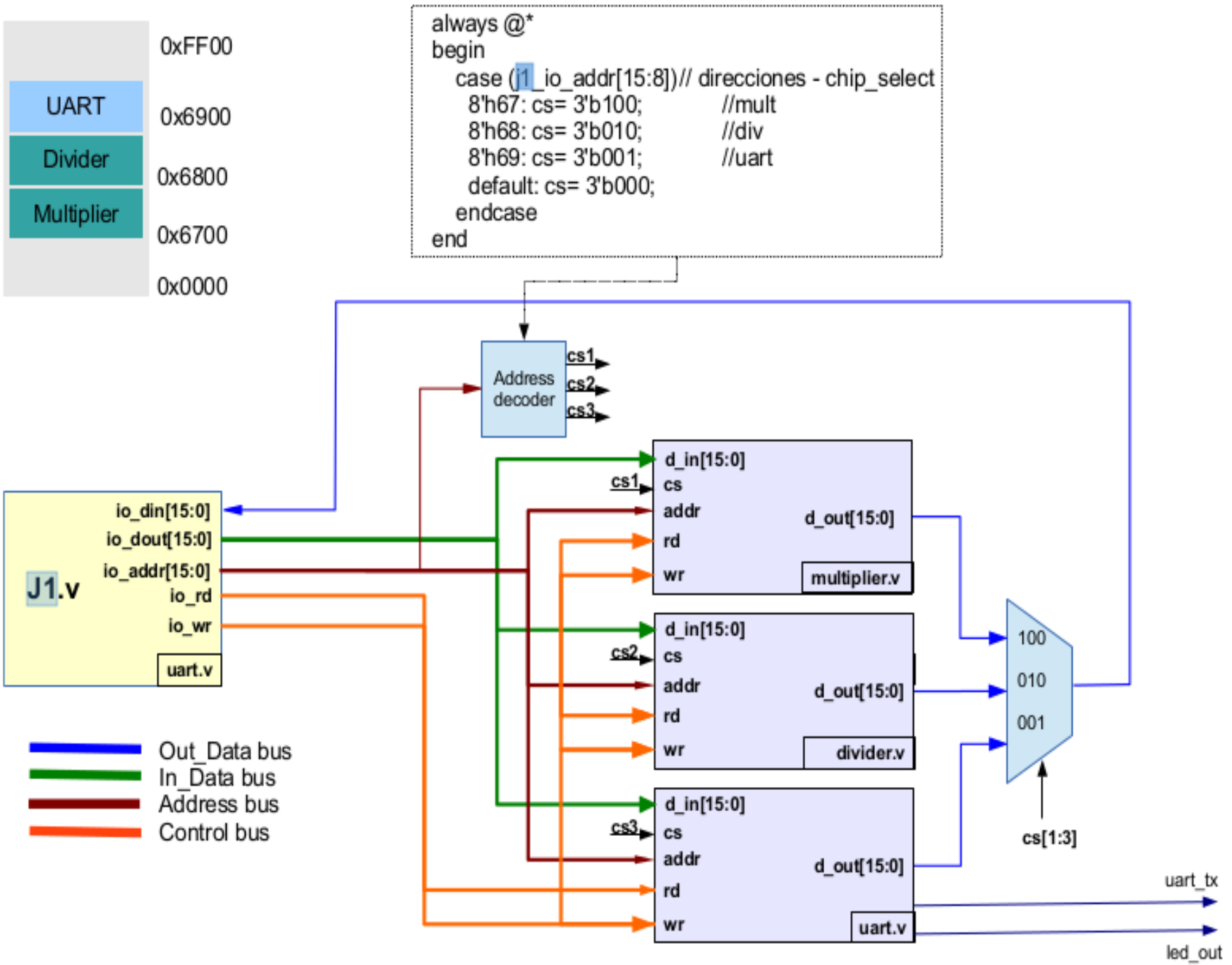
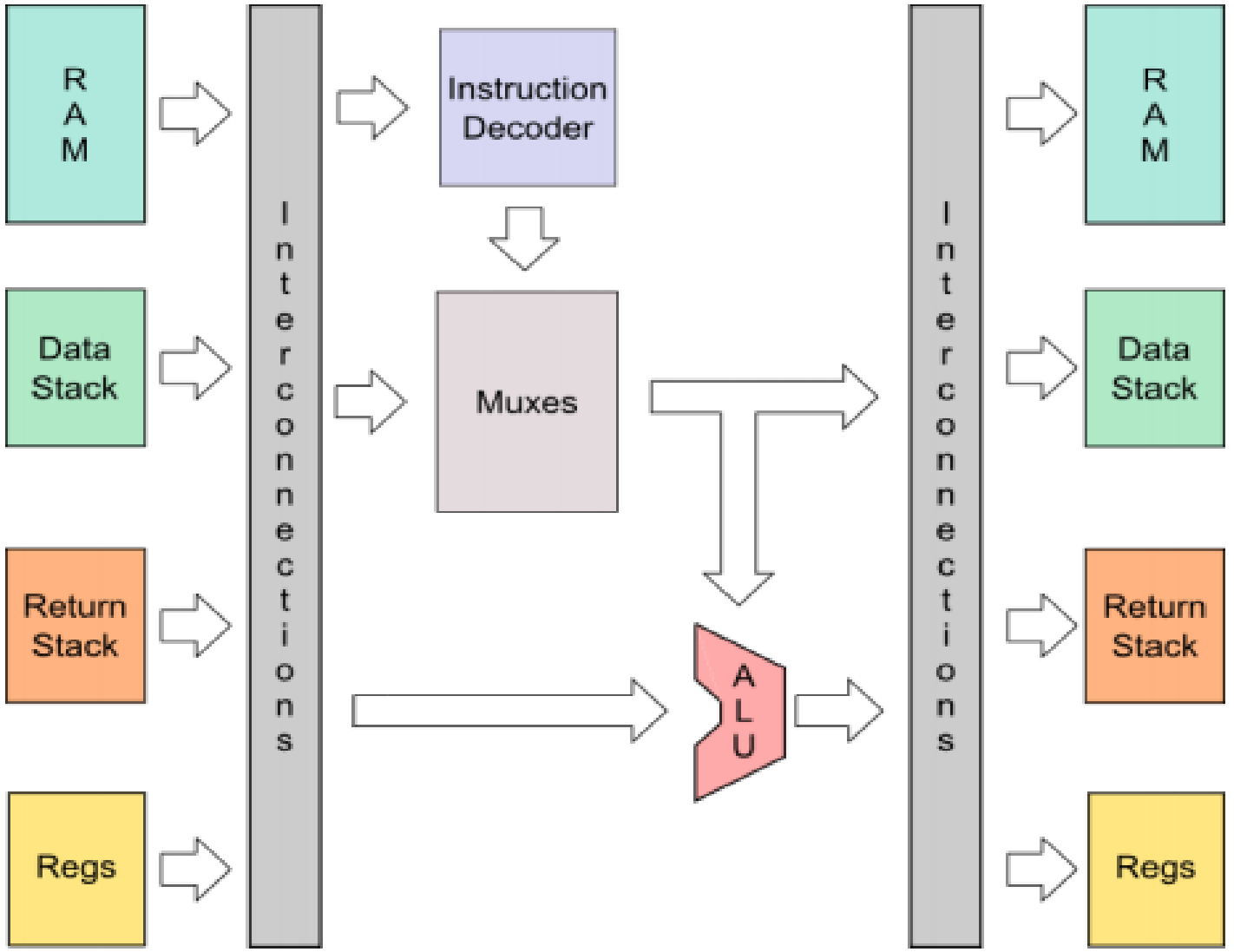
