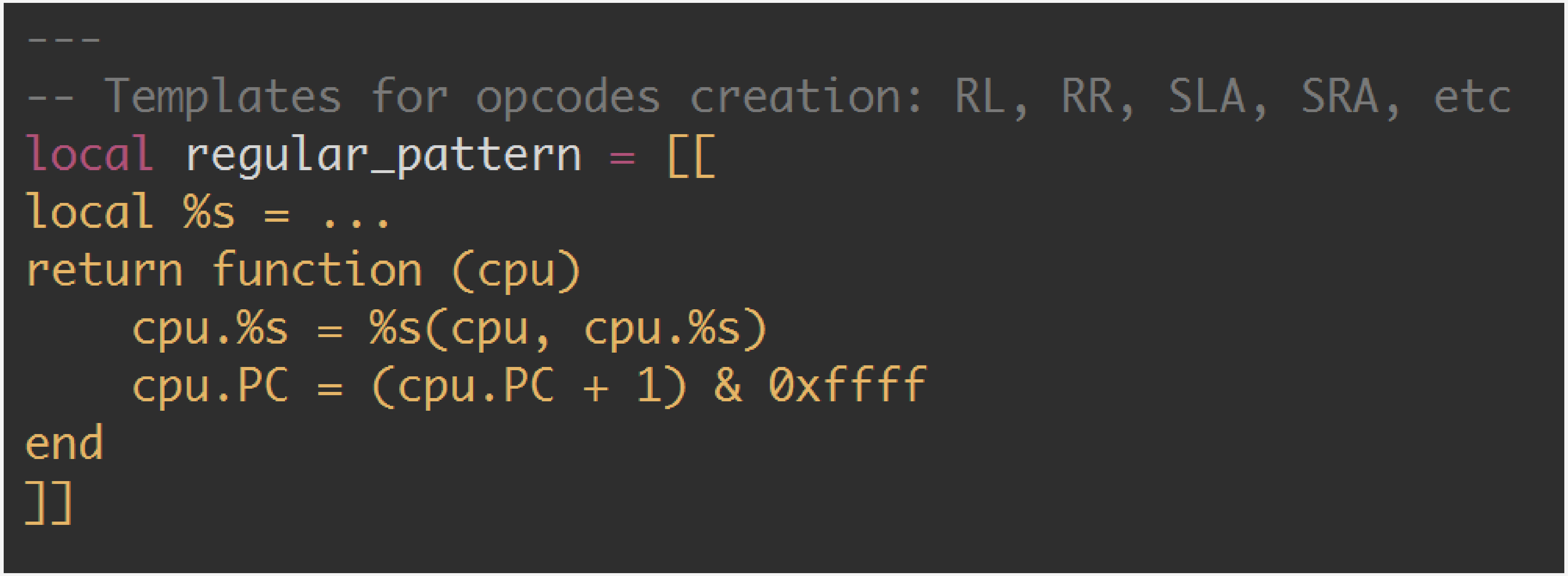
CPU - Opcodes

```lua
local opcodes = {}

-- LD A,nn
opcodes[0x3E] = function (cpu)
    cpu.A = read_mem_byte(cpu, cpu.PC)
    cpu.PC = (cpu.PC + 1) & 0xffff
end

-- DEC A
opcodes[0x3d] = function (cpu)
    cpu.A = DEC(cpu, cpu.A)
end



local function DEC (cpu, byte)
    assert(byte)
    cpu.F = (cpu.F & FLAG_C) | ( ((byte & 0x0f) ~= 0) and 0 or FLAG_H ) | FLAG_N
    byte = (byte - 1) & 0xff
    cpu.F = cpu.F | ( byte == 0x7f and FLAG_PV or 0 ) | sz53_table[byte]
    return byte
end
```

