ANEXO A-1

Descrição HDC

```
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* |                                                                   | */
/* |                          +---------+                              | */
/* |                          |  Risco  |                              | */
/* |                          +---------+                              | */
/* |                                                                   | */
/* |              Microprocessador RISC CMOS 32 bits                   | */
/* |                                                                   | */
/* |              Autores: Andre', Dani, Giba, Gil,                    | */
/* |                       GME, Jorge, Junqueira, Luigi,               | */
/* |                       Marcon, Paulo, Soto, Suzim.                 | */
/* |                                                                   | */
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                     Risco  [ fonte.hdc  ]                         | */
/* +-------------------------------------------------------------------+ */


/*-----------------------------------------------------------------------*/
/*              Declaracao dos Nodos do Sistema Digital                  */
/*-----------------------------------------------------------------------*/
/* PARTE DE CONTROLE pcon */
typedef struct
{
unsigned char
CONTA,       CONTOU,      t1,          t0,          cod,
codb4,       codb3,       codb2,       codb1,       codb0,
aps,         f1,          f0,          enddst,      enddstb4,
enddstb3,    enddstb2,    enddstb1,    enddstb0,    endft1,
endft1b4,    endft1b3,    endft1b2,    endft1b1,    endft1b0,

ss2,         endft2,      endft2b4,    endft2b3,    endft2b2,
endft2b1,    endft2b0,    a_ft1,       a_r0,        a_dst,
b_ft2,       b_k,         b_kp,        b_kgl,       b_kgh,
l_prox_est,  l_est_atual, s0,          s1,          m1,
m0,          m2,          m01,         m02,         ila,

isa,         ime,         isr,         ila0,        isa0,
imel,        ime2,        isr1,        isr2,        dst_eh_r0,
dst_eh_psw,  dst_eh_pc,   ft2_eh_pc,   ft2_ig_dst,  load,
store,       preinc,      posinc,      posdec,      altft2,
usa_ud,      usa_ula,     codula,      codud,       cond_v,

int_ac,      bout_bb_f1,  bout_inc_f1, pc_bout_f1,  bsis_bout_f1,
bout_pc_f1,  ale_f1,      bb_rmem_f1,  bb_ud_f1,    bb_ula_f1,
dst_bb_f1,   ft2_ba_f1,   ba_ula_f1,   wr_f1,       rd_f1,
bout_bb_f1_1, bout_inc_f1_1,           pc_bout_f1_1, bsis_bout_f1_1,          bout_pc_f1_1,
ale_f1_1,    bb_rmem_f1_1, bb_ud_f1_1, bb_ula_f1_1, dst_bb_f1_1,

ft2_ba_f1_1, ba_ula_f1_1, wr_f1_1,    rd_f1_1,     enddst_1,
endft1_1,    endft2_1,    ba_ft1_f2,  ba_r0_f2,    ba_dst_f2,
bb_ft2_f2,   bb_k_f2,     bb_r0_f2,   rub_bb_f2,   rua_ba_f2,
rula_codula_f2,           rda_ba_f2,  rdb_bb_f2,   rud_codud_f2, rd_f2,
wr_f2,       pcinc_bout_f2,           bout_pc_f2,  bout_inc_f2,  bout_bb_f2,

bb_dst_f2,   bsis_bout_f2, rd_f2_1,    wr_f2_1,     proxri_bsis_f3,
proxri_int_isr_f3,        proxri_int_noop_f3,      proxK_bsis_f3,           ri_proxri_f3,
                                                                                    K_proxK_f3,
bb_1_f3,     rmem_bsis_f3, bb_dst_f3,  bout_bb_f3,  bsis_bout_f3,
bout_pc_f3,  bout_inc_f3,  rd_f3,      wr_f3,       proxri_bsis_f3_1,
