Natalius 8 bit RISC Processor

Fabio Andrés Guzmán Figueroa
fabioandres.guzman@opencores.org
fabio.guzmanf@upb.edu.co

1. Description:

Natalius is a compact, capable and fully embedded 8 bit RISC processor core described 100% in Verilog. It occupies about 268 Slices, 124 FF, 503 LUTs (4 input) in Xilinx Spartan3E1600 (around 1.67% slices). Natalius offers an assembler that can run on any python console. The instruction memory is implemented using two Xilinx BlockRAM Memories, it stores 2048 instructions, each instruction has a width of 16 bits (2048x16). Each instruction takes 3 clock cycles to be executed.

Features:

1. 8 Bit ALU
2. 8x8 Register File
3. 2048x16 Instruction Memory
4. 32x8 Ram Memory
5. 16x11 Stack Memory
6. Three CLK / Instruction
7. Carry and Zero flags
8. 8 bit Address Port (until 256 Peripherals)

2. Instruction Set:

Natalius contains the typical instruction set on a processor. These are: Memory access, arithmetic, logical and flow control, as described in Table 1. Detailed instruction set shown in Table 3.

Instruction	Description	Type	Instruction	Description	Type
ldi	load immediate	memory access	ret	return subrutine	flow control
ldm	load from memory	memory access	adi	add with imm	arithmetic
stm	store to memory	memory access	csz	csr if zero	flow control
cmp	compare	arithmetic	cnz	csr if no zero	flow control
add	addition	arithmetic	csc	csr if carry	flow control
sub	subtraction	arithmetic	cnc	csr if no carry	flow control
and	logic and	logical	sl0	shift left zero fill	logical
oor	logic or	logical	sl1	shift left one fill	logical
xor	logic xor	logical	sr0	shift right zero fill	logical
jmp	jump	flow control	sr1	shiftright one fill	logical
jpz	jump if zero	flow control	rrl	rotary register left	logical
jnz	jump if no zero	flow control	rrr	rotay register right	logical
jpc	jump if carry	flow control	not	logic not	logical
jnc	jump if no carry	flow control	nop	no operation	nop
csr	call subrutine	flow control			

Table 1. Natalius Instruction Set

3. Natalius Interface Signals:

The top-level interface signals to the Natalius Processor appear in Figure 1. and are described in Table 2.

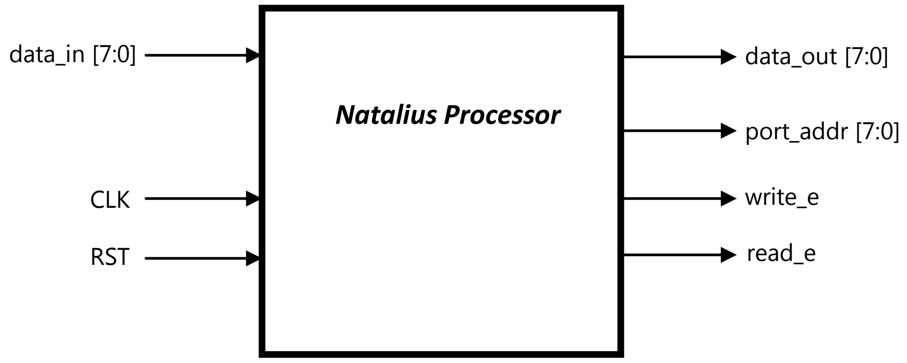


Figure 1. Natalius interface connections

Signal	Direction	Description
clk	input	**clock input:** All Natalius registers are clocked from the rising clock edge
rst	input	**reset input:** To reset the Natalius processor, this rst is asynchronous input and it set program counter register to zero address
data_in[7:0]	input	**Input data port:** The data is captured on the rising edge of CLK (used in *ldm* instruction)
data_out[7:0]	output	**Output data port:** Output data appears on this port for three CLK cycles during a stm instruction, capture this data when write_e is high (uded in *stm* instruction)
port_addr[7:0]	output	**Port address:** This addresses the peripheral port to the input or output by instruction *ldm* or *stm*