CPU – Instruction families

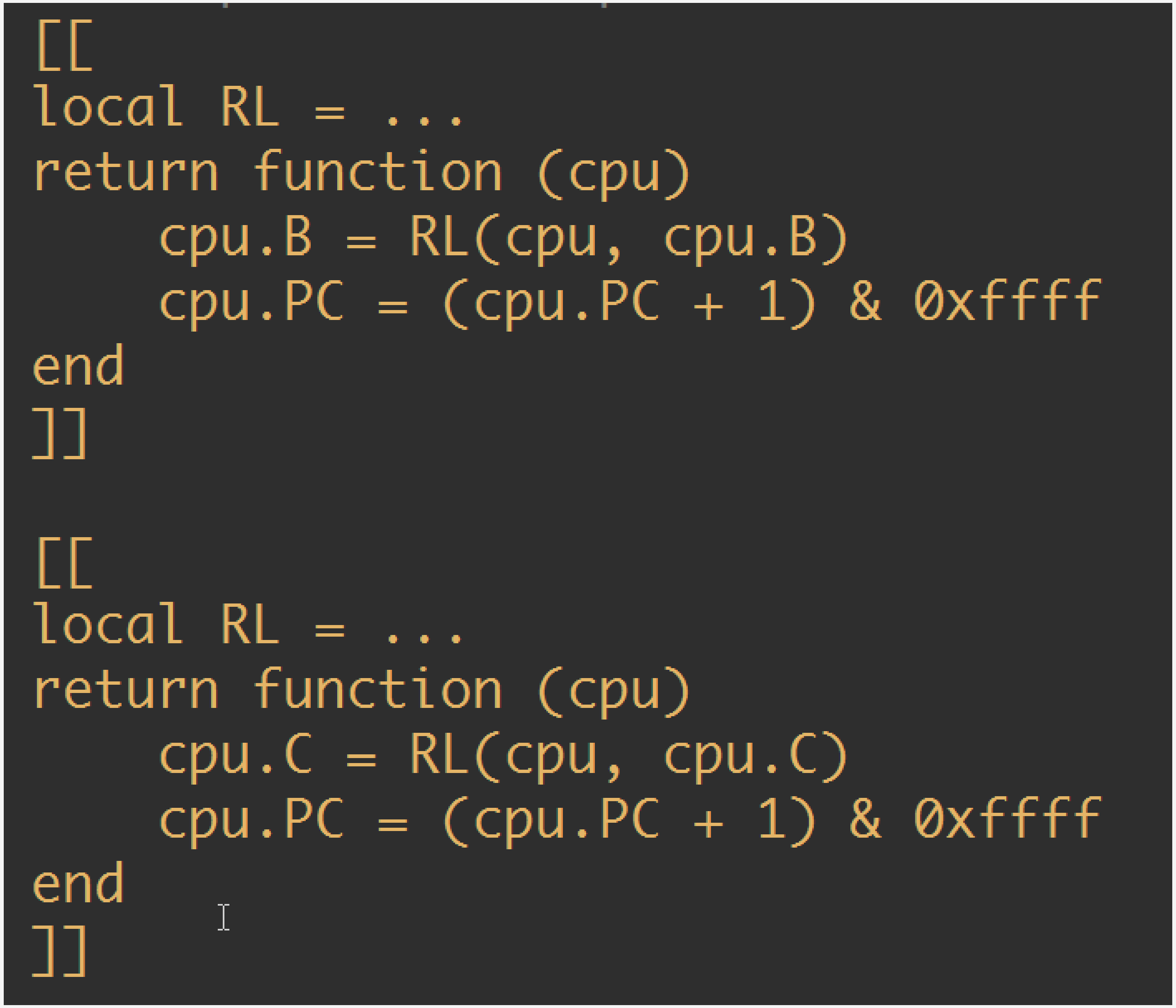
- Same operation is applied to different registers
- One instruction for each combination
- Avoid writing the same function over and over
- Generate them using a template