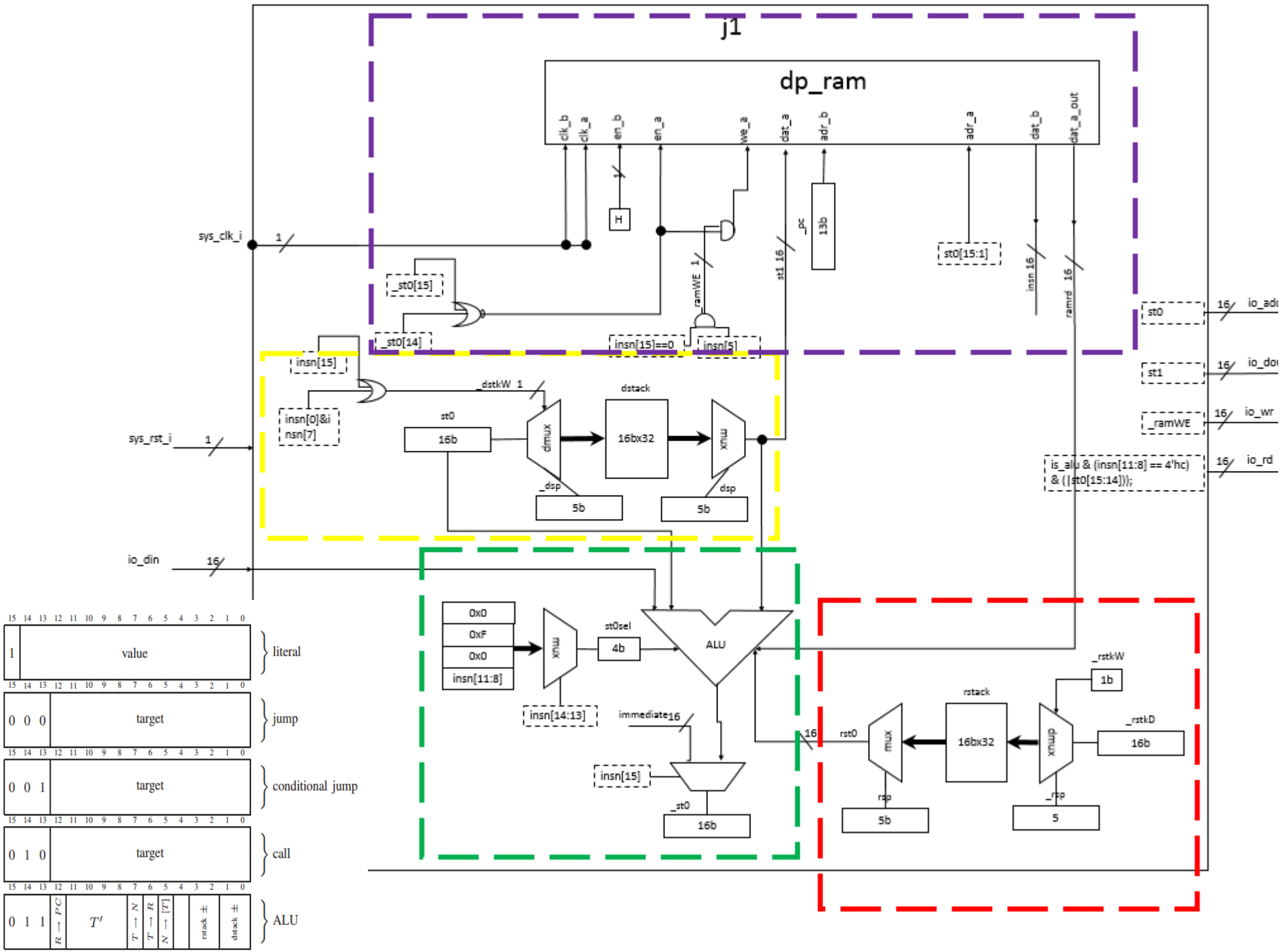