proxri_int_isr_f3_1,      proxri_int_noop_f3_1,    proxK_bsis_f3_1,
                                                                        ri_proxri_f3_1,          K_proxK_f3_1,

bb_1_f3_1,   rmem_bsis_f3_1,           bb_dst_f3_1, bout_bb_f3_1, bsis_bout_f3_1,
bout_pc_f3_1, bout_inc_f3_1,           rd_f3_1,     wr_f3_1;
```

```
unsigned long
proxri,      ri;
} parte_controle;

/* INTERFACE pint */
typedef struct
{
unsigned char
ale,         ba_dst,      ba_ft1,      ba_r0,       ba_ula,
bb_1,        bb_K,        bb_dst,      bb_ft2,      bb_r0,

bb_rmem,     bb_ud,       bb_ula,      bout_bb,     bout_inc,
bout_pc,     bsis_bout,   proxK_bsis,  K_proxK,     dst_bb,

ft2_ba,      pcinc_bout,  pc_bout,     proxri_bsis, proxri_int_isr,
proxri_int_noop,          rda_ba,      rdb_bb,      rd,          ri_proxri,

rmem_bsis,   rua_ba,      rub_bb,      rud_codud,   rula_codula,
wr;
} interface;

/* PARTE OPERATIVA po */
typedef struct
{
long unsigned
bout,        pcinc,       inc,         bb,          ba,
bsis,        rmem,        proxK,       K,           K_ext,
breg[32],    ula,         a,           b,           rua,
rub;

unsigned char
sinal_kp,    sinal_kg,    rula,        log,         arit,
and,         or,          xor,         add,         inv_a,
inv_b,       com_c,       inv_c,       cin,         sel_and,
sel_or,      sel_xor,     sel_add,     c,           cout,
overf,       negat,       zero;
} parte_operativa;

/* MEMORIA mem */
typedef struct
{
long unsigned  ram[2345];
long unsigned  latch_end;
} memoria;

typedef struct
{
parte_controle       pcon;
interface            pint;
parte_operativa      po;
memoria              mem;
} circuito;

/* +-------------------------------------------------------------------+ */
/* ! defines utilizados na descricao do Risco                          ! */
/* +-------------------------------------------------------------------+ */
#define BIT31   0x80000000L   /*  -BITxx e' uma constante  */
#define BIT30   0x40000000L   /*  de 32 bits onde          */
#define BIT24   0x1000000L    /*  todos os bits sao zero,  */
#define BIT23   0x800000L     /*  exceto o bit xx          */
#define BIT22   0x400000L
#define BIT16   0x10000L
#define BIT11   0x800L
#define BIT10   0x400L
#define BIT4    0x10L
#define BIT3    0x8L
#define BIT2    0x4L
#define BIT1    0x2L
#define BIT0    0x1L
```

```
#define CAMPOCOD        0x3e000000L
#define CAMPODST        0x3e0000L
#define CAMPOFT1        0x1f000L
#define CAMPOFT2        0x7c0L
#define CAMPOKP         0x7ffL
#define CAMPOKG         0x1fffffL

#define F       ( 0x0L )
#define V       ( ~F   )
#define VERD    V

#define NUM_0   0x0
#define NUM_1   0x1
#define NUM_31  0x1f

#define END_R00 NUM_0
#define END_PSW NUM_1
#define END_PC  NUM_31

#define R00     breg[END_R00]
#define PSW     breg[END_PSW]
#define PC      breg[END_PC]

/*-------------------------------------------------------------------------*/
/* Inclusao do "globais.h" (obrigatorio) e de outras bibliotecas (opcional)*/
/*-------------------------------------------------------------------------*/
#include "e:globais.h"
#include "e:funcoes.bib"

/*-------------------------------------------------------------------------*/
/*    Declaracao dos nodos que serao alterados no decorrer da simulacao    */
/*                NODO(char *, <long, unsigned, int, char>, int);          */