CPU – Instruction families



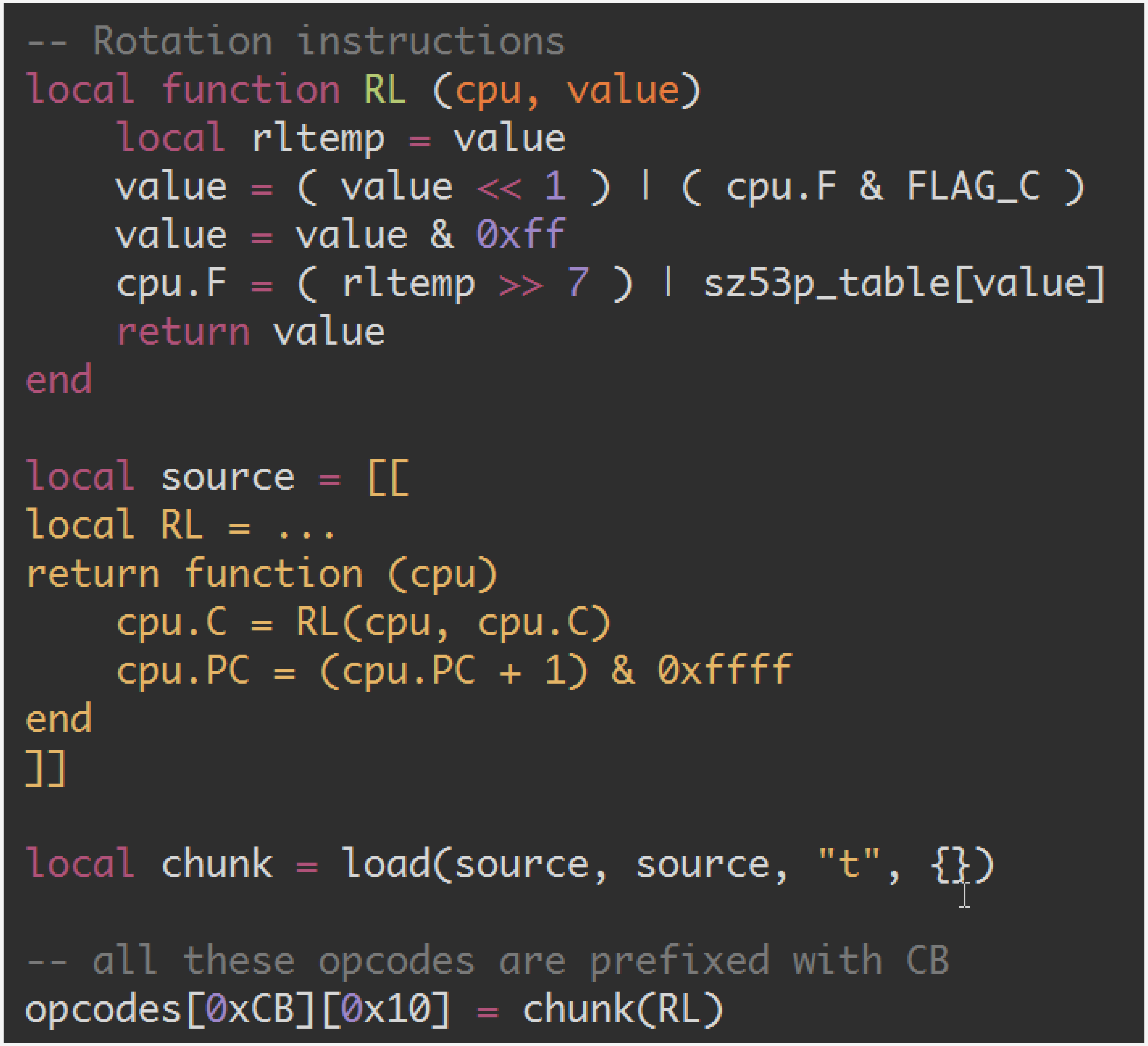
```
---
-- Templates for opcodes creation: RL, RR, SLA, SRA, etc
local regular_pattern = [[
local %s = ...
return function (cpu)
    cpu.%s = %s(cpu, cpu.%s)
    cpu.PC = (cpu.PC + 1) & 0xffff
end
]]
```

CPU – Instruction families



```
[[
local RL = ...
return function (cpu)
    cpu.B = RL(cpu, cpu.B)
    cpu.PC = (cpu.PC + 1) & 0xffff
end
]]

[[
local RL = ...
return function (cpu)
    cpu.C = RL(cpu, cpu.C)
    cpu.PC = (cpu.PC + 1) & 0xffff
end
]]
```