Table 2. Interface signals descriptions

4. Organization of source codes:

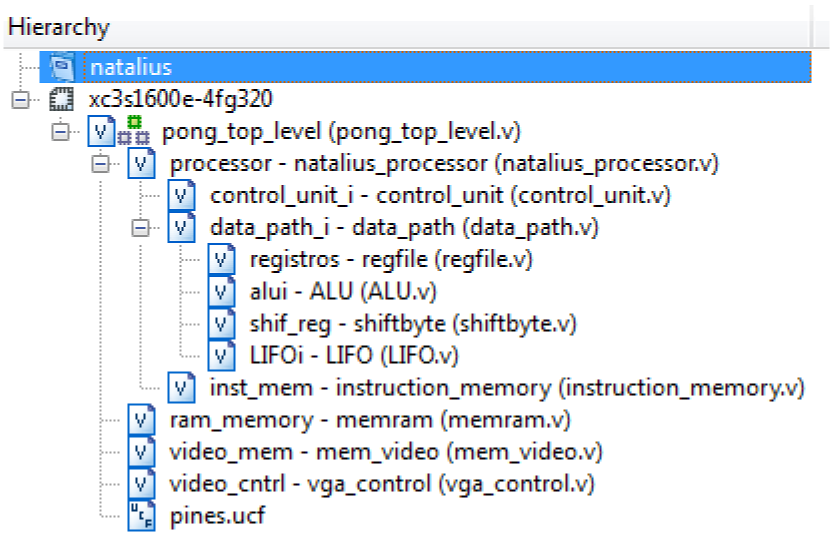
The hierarchy of the files are shown in Figure 2.

Opcode	Instr	Description	Use
2	ldi	load immediate	`ldi rd,imm          (rd=imm)`
3	ldm	load from memory	`ldm rd,port_addr    (rd=data_in <= mem[port_addr])`
4	stm	store to memory	`stm rd,port_addr    (rd=data_out => mem[port_addr])`
5	cmp	compare	`cmp rd,rs       (affects carry and zero)`
6	add	addition	`add rd,rs       (rd=rd+rs)`
7	sub	subtraction	`sub rd,rs       (rd=rd-rs)`
8	and	logic and	`and rd,rs       (rd=rd and rs)`
9	oor	logic or	`oor rd,rs       (rd=rd or rs)`
10	xor	logic xor	`xor rd,rs       (rd=rd xor rs)`
11	jmp	jump	`jmp inst_addr   (pc=inst_addr)`
12	jpz	jump if zero	`jpz inst_addr   (pc=inst_addr if zero)`
13	jnz	jump if no zero	`jnz inst_addr   (pc=inst_addr if no zero)`
14	jpc	jump if carry	`jpc inst_addr   (pc=inst_addr if carry)`
15	jnc	jump if no carry	`jnc inst_addr   (pc=inst_addr if no carry)`
16	csr	call subrutine	`csr inst_addr   (pc=inst_addr) save pc+1 in stack`
17	ret	return subrutine	`ret             (pc=value stored in stack)`
18	adi	add with imm	`add rd,imm      (rd=rd+imm)`
19	csz	csr if zero	`csr inst_addr   (pc=inst_addr if zero) affects stack`
20	cnz	csr if no zero	`csr inst_addr   (pc=inst_addr if no zero) affects stack`
21	csc	csr if carry	`csr inst_addr   (pc=inst_addr if carry) affects stack`
22	cnc	csr if no carry	`csr inst_addr   (pc=inst_addr if no carry) affects stack`
23	sl0	shift left zero fill	`sl0 rd          (rd <= {rd[6:0],0})`
24	sl1	shift left one fill	`sl1 rd          (rd <= {rd[6:0],1})`
25	sr0	shift right zero fill	`sr0 rd          (rd <= {0,rd[7:1]})`
26	sr1	shiftright one fill	`sr1 rd          (rd <= {1,rd[7:1])`
27	rrl	rotary register left	`rrl rd          (rd <= {rd[6:0],rd[7]})`
28	rrr	rotay register right	`rrr rd          (rd <= {rd[0],rd[7:1])`
29	not	logic not	`sub rd,rs       (rd=rd-rs)`
30	nop	no operation	`no operation    (take 3 clk )`