0xFF00
UART
0x6900
Divider
0x6800
Multiplier
0x6700
0x0000
always @*
begin
case (j1_io_addr[15:8])// direcciones - chip_select
8'h67: cs= 3'b100; //mult
8'h68: cs= 3'b010; //div
8'h69: cs= 3'b001; //uart
default: cs= 3'b000;
endcase
end
Address decoder
cs1
cs2
cs3
J1.v
io_din[15:0]
io_dout[15:0]
io_addr[15:0]
io_rd
io_wr
uart.v
d_in[15:0]
cs
addr
rd
wr
d_out[15:0]
multiplier.v
divider.v
uart.v
100
010
001
cs[1:3]
uart_tx
led_out
Out_Data bus
In_Data bus
Address bus
Control bus

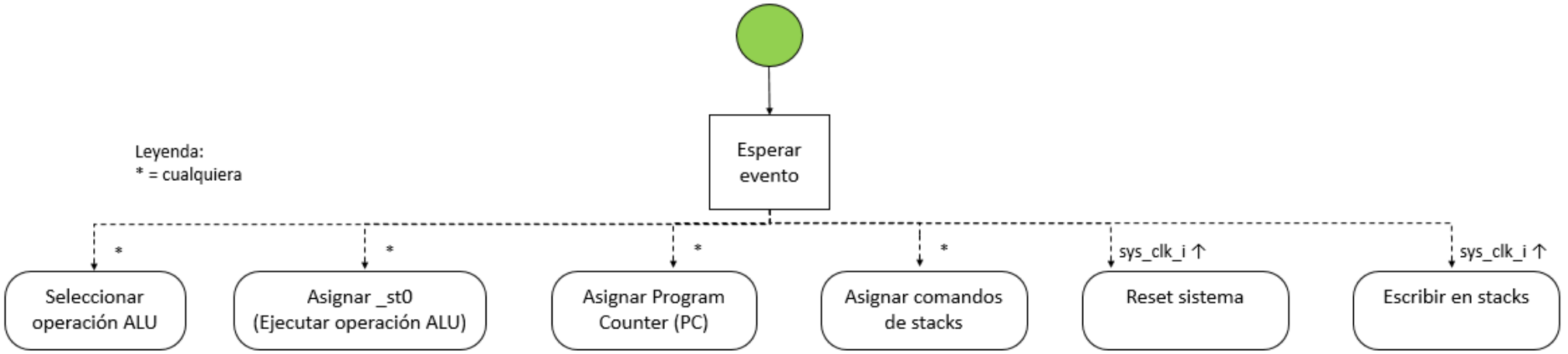
Diagrama Funcional



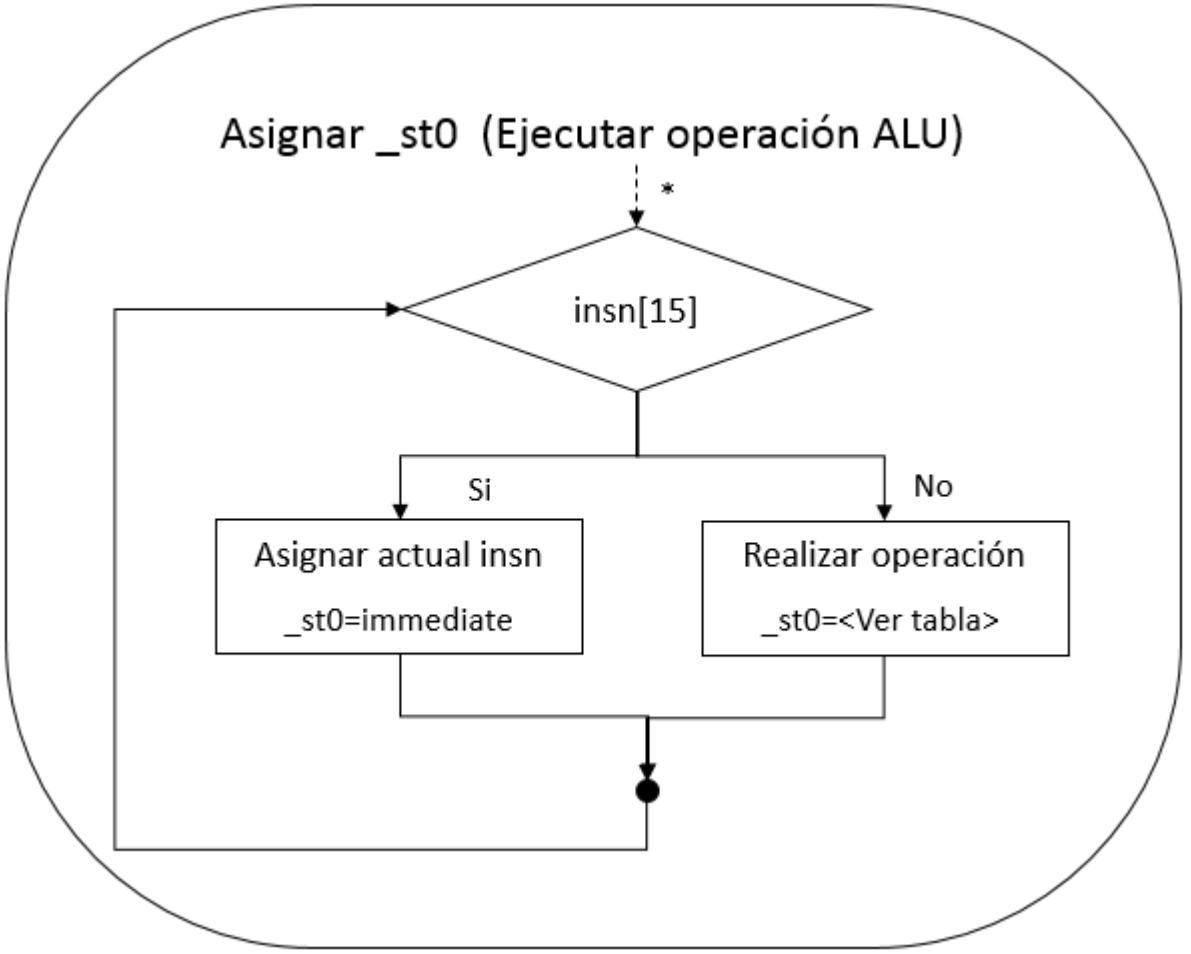
Reset and Start Signal*


j1
dp_ram
clk_b
clk_a
en_b
en_a
we_a
dat_a
adr_b
adr_a
dat_b
dat_a_out
H
_pc
13b
st0[15:1]
insn 16
ramrd 16
st1 16
ramWE 1
sys_clk_i
1
_st0[15]
_st0[14]
insn[15]==0
insn[5]
st0
st1
_ramWE
is_alu & (insn[11:8] == 4'hc) & (|st0[15:14]);
io_adr
io_dout
io_wr
io_rd
insn[15]
insn[0]&insn[7]
_dstkW 1
dstack
st0
16b
dmux
16bx32
xmux
_dsp
dsp
5b
sys_rst_i
io_din
16
0x0
0xF
0x0
insn[11:8]
mux
st0sel
4b
ALU
insn[14:13]
immediate16
insn[15]
_st0
16b
_rstkW
1b
rstack
_rstkD
16b
rst0
rsp
_rsp
5b
5
literal
jump
conditional jump
call
ALU
value
target
$R \rightarrow PC$
T'
$T \rightarrow N$
$T \rightarrow R$
$N \rightarrow [T]$
rstack ±
dstack ±