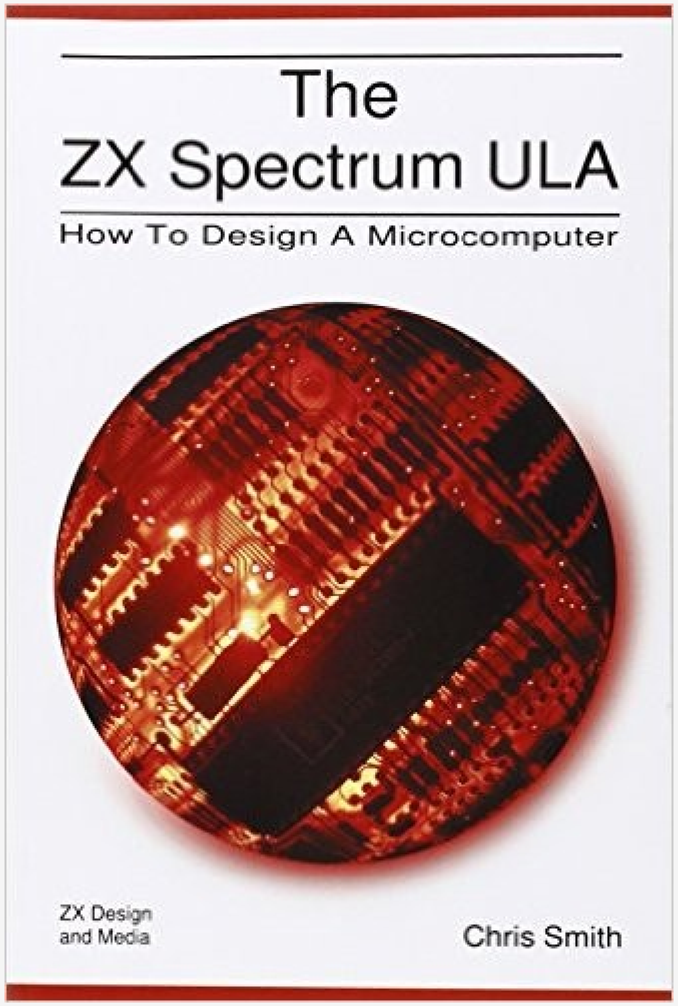
CPU – Instruction families



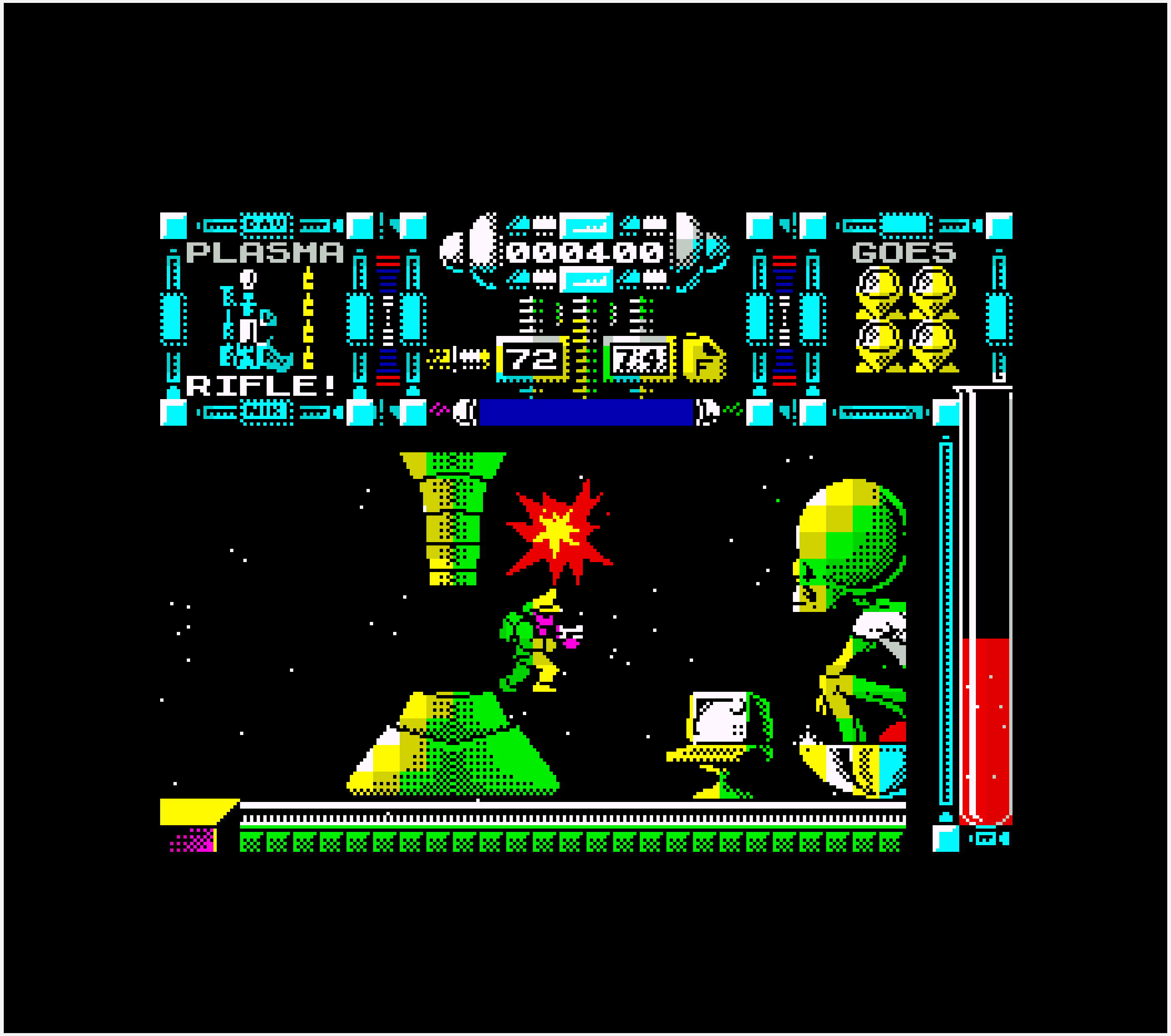
```lua
-- Rotation instructions
local function RL (cpu, value)
    local rltemp = value
    value = ( value << 1 ) | ( cpu.F & FLAG_C )
    value = value & 0xff
    cpu.F = ( rltemp >> 7 ) | sz53p_table[value]
    return value
end

local source = [[
local RL = ...
return function (cpu)
    cpu.C = RL(cpu, cpu.C)
    cpu.PC = (cpu.PC + 1) & 0xffff
end
]]

local chunk = load(source, source, "t", {})

-- all these opcodes are prefixed with CB
opcodes[0xCB][0x10] = chunk(RL)
```