Table 3. Detailed Instruction Set

5. Assembler script

The assembler script was written in *python*, you can download *python* for any operating system from: www.python.org. How to use the script from the console is as follows:



```
natalius
xc3s1600e-4fg320
  pong_top_level (pong_top_level.v)
    processor - natalius_processor (natalius_processor.v)
      control_unit_i - control_unit (control_unit.v)
      data_path_i - data_path (data_path.v)
        registros - regfile (regfile.v)
        alui - ALU (ALU.v)
        shif_reg - shiftbyte (shiftbyte.v)
        LIFOi - LIFO (LIFO.v)
      inst_mem - instruction_memory (instruction_memory.v)
    ram_memory - memram (memram.v)
    video_mem - mem_video (mem_video.v)
    video_cntrl - vga_control (vga_control.v)
    pines.ucf
```

Figure 2. Source files hierarchy for Natalius processor and reference design

```
./assembler.py code.asm (linux)
python assembler.py code.asm (win cmd)
```

Where "code.asm" is the file that will be assembled. The output file is named: " instructions.mem", **and it must be included on the same route where the source codes are**, because the hardware description of the instruction memory (instruction_memory.v) is needed.

It is also possible an assembled view line by line, this option is possible by adding the flag [-s] as shown below:

```
./assembler.py -s code.asm (linux)
python assembler.py -s code.asm (win cmd)
```

5.1 Example

Consider the following code:

```
           csr negro
end        jmp end

negro      stm r0,32
           ldi r1,80
           ldi r2,60
           ldi r3,0
           ldi r4,0
nextc      cmp r4,r1
           jpz inc_fil
           stm r3,64
           stm r4,32
           adi r4,1
           jmp nextc
inc_fil    ldi r4,0
           cmp r3,r2
           jpz fneg
           adi r3,1
           jmp nextc
fneg       ret
```

The above code is saved in the file "example.asm". When running the line:
./assembler.py -s example.asm

the console displays the following:

```
addr inst asm
0    8002 csr 2
1    5801 jmp 1
2    2020 stm r0,32
3    1150 ldi r1,80
4    123C ldi r2,60
5    1300 ldi r3,0
6    1400 ldi r4,0
7    2C20 cmp r4,r1
8    600D jpz 13
9    2340 stm r3,64
10   2420 stm r4,32
11   9401 adi r4,1
12   5807 jmp 7
13   1400 ldi r4,0
14   2B40 cmp r3,r2
15   6012 jpz 18
16   9301 adi r3,1
17   5807 jmp 7
18   8800 ret
```

6. Reference Design:

The reference design is the implementation of the classic video game (pong) using the processor Natalius. The architecture of this design is shown in Figure 3, a photo of the game in action is shown in Figure 4.