Macroprocesos de la Arquitectura

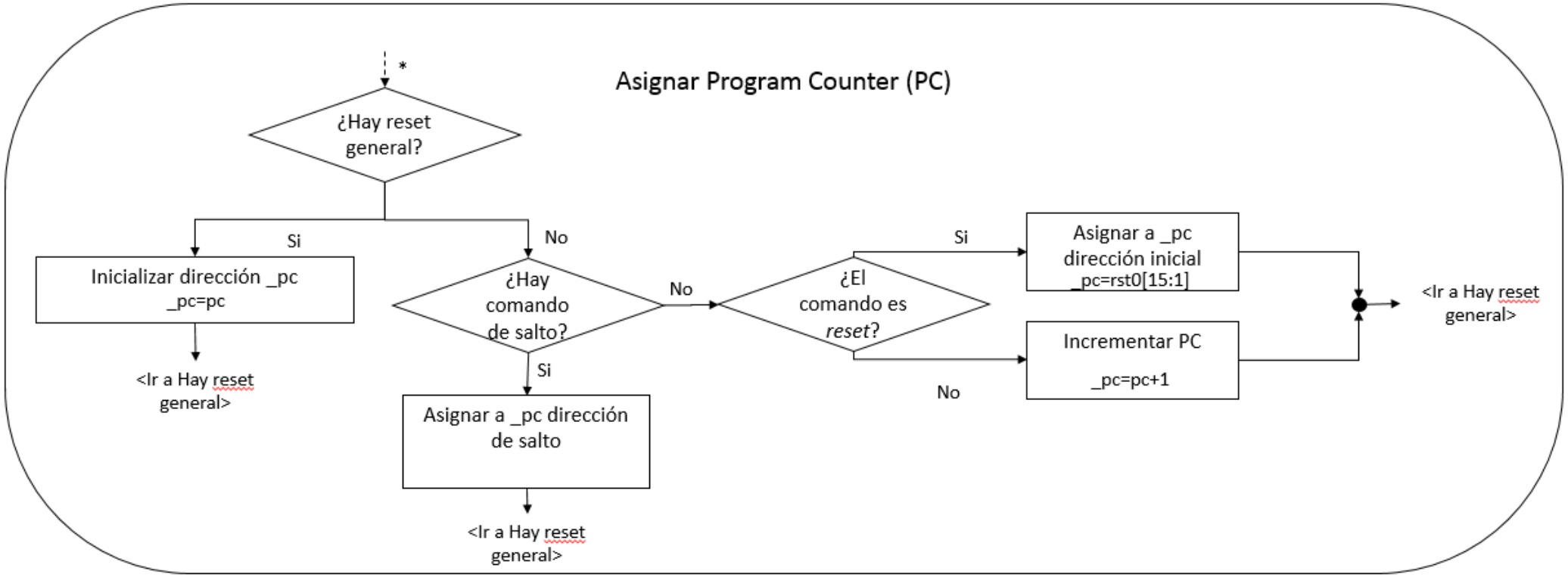


Ejecutar operación ALU

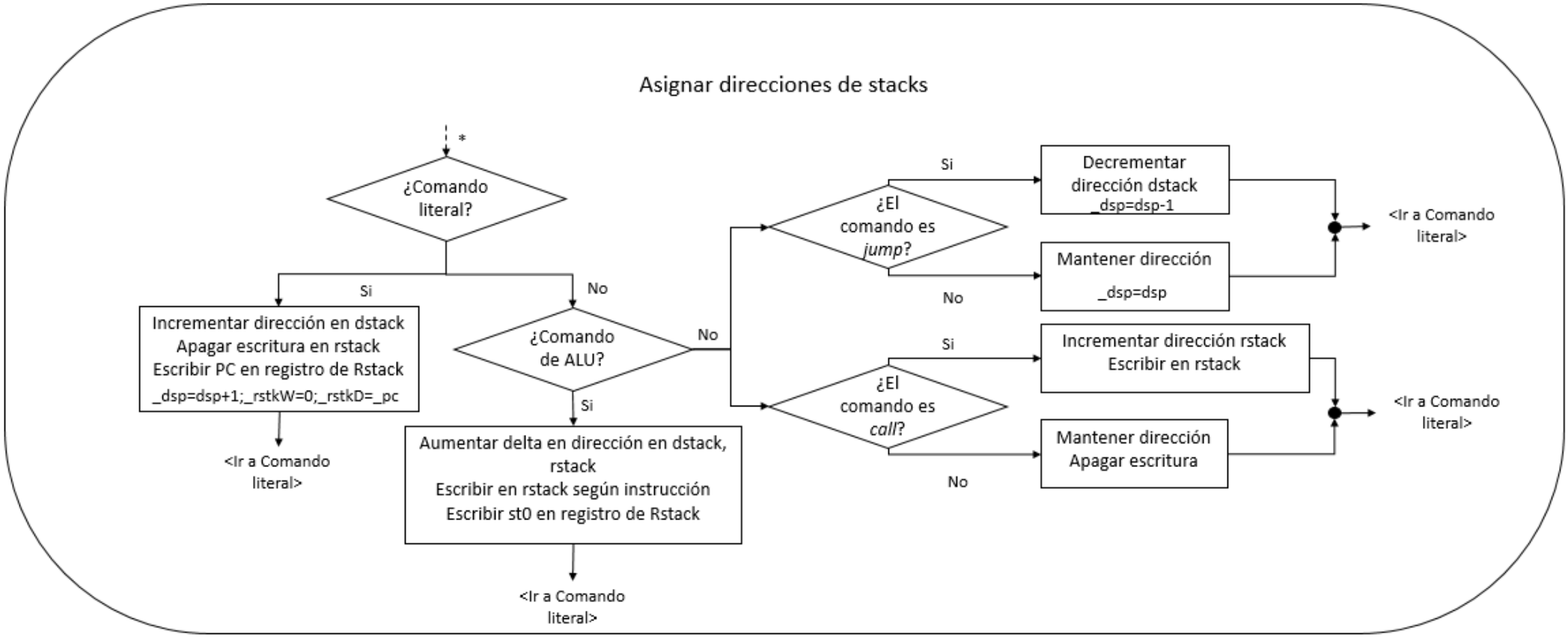


code	operation
0	T
1	N
2	$T + N$
3	$T \texttt{and} N$
4	$T \texttt{or} N$
5	$T \texttt{xor} N$
6	$\sim T$
7	$N = T$
8	$N < T$
9	$N \texttt{rshift} T$
10	$T - 1$
11	R
12	$[T]$
13	$N \texttt{lshift} T$
14	depth
15	$N \texttt{u<} T$