See what's going on

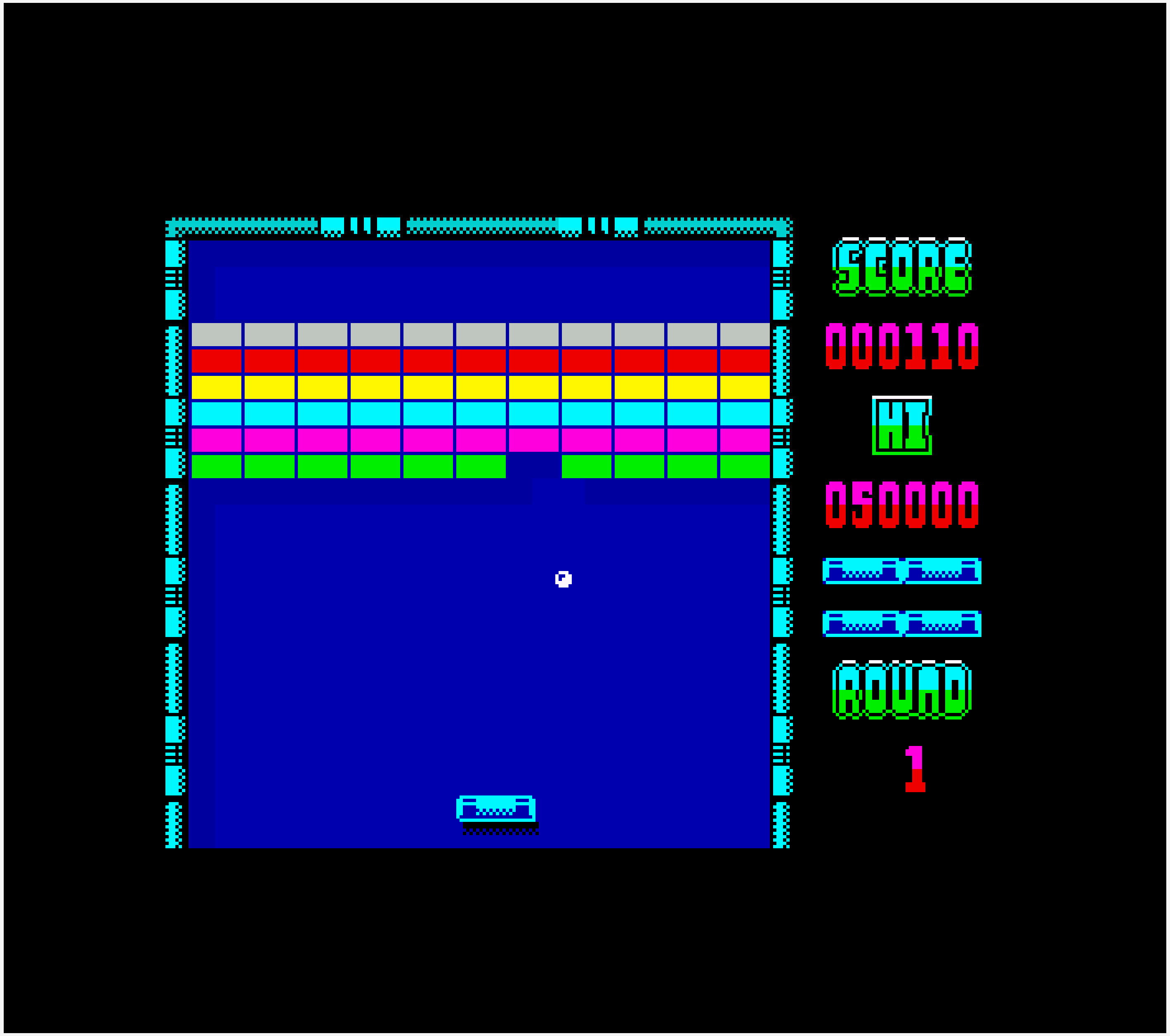
Drawing to the Screen


The
ZX Spectrum ULA
How To Design A Microcomputer
ZX Design and Media
Chris Smith

Drawing to the Screen



Drawing to the Screen



Drawing to the Screen

Drawing to the Screen

luasdl2

https://github.com/Tangent128/luasdl2

More information on the subject:

How "oldschool" graphics worked video series

Getting software in it

(aka, how do I load games on this?)