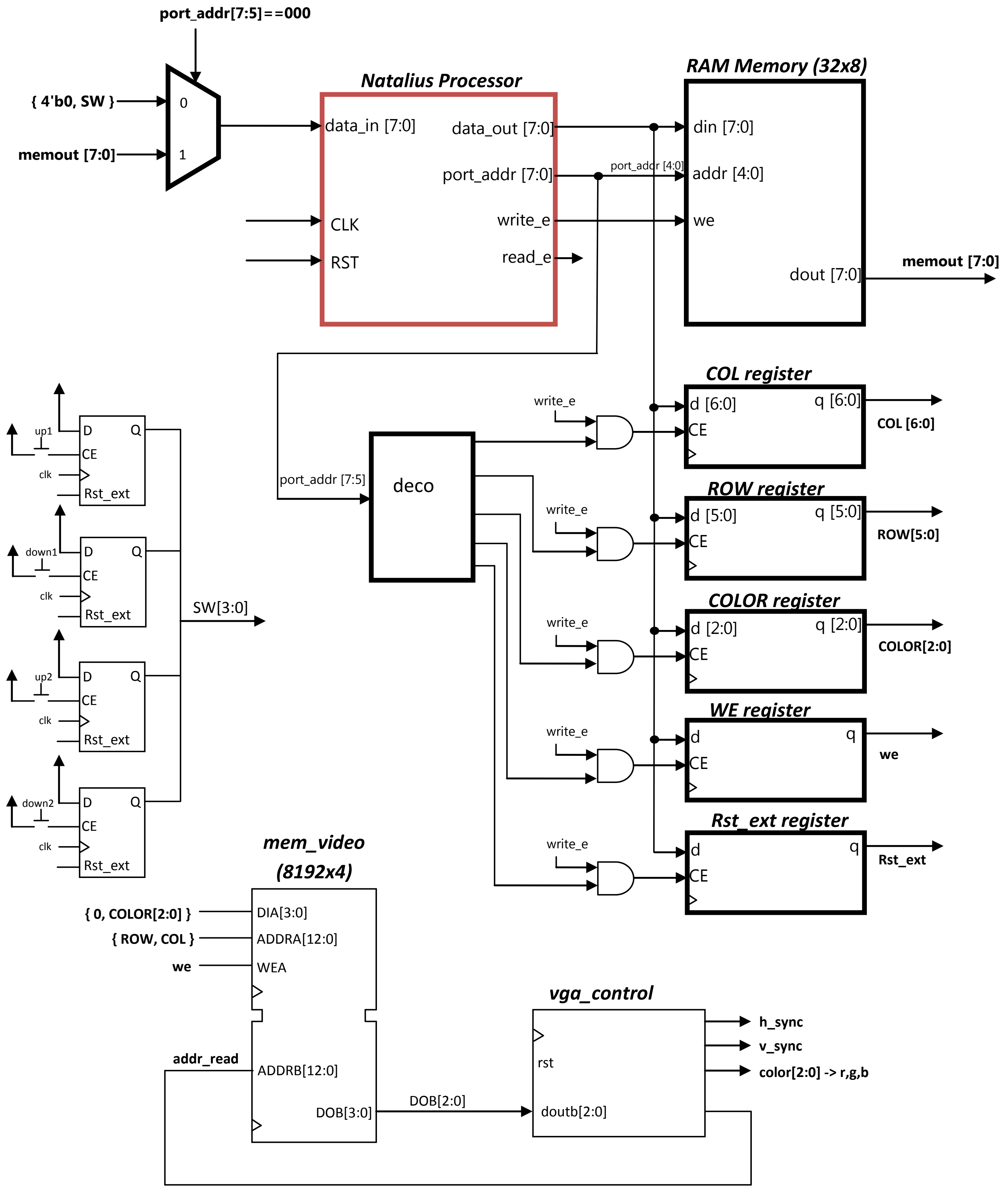


Figure 3. Hardware architecture of reference design (pong video game)