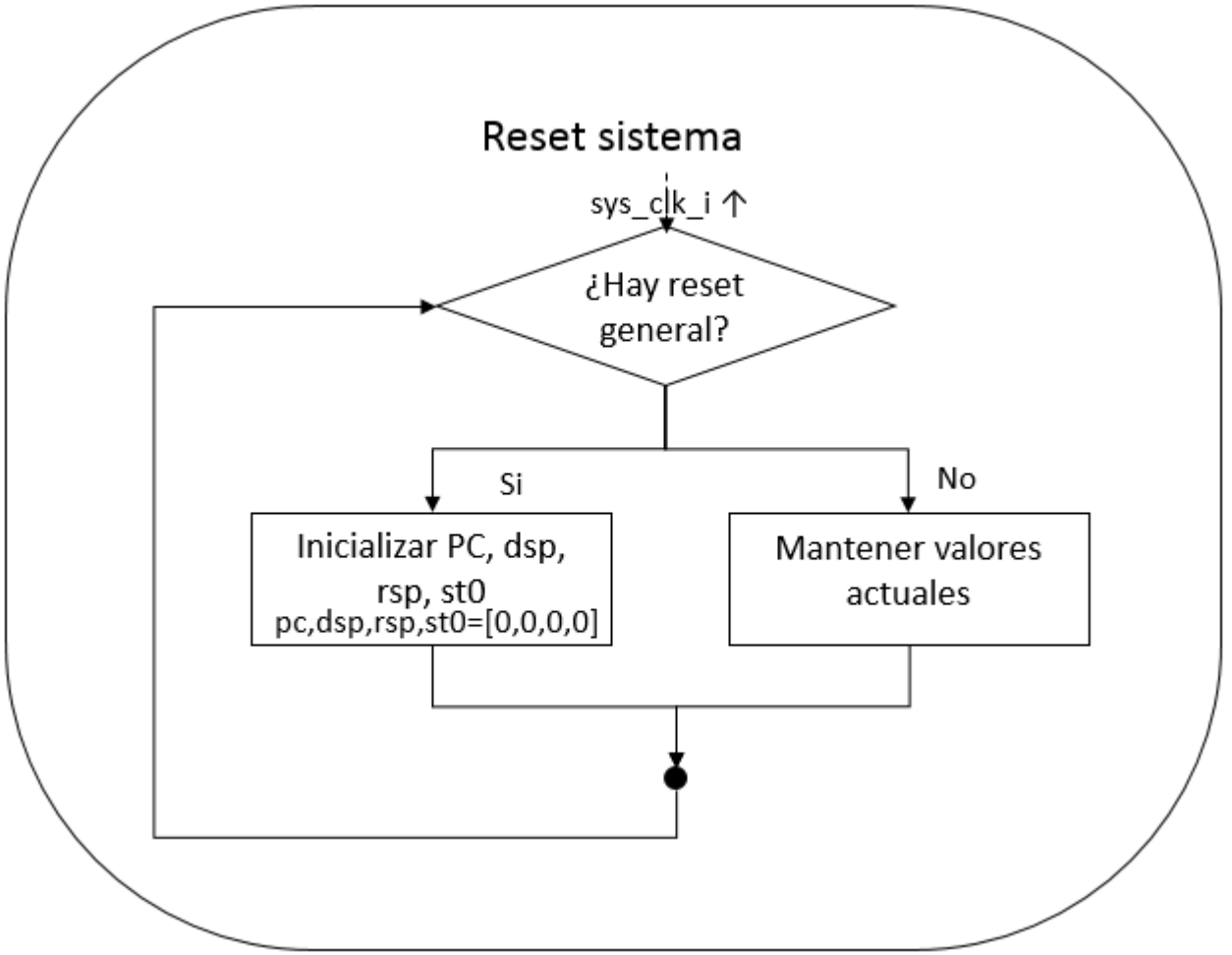
Asignar Program Counter



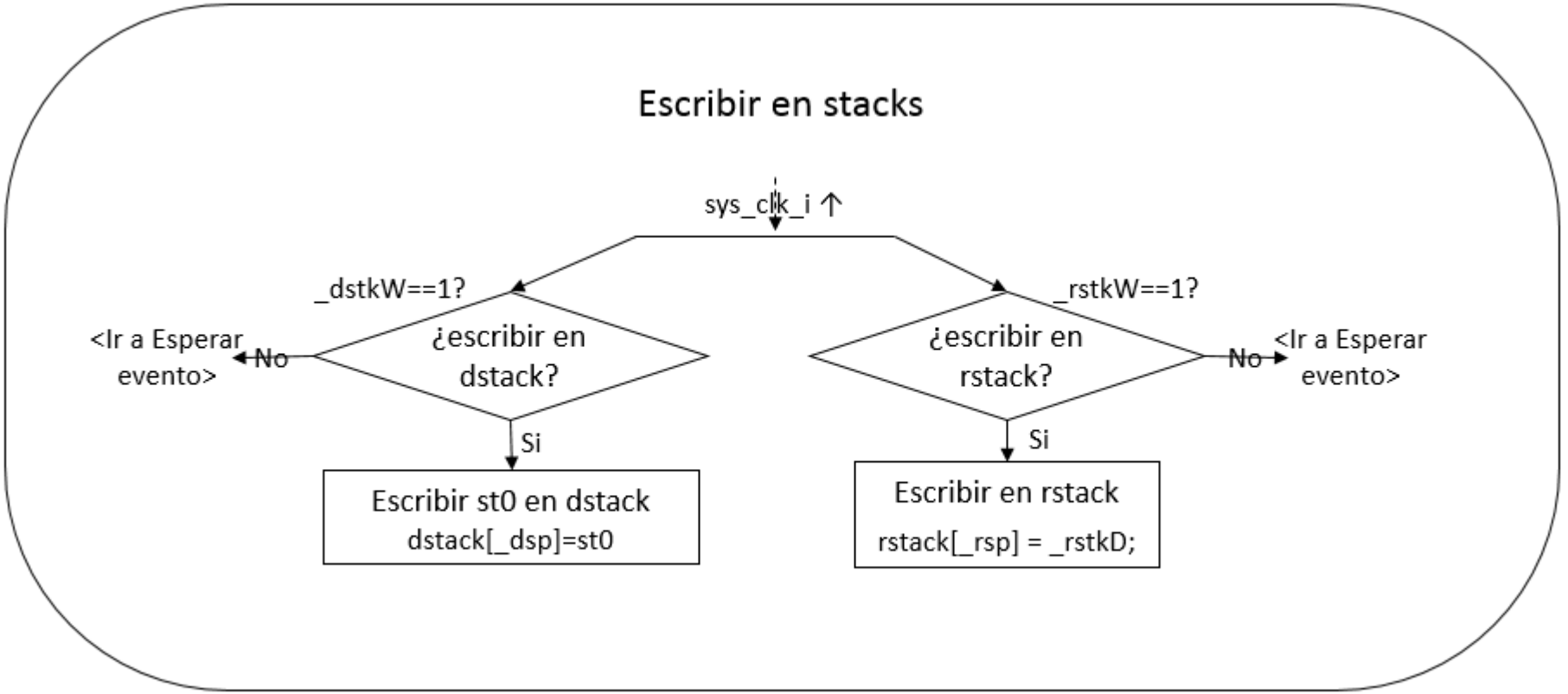
Asignar direcciones de stacks