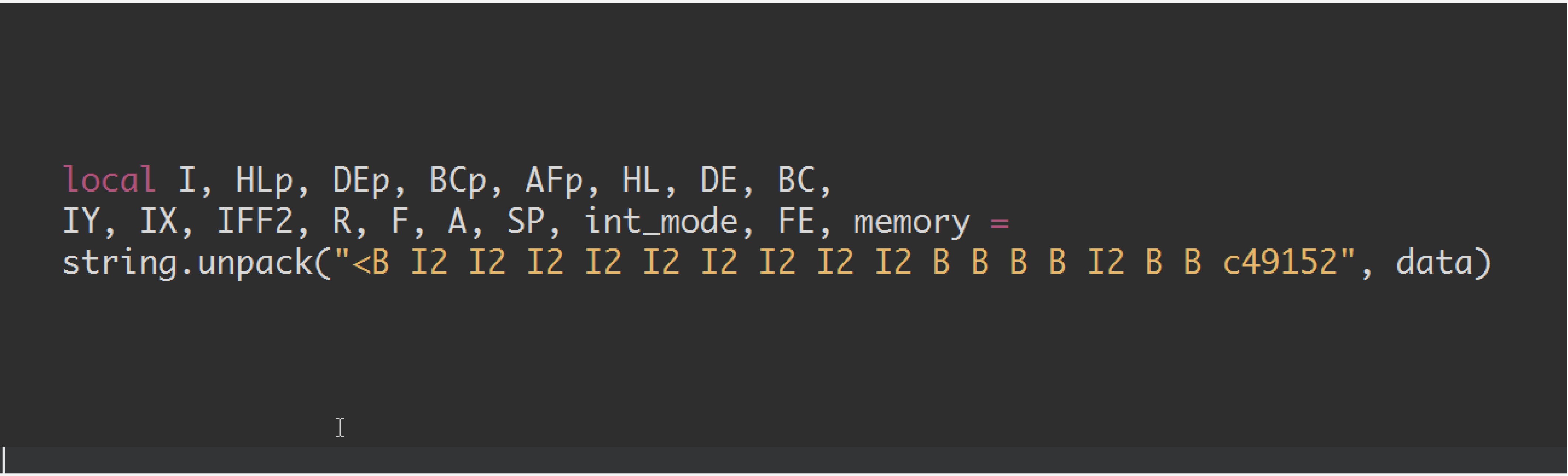
Parsing Binary data

File Formats - SNA

Offset (bytes)	Size	Description
0	1 byte	I
1	8 words	HL', DE', BC', AF'
9	10 words	HL, DE, BC, IY, IX
19	1 byte	Interrupt
20	1 byte	R
21	4 words	AF, SP
25	1 byte	Interruption Mode
26	1 byte	Border Color
27	49152 bytes	RAM dump 16384 .. 65535
Total: 49179 bytes		

File Formats - SNA

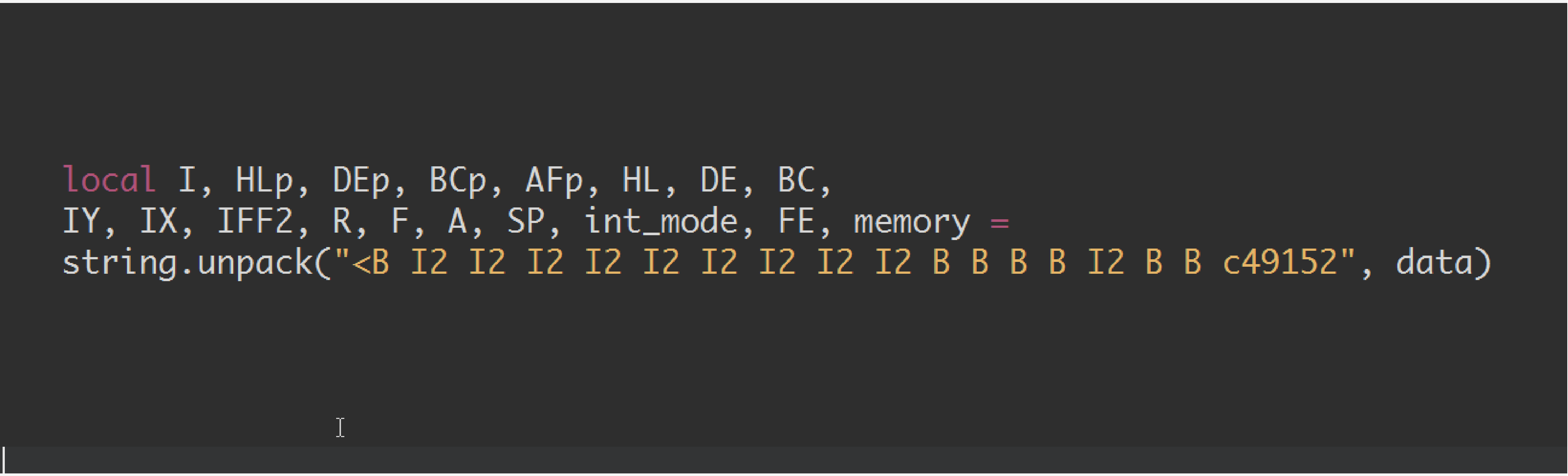
- Parsing binary formats
- Don't want to split strings
- Enter **string.unpack**



```lua
local I, HLp, DEp, BCp, AFp, HL, DE, BC,
IY, IX, IFF2, R, F, A, SP, int_mode, FE, memory =
string.unpack("<B I2 I2 I2 I2 I2 I2 I2 I2 I2 B B B B I2 B B c49152", data)
```