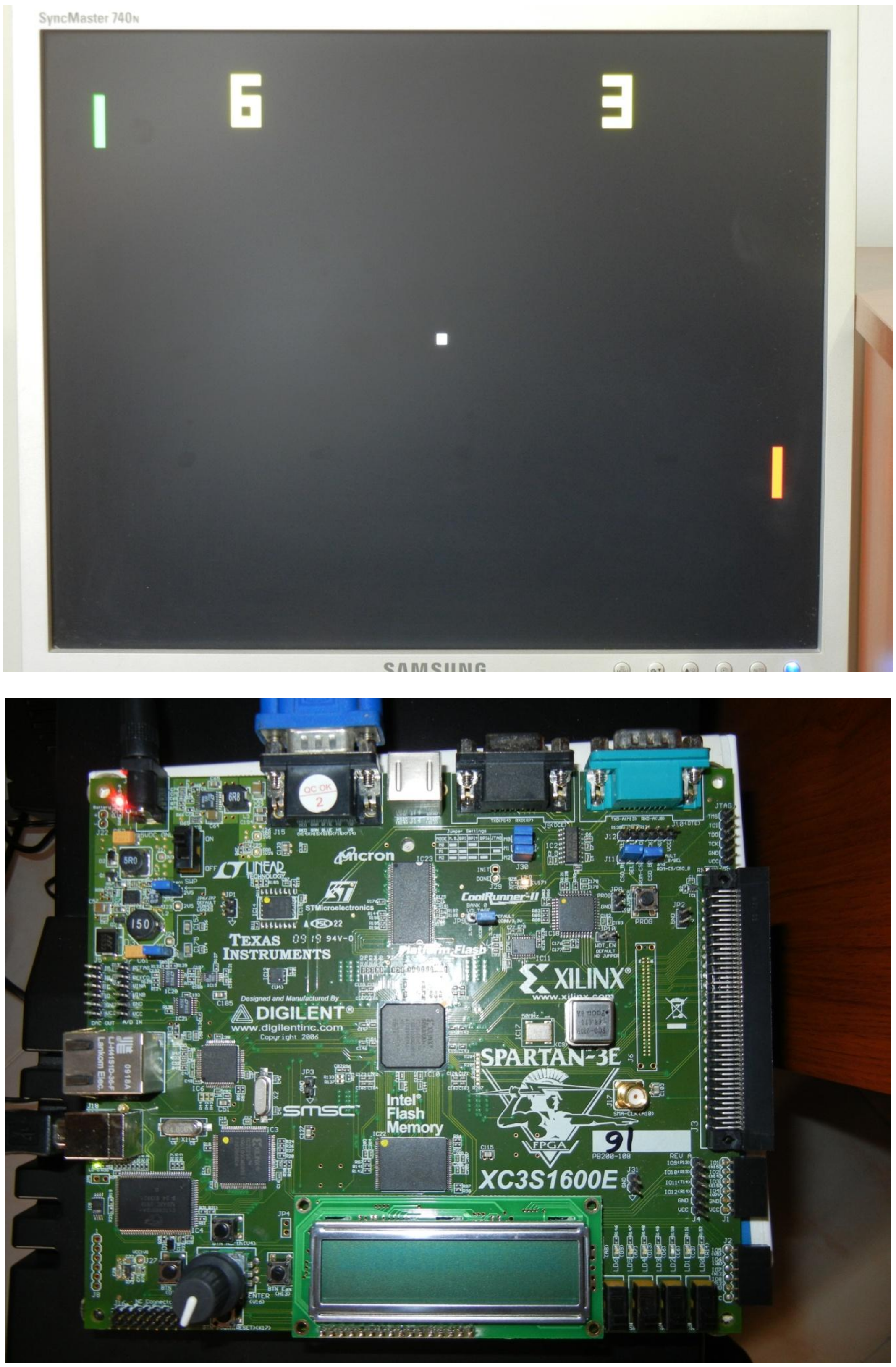


Figure 4. Reference design implementation on Spartan 3E 1600E Starter Kit Board

ANNEX (Pong assembler code)

```
          ldi r0,0
          csr negro
          ldi r1,25
          stm r1,21
          stm r1,22
          ldi r1,5
          stm r1,11
          ldi r1,75
          stm r1,12
          stm r0,5
          stm r0,6
          ldi r1,1
          stm r0,8
          stm r1,9
punto     ldi r1,40
          stm r1,10
          ldi r1,30
          stm r1,20
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
          csr dbl
inicio    csr marca
          csr vermarc
          csr negro
          csr marca
          ldm r2,21
          ldm r5,32
          ldi r6,1
          and r6,r5
          ldi r7,2
          and r7,r5
          sr0 r7
          csr moli
          stm r2,21
          ldm r2,22
          ldm r5,32
          ldi r6,4
          and r6,r5
          sr0 r6
          sr0 r6
          ldi r7,8
          and r7,r5
          sr0 r7
          sr0 r7
          sr0 r7
          csr moli
          stm r2,22
          csr cobo
          csr mobo
          ldm r1,11
          ldm r2,21
          ldi r4,7
          csr lineav
          ldm r1,12
          ldm r2,22
          ldi r4,7
          csr lineav
          csr bola
          csr delay
          csr delay
          csr delay
          csr delay
          ldi r7,1
          stm r7,128
          stm r0,128
          jmp inicio
delay     ldi  r1,0
          ldi r2,0
          ldi r3,255
pat04     cmp  r1,r3
          jpz pat03
pat02     cmp  r2,r3
          jpz pat01
          adi r2,1
          jmp pat02
```

```
pat01      adi r1,1
           ldi r2,0
           jmp pat04
pat03      ret
bola       ldm r1,10
           ldm r2,20
           ldi r4,7
           stm r1,32
           stm r2,64
           stm r4,96
           csr we
           ret
mobo       ldm r1,10
           ldm r2,20
           ldm r3,8
           ldm r4,9
           ldi r5,1
           cmp r3,r5
           jpz comp12
           sub r2,r5
           jmp comp13
comp12     adi r2,1
comp13     cmp r4,r5
           jnz comp14
           adi r1,1
           jmp comp15
comp14     sub r1,r5
comp15     stm r1,10
           stm r2,20
           ret
cobo       ldm r1,10
           ldm r2,20
           ldi r3,78
           ldi r4,58
           ldi r7,2
           cmp r1,r7
           jnz comp04
           ldm r6,6
           adi r6,1
           stm r6,6
           ldi r6,1
           stm r6,9
           jmp punto
comp04     cmp r1,r3
           jnz comp05
           ldm r6,5
```

```
           adi r6,1
           stm r6,5
           stm r0,9
           jmp punto
comp05     cmp r2,r7
           jnz comp06
           ldi r6,1