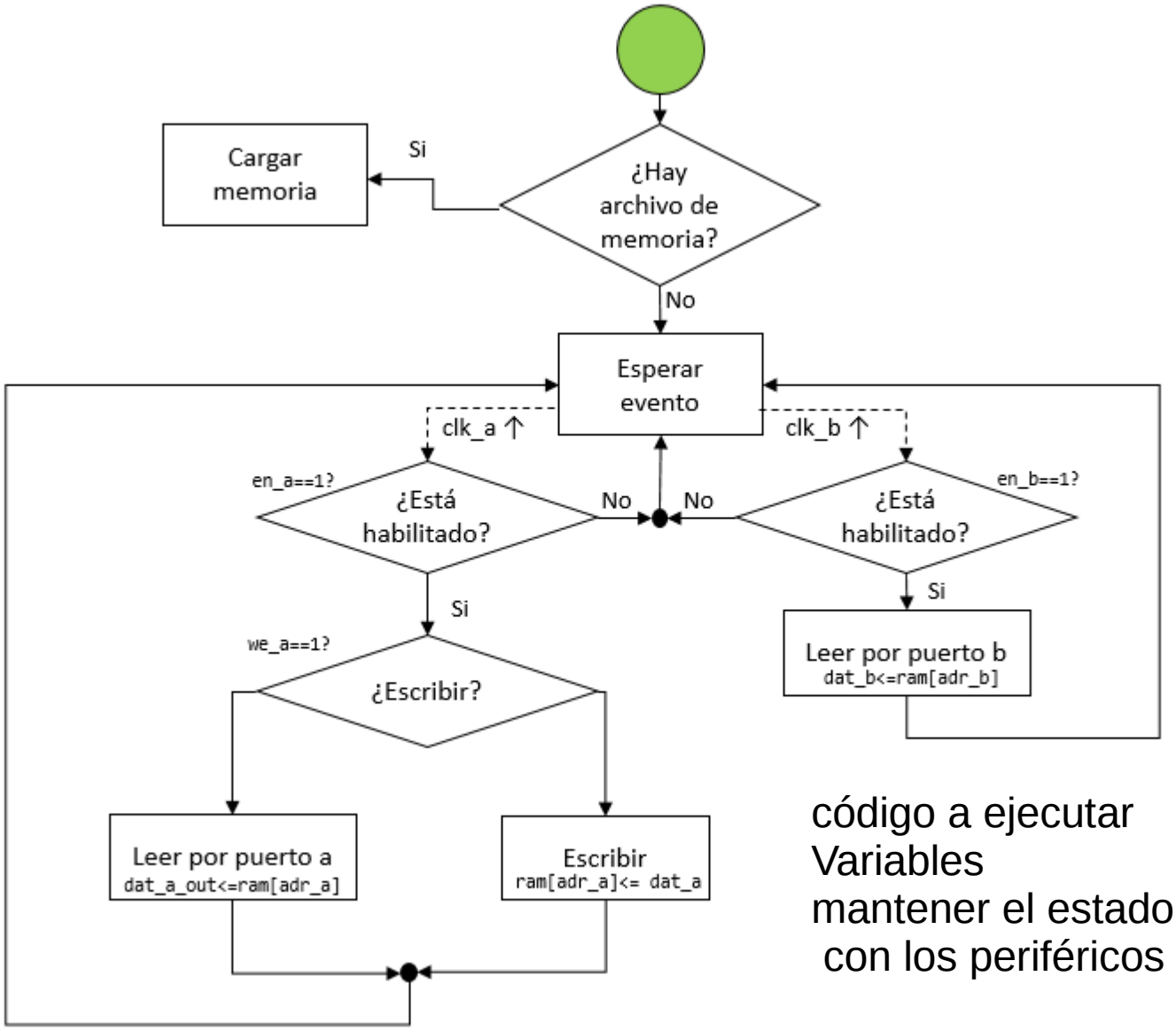
Reset del sistema

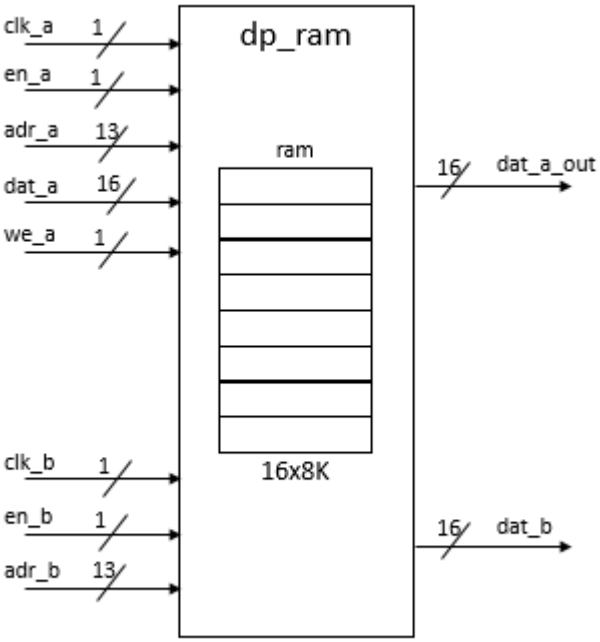


Escribir en Stacks



Dual port ram

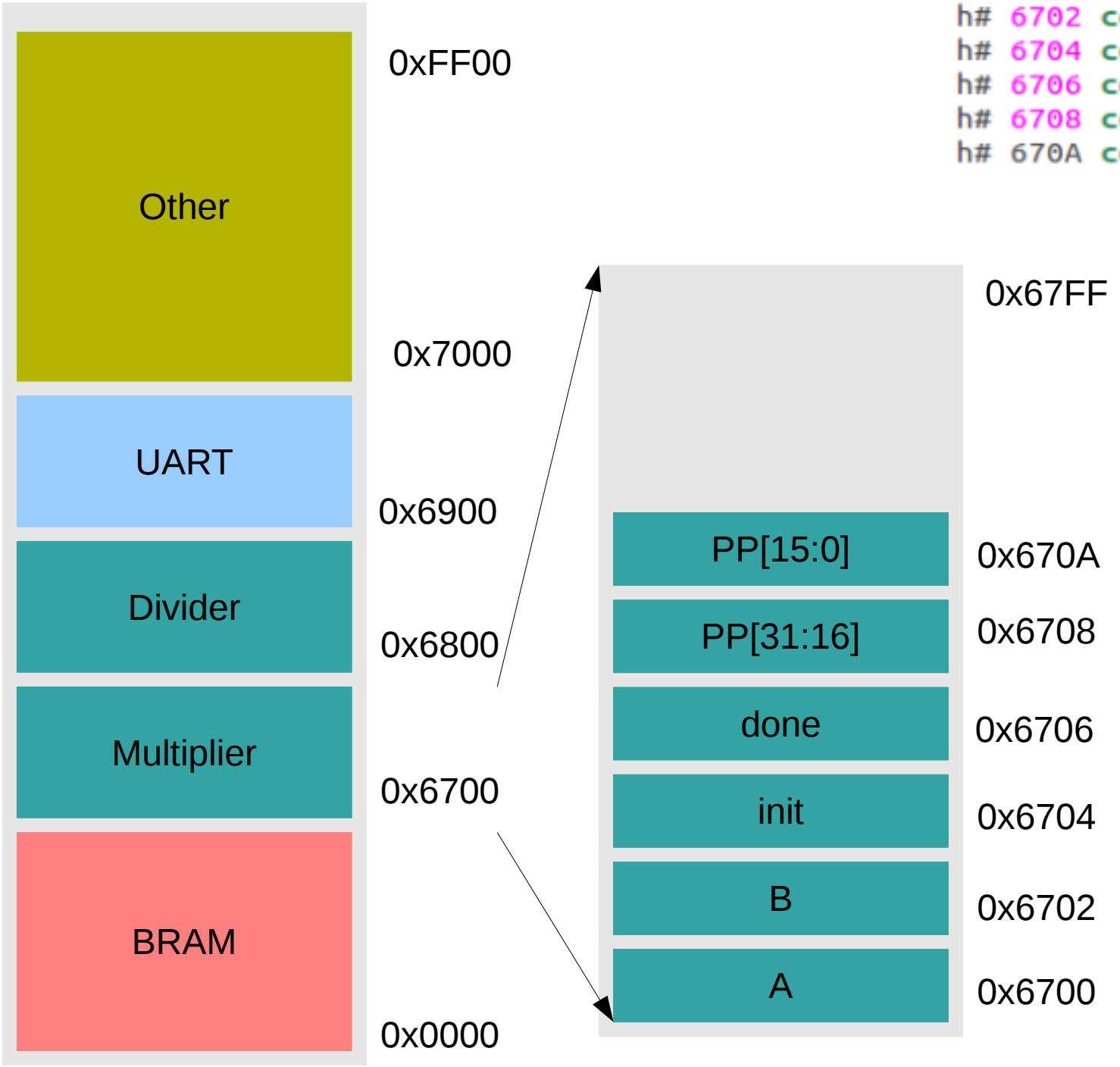