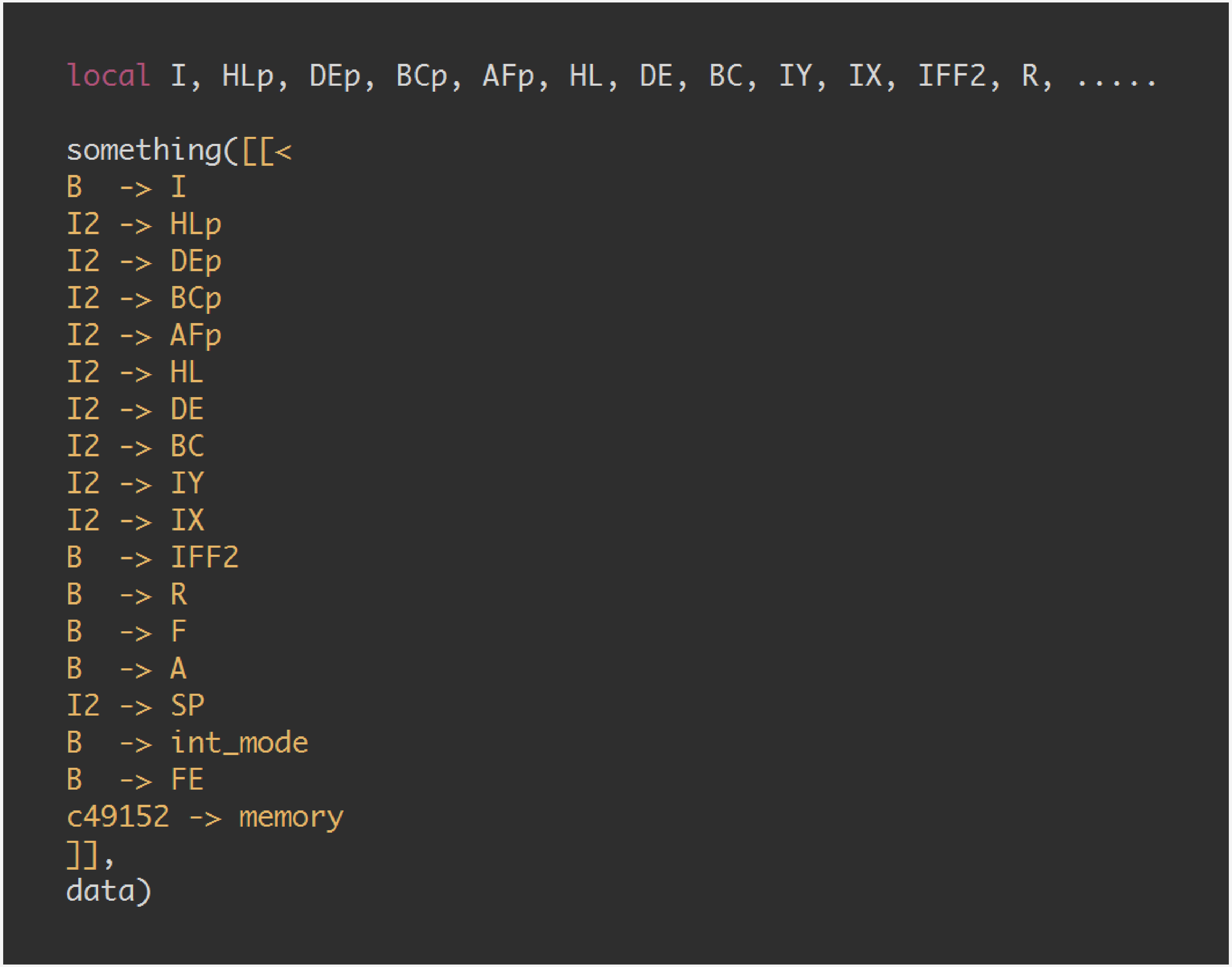
File Formats - SNA

- Cumbersome to read
- What value ends up where?
- Extend the unpack format string