           stm r6,8
comp06     cmp r2,r4
           jnz comp07
           stm r0,8
comp07     ldm r3,11
           ldm r4,21
           adi r3,1
           cmp r1,r3
           jnz comp08
           cmp r2,r4
           jpz comp09
           adi r4,1
           cmp r2,r4
           jpz comp09
           adi r4,1
           cmp r2,r4
           jpz comp09
           adi r4,1
           cmp r2,r4
           jpz comp09
           adi r4,1
           cmp r2,r4
           jpz comp09
           adi r4,1
           cmp r2,r4
           jnz comp08
comp09     ldi r6,1
           stm r6,9
comp08     ldm r3,12
           ldi r6,1
           ldm r4,22
           sub r4,r6
           cmp r1,r3
           jnz comp10
           cmp r2,r4
           jpz comp11
           adi r4,1
           cmp r2,r4
           jpz comp11
```

```
          adi r4,1
          cmp r2,r4
          jpz comp11
          adi r4,1
          cmp r2,r4
          jpz comp11
          adi r4,1
          cmp r2,r4
          jpz comp11
          adi r4,1
          cmp r2,r4
          jnz comp10
comp11    stm r0,9
comp10    ret
moli      ldi r3,1
          ldi r4,55
          ldi r5,2
          cmp r6,r3
          jnz comp03
          cmp r2,r3
          jpz finmol
          sub  r2,r5
comp03    cmp r7,r3
          jnz finmol
          cmp r2,r4
          jpz finmol
          add r2,r5
finmol    ret
marca     ldm  r5,5
          ldi r1,19
          ldi r2,5
          ldi r4,6
          csr impnum
          ldm r5,6
          ldi r1,57
          ldi r2,5
          csr impnum
          ret
vermarc   ldm r1,5
          ldm r2,6
          ldi r3,6
          cmp r1,r3
          jnz comp01
          ldm r1,12
          ldm r2,22
          ldi r4,4
```

```
          csr lineav
gana1     jmp gana1
comp01    ldi r3,6
          cmp r2,r3
          jnz comp02
          ldm r1,11
          ldm r2,21
          ldi r4,4
          csr lineav
gana2     jmp gana2
comp02    ret
lineav    ldi r3,5
          add r3,r2
con       cmp r2,r3
          jnc ter
          stm r2, 64
          stm r1, 32
          stm r4, 96
          csr we
          adi r2,1
          jmp con
ter       ret
negro     ldi r7,1
          stm r7,160
          stm r0,96
          ldi r1,80
          ldi r2,60
          ldi r3,0
          ldi r4,0
nextc     cmp r4,r1
          jpz inc_fil
          stm r3,64
          stm r4,32
          adi r4,1
          jmp nextc
inc_fil   ldi r4,0
          cmp r3,r2
          jpz fneg
          adi r3,1
          jmp nextc
fneg      stm r0,160
          ret
we        ldi r7,1
          stm r7,160
          ldi r7,0
          stm r7,160
```

```
            ret
dbl         csr delay
            csr delay
            csr delay
            ldm r1,11