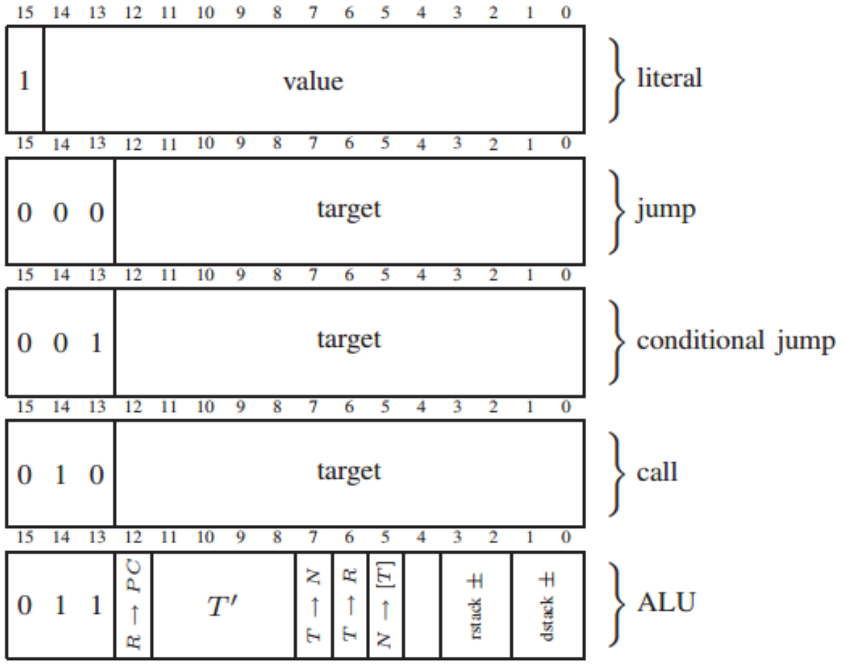
código a ejecutar
Variables
mantener el estado de registros
con los periféricos de la arquitectura.