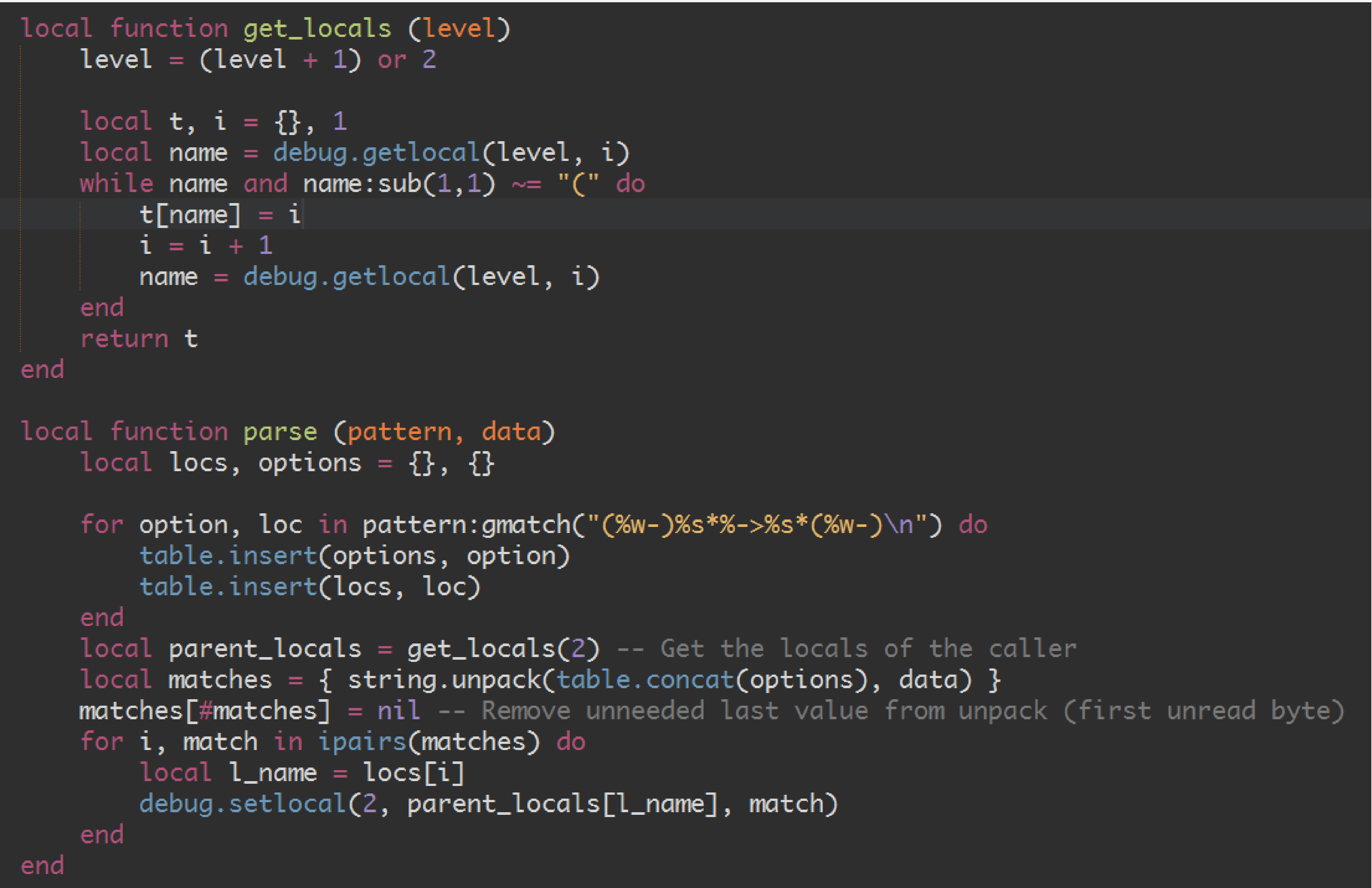
```
local I, HLp, DEp, BCp, AFp, HL, DE, BC,
IY, IX, IFF2, R, F, A, SP, int_mode, FE, memory =
string.unpack("<B I2 I2 I2 I2 I2 I2 I2 I2 I2 B B B B I2 B B c49152", data)
```

File Formats - SNA



```
local I, HLp, DEp, BCp, AFp, HL, DE, BC, IY, IX, IFF2, R, .....

something([[<
B  -> I
I2 -> HLp
I2 -> DEp
I2 -> BCp
I2 -> AFp
I2 -> HL
I2 -> DE
I2 -> BC
I2 -> IY
I2 -> IX
B  -> IFF2
B  -> R
B  -> F
B  -> A
I2 -> SP
B  -> int_mode
B  -> FE
c49152 -> memory
]],
data)
```

File Formats - SNA



```lua
local function get_locals (level)
    level = (level + 1) or 2

    local t, i = {}, 1
    local name = debug.getlocal(level, i)
    while name and name:sub(1,1) ~= "(" do
        t[name] = i
        i = i + 1
        name = debug.getlocal(level, i)
    end
    return t
end

local function parse (pattern, data)
    local locs, options = {}, {}

    for option, loc in pattern:gmatch("(%w-)%s*%->%s*(%w-)\n") do
        table.insert(options, option)
        table.insert(locs, loc)
    end
    local parent_locals = get_locals(2) -- Get the locals of the caller
    local matches = { string.unpack(table.concat(options), data) }
    matches[#matches] = nil -- Remove unneeded last value from unpack (first unread byte)
    for i, match in ipairs(matches) do
        local l_name = locs[i]
        debug.setlocal(2, parent_locals[l_name], match)
    end
end
```

Testing

- Run little code snippets
- Make sure registers have the correct values
- Correct memory locations are read and written at the right times

Testing

- Run little code snippets
- Make sure registers have the correct values
- Correct memory locations are read and written at the right times

Demo

Where to get it?

https://github.com/ignacio/luagleck

Where do I get software to try?

http://www.worldofspectrum.org/archive.html

Thank you!

Any questions?

Ignacio Burgueño - @iburgueno