/*             NODO("Idetificacao", variavel_a_ser_alterada, radical);     */
/*    radical: 16 (hexadecimal), 10 (decimal), 8 (octal), 2 (binaria)      */
/*-------------------------------------------------------------------------*/
ALTNODOS
{
/*
     NODO("rda_ba", sa->pint.rda_ba, 16);
*/
}

/*-------------------------------------------------------------------------*/
/*                     Descricao das fases do circuito                     */
/*-------------------------------------------------------------------------*/
DESCFASES
{
             nfases = 3;
/*
             Fase(1, atraso inicial, periodo total, periodo ativo);
             Fase(2, atraso inicial, periodo total, periodo ativo);
             Fase(3, atraso inicial, periodo total, periodo ativo);
*/
}

/*-------------------------------------------------------------------------*/
/*                         Parametros de Simulacao                         */
/*-------------------------------------------------------------------------*/
PARAMETROS
{
#define ZERO        0
#define ANALISADOR (-1)
#define ATRASO      1

/*
             Arquivo(0, 10000, 1);
        SIMULADOR = ZERO;
        SIMULADOR = ANALISADOR;
        SIMULADOR = ATRASO;
*/
        SIMULADOR = ATRASO;
}
```

```
/*-------------------------------------------------------------------------*/
/*                          Iniciacao das variaveis                        */
/*-------------------------------------------------------------------------*/
INICIALIZACAO
{
/*
     sn->pint.rda_ba = sa->pint.rda_ba;
*/
}

/*-------------------------------------------------------------------------*/
/*                      Descricao do Sistema Digital                       */
/*-------------------------------------------------------------------------*/
void funcao_parte_controle( void );
void funcao_interface( void );
void funcao_parte_operativa( void );
void funcao_memoria( void );

DESCRICAO
{
funcao_parte_controle();
funcao_interface();
funcao_parte_operativa();
funcao_memoria();
}
#include "R-PC.C"
#include "R-INT.C"
#include "R-PO.C"
#include "R-MEM.C"

/*-------------------------------------------------------------------------*/
/*                     Exibicao grafica das variaveis                      */
/*                          JANELA(long, char *);                          */
/*                        VALOR(long, char *, int);                        */
/*               JANELA(valor_a_ser_exibido, "Idetificacao");              */
/*          VALOR(valor_a_ser_exibido, "Idetificacao", radical);           */
/*               radical: 16 (hexadecimal), 10 (decimal)                   */
/*-------------------------------------------------------------------------*/
EXIBEJANELA
{
#define xJAN( a, b ) /* nada */
#define XJAN( a, b ) /* nada */
#define JAN( a, b ) JANELA( b, sa->pint.a )

 JANELA( "f1", f[1] );
 JANELA( "f2", f[2] );
 JANELA( "f3", f[3] );
 JANELA( "S0", sa->pcon.s0 );
 JANELA( "S1", sa->pcon.s1 );

 JAN( ale, "ale" );
 JAN( rd, "rd" );
 JAN( wr, "wr" );

xJAN( proxri_bsis, "proxri_bsis" );
xJAN( rmem_bsis, "rmem_bsis" );
xJAN( proxK_bsis, "proxK_bsis" );
xJAN( ri_proxri, "ri_proxri" );
xJAN( K_proxK, "K_proxK" );

 JAN( ba_dst, "ba_dst" );
 JAN( ba_ft1, "ba_ft1" );
 JAN( ba_r0, "ba_r0" );

 JAN( bb_ft2, "bb_ft2" );
 JAN( bb_K, "bb_K" );
 JAN( bb_dst, "bb_dst" );
 JAN( bb_r0, "bb_r0" );

xJAN( rua_ba, "rua_ba" );