El puerto A permite lectura y escritura, mientras que el puerto B sólo permite lectura

Memory_map J1_soc




```
\ memory map multiplier:
h# 6700 constant multi_a
h# 6702 constant multi_b
h# 6704 constant multi_init
h# 6706 constant multi_done
h# 6708 constant multi_pp_high
h# 670A constant multi_pp_low
```



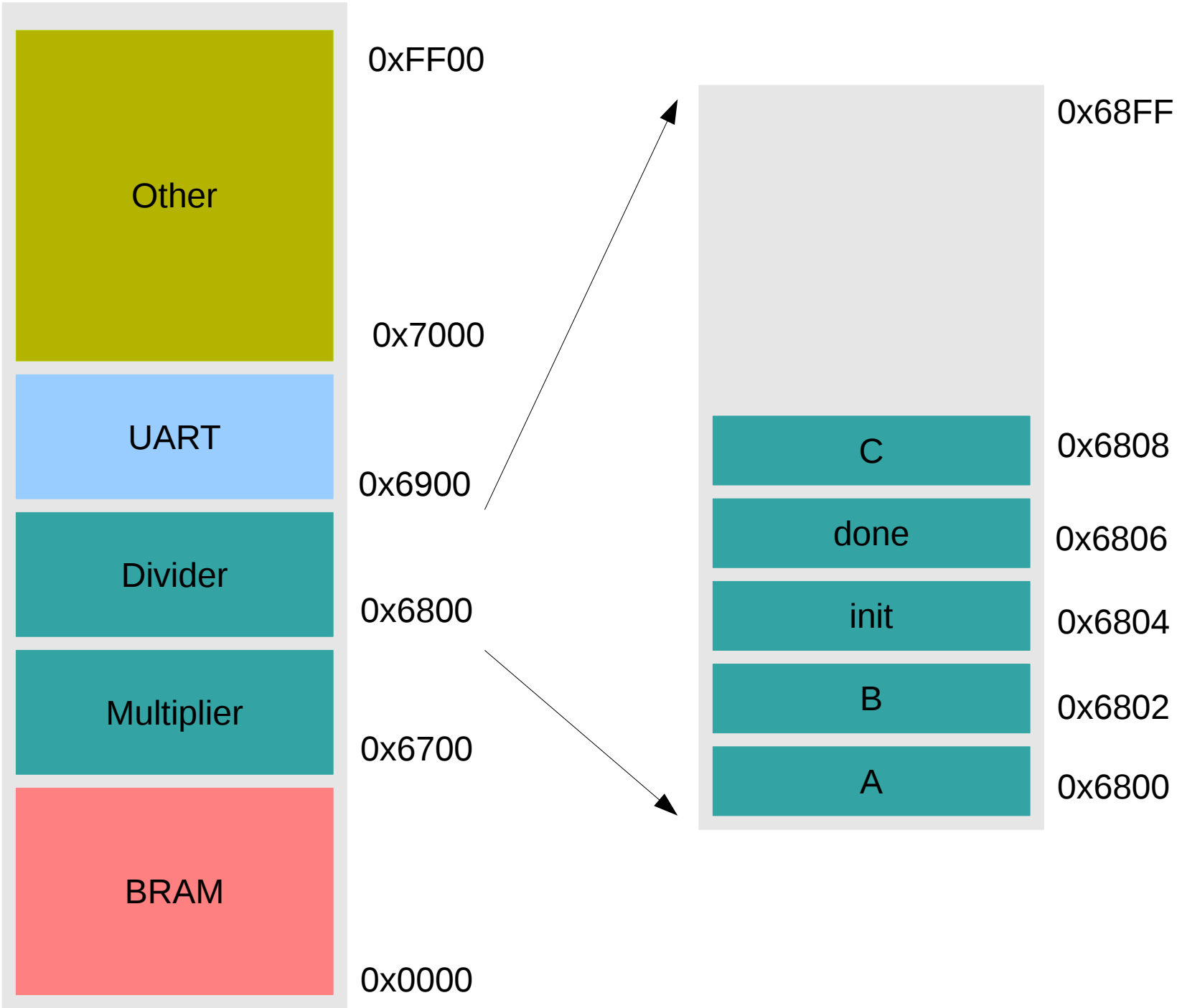

```
1 : multiplicar            \ utiliza peripheral_mult.v
2 swap multi_a !
3 multi_b !
4 d# 1 multi_init !
5 begin multi_done @ d# 1 = until \ bucle while
6 multi_pp_high @
7 multi_pp_low @
8 ;
9
10
11 : main
12
13 d# 5 d# 3 multiplicar
```

```
0924 8005   LIT $5
0926 8003   LIT $3
0928 4443   CALL multiplicar
092A 8008   LIT $8
092C 8002   LIT $2
092E 0457   BRANCH dividir
```

```
\ multiplicar
0886 0816   ALU $6180
0888 E700   LIT $6700
088A 6023   ALU $6023
088C 6103   ALU $6103
088E E702   LIT $6702
0890 6023   ALU $6023
0892 6103   ALU $6103
0894 8001   LIT $1
0896 E704   LIT $6704
0898 6023   ALU $6023
089A 6103   ALU $6103
```

```
\ (begin)
089C E706   LIT $6706
089E 6C00   ALU $6C00
08A0 8001   LIT $1
08A2 6703   ALU $6703
08A4 244E   0BRANCH (begin)
08A6 E708   LIT $6708
08A8 6C00   ALU $6C00
08AA E70A   LIT $670A
08AC 7C0C   ALU $7C0C
```

Memory_map J1_soc



Memory_map J1_soc