            ldm r2,21
            ldi r4,2
            csr lineav
            ldm r1,12
            ldm r2,22
            ldi r4,2
            csr lineav
            csr bola
            ret
segh        ldi r3,3
            add r3,r1
pon1        cmp r1,r3
            jpz mer1
            stm r2, 64
            stm r1, 32
            stm r4, 96
            csr we
            adi r1,1
            jmp pon1
mer1        ldi r3,3
            sub r1,r3
            ret
segv        ldi r3,3
            add r3,r2
pon2        cmp r2,r3
            jpz mer2
            stm r2, 64
            stm r1, 32
            stm r4, 96
            csr we
            adi r2,1
            jmp pon2
mer2        ldi r3,3
            sub r2,r3
            ret
sega        csr segh
            ret
segb        ldi r7,2
            add r1,r7
            csr segv
            ldi r7,2
            sub r1,r7
            ret
segc        ldi r7,2
            add r1,r7
            add r2,r7
            csr segv
            ldi r7,2
            sub r1,r7
            sub r2,r7
            ret
segd        ldi r7,4
            adi r2,4
            csr segh
            ldi r7,4
            sub r2,r7
            ret
sege        ldi r7,2
            adi r2,2
            csr segv
            ldi r7,2
            sub r2,r7
            ret
segf        csr segv
            ret
segg        ldi r7,2
            adi r2,2
            csr segh
            ldi r7,2
            sub r2,r7
            ret
impnum      ldi r7,1
            cmp r5,r7
            jpz num01
            ldi r7,4
            cmp r5,r7
            jpz num01
            csr sega
num01       ldi r7,5
            cmp r5,r7
            jpz num02
            ldi r7,6
            cmp r5,r7
            jpz num02
            csr segb
num02       ldi r7,2
            cmp r5,r7
```

```
          jpz num03
          csr segc
num03     ldi r7,1
          cmp r5,r7
          jpz num04
          ldi r7,4
          cmp r5,r7
          jpz num04
          ldi r7,7
          cmp r5,r7
          jpz num04
          csr segd
num04     ldi r7,0
          cmp r5,r7
          jpz num05
          ldi r7,2
          cmp r5,r7
          jpz num05
          ldi r7,6
          cmp r5,r7
          jpz num05
          ldi r7,8
          cmp r5,r7
          jnz num06
num05     csr sege
num06     ldi r7,1
          cmp r5,r7
          jpz num07
          ldi r7,2
          cmp r5,r7
          jpz num07
          ldi r7,3
          cmp r5,r7
          jpz num07
          ldi r7,7
          cmp r5,r7
          jpz num07
          csr segf
num07     ldi r7,0
          cmp r5,r7
          jpz num08
          ldi r7,1
          cmp r5,r7
          jpz num08
          csr segg
num08     ret
```