xJAN( rub_bb, "rub_bb" );
xJAN( rula_codula, "rula_codula" );
```

```
xJAN( bb_1, "bb_1" );
xJAN( ba_ula, "ba_ula" );
xJAN( bb_ula, "bb_ula" );

 JAN( bb_rmem, "bb_rmem" );

 JAN( ft2_ba, "ft2_ba" );
 JAN( dst_bb, "dst_bb" );

 JAN( bout_bb, "bout_bb" );
 JAN( bout_inc, "bout_inc" );
xJAN( bout_pc, "bout_pc" );

xJAN( bsis_bout, "bsis_bout" );

xJAN( pcinc_bout, "pcinc_bout" );
xJAN( pc_bout, "pc_bout" );

xJAN( rda_ba, "rda_ba" );
xJAN( rdb_bb, "rdb_bb" );
xJAN( rud_codud, "rud_codud" );
xJAN( bb_ud, "bb_ud" );
}

/*-------------------------------------------------------------------------*/
EXIBEVALOR
{
#define xVAL( a, b, c ) /* nada */
#define XVAL( a, b, c ) /* nada */
#define xVALOR( a, b, c ) /* nada */
#define XVALOR( a, b, c ) /* nada */
#define VAL( a, b, c ) VALOR( b, sa->po.a, c )

 VALOR( "proxri", sa->pcon.proxri, 16 );
 VALOR( "ri", sa->pcon.ri, 16 );
xVALOR( "CONTA", sa->pcon.CONTA, 16 );

xVALOR( "dst", sa->pcon.enddst, 16 );
xVALOR( "ft2", sa->pcon.endft2, 16 );
xVALOR( "dst=pc", sa->pcon.dst_eh_pc, 16 );

 VAL( proxK, "proxK", 16 );
 VAL( K_ext, "K_ext", 16 );
 VAL( rmem, "rmem", 16 );

 VAL( breg[0], "R0", 16 );
 VAL( breg[1], "R1", 16 );
 VAL( breg[2], "R2", 16 );
 VAL( breg[3], "R3", 16 );
 VAL( breg[4], "R4", 16 );
xVAL( breg[5], "R5", 16 );
xVAL( breg[6], "R6", 16 );
xVAL( breg[7], "R7", 16 );
xVAL( breg[8], "R8", 16 );
xVAL( breg[9], "R9", 16 );
xVAL( breg[10], "R10", 16 );
xVAL( breg[29], "R29", 16 );
 VAL( breg[30], "SP", 16 );

 VAL( rula, "rula", 16 );
 VAL( rua, "rua", 16 );
 VAL( rub, "rub", 16 );

 VAL( ba, "ba", 16 );
 VAL( bb, "bb", 16 );

 VAL( ula, "ula", 16 );

 VAL( bout, "bout", 16 );
 VAL( bsis, "bsis", 16 );

 VAL( breg[31], "PC", 16 );
 VAL( inc, "inc", 16 );
}
```

```
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* |                                                                   | */
/* |                          +---------+                              | */
/* |                          |  Risco  |                              | */
/* |                          +---------+                              | */
/* |                                                                   | */
/* |              Microprocessador RISC CMOS 32 bits                   | */
/* |                                                                   | */
/* |              Autores: Andre', Dani, Giba, Gil,                    | */
/* |                       GME, Jorge, Junqueira, Luigi,               | */
/* |                       Marcon, Paulo, Soto, Suzim.                 | */
/* |                                                                   | */
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                     Risco  [  r-pc.c    ]                         | */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                     PARTE DE CONTROLE                             | */
/* +-------------------------------------------------------------------+ */

void funcao_parte_controle()
{
/* +------------------+ */
/* |      FASE: 1     | */
/* +------------------+ */

/* +------------------+ */
/* |  DECODIFICACAO !! | */
/* +------------------+ */

#define  INT_ISR   0xfe3c0c001
#define  INT_NOOP  0x000000001

if(sa->pint.proxri_bsis)      sn->pcon.proxri = sa->po.bsis;
if(sa->pint.proxri_int_isr)   sn->pcon.proxri = INT_ISR;
if(sa->pint.proxri_int_noop)  sn->pcon.proxri = INT_NOOP;
if(sa->pint.ri_proxri)        sn->pcon.ri = sa->pcon.proxri;


/* +-------------------------------------------------------------------+ */
/* | extracao dos campos da palavra de instrucao                       | */
/* +-------------------------------------------------------------------+ */

/* - t1 e t0 indicam o tipo de instrucao                                */
/* - t1 indica se ha' acesso a memoria (ime e isr)                      */
sn->pcon.t1       =  ( ( sa->pcon.ri      &  BIT31 ) ? V : F );

/* - t0 indica se cod e' codigo (ila e ime) ou teste (isa e isr)        */
sn->pcon.t0       =  ( ( sa->pcon.ri      &  BIT30 ) ? V : F );

sn->pcon.cod      =    ( sa->pcon.ri      &  CAMPOCOD )  >>  25 ;
sn->pcon.codb4    =  ( ( sa->pcon.cod     &  BIT4  ) ? V : F );
sn->pcon.codb3    =  ( ( sa->pcon.cod     &  BIT3  ) ? V : F );
sn->pcon.codb2    =  ( ( sa->pcon.cod     &  BIT2  ) ? V : F );
sn->pcon.codb1    =  ( ( sa->pcon.cod     &  BIT1  ) ? V : F );
sn->pcon.codb0    =  ( ( sa->pcon.cod     &  BIT0  ) ? V : F );

/* - aps indica se a palavra de status sera atualizada ao fim da inst.  */
sn->pcon.aps      =  ( ( sa->pcon.ri      &  BIT24 ) ? V : F );

/* - f1 e f0, junto com ss2, indicam o formato da instrucao             */
sn->pcon.f1       =  ( ( sa->pcon.ri      &  BIT23 ) ? V : F );
sn->pcon.f0       =  ( ( sa->pcon.ri      &  BIT22 ) ? V : F );
```

```
/* - endxxx e' o  endereco de dst (registrador destino),                */
/* ft1 (reg. fonte 1) e ft2 (reg. fonte 2).                             */
sn->pcon.enddst    =    ( sa->pcon.ri      &  CAMPODST )  >>  17 ;
sn->pcon.enddstb4  =  ( ( sa->pcon.enddst  &  BIT4  ) ? V : F );
sn->pcon.enddstb3  =  ( ( sa->pcon.enddst  &  BIT3  ) ? V : F );
sn->pcon.enddstb2  =  ( ( sa->pcon.enddst  &  BIT2  ) ? V : F );
sn->pcon.enddstb1  =  ( ( sa->pcon.enddst  &  BIT1  ) ? V : F );
sn->pcon.enddstb0  =  ( ( sa->pcon.enddst  &  BIT0  ) ? V : F );

sn->pcon.endft1    =    ( sa->pcon.ri      &  CAMPOFT1 )  >>  12 ;
sn->pcon.endft1b4  =  ( ( sa->pcon.endft1  &  BIT4  ) ? V : F );
sn->pcon.endft1b3  =  ( ( sa->pcon.endft1  &  BIT3  ) ? V : F );
sn->pcon.endft1b2  =  ( ( sa->pcon.endft1  &  BIT2  ) ? V : F );
sn->pcon.endft1b1  =  ( ( sa->pcon.endft1  &  BIT1  ) ? V : F );
sn->pcon.endft1b0  =  ( ( sa->pcon.endft1  &  BIT0  ) ? V : F );

sn->pcon.ss2       =  ( ( sa->pcon.ri      &  BIT11 ) ? V : F );

sn->pcon.endft2    =    ( sa->pcon.ri      &  CAMPOFT2 )  >>   6 ;
sn->pcon.endft2b4  =  ( ( sa->pcon.endft2  &  BIT4  ) ? V : F );
sn->pcon.endft2b3  =  ( ( sa->pcon.endft2  &  BIT3  ) ? V : F );
sn->pcon.endft2b2  =  ( ( sa->pcon.endft2  &  BIT2  ) ? V : F );
sn->pcon.endft2b1  =  ( ( sa->pcon.endft2  &  BIT1  ) ? V : F );
sn->pcon.endft2b0  =  ( ( sa->pcon.endft2  &  BIT0  ) ? V : F );


/* +-------------------------------------------------------------------+ */
/* | define os operandos e o destino destes                           | */
/* +-------------------------------------------------------------------+ */
/* exemplo de mnemonico: a_ft1 significa bus a recebera' ft1            */

sn->pcon.a_ft1  =  *sa->pcon.f1  &  *sa->pcon.f0;
sn->pcon.a_r0   =  *sa->pcon.f1  &   sa->pcon.f0;
sn->pcon.a_dst  =   sa->pcon.f1;

sn->pcon.b_ft2  =  *sa->pcon.f1  &  *sa->pcon.f0  &  *sa->pcon.ss2;
sn->pcon.b_k    =  *sa->pcon.b_ft2;
sn->pcon.b_kp   =  *sa->pcon.f1  &  *sa->pcon.f0  &   sa->pcon.ss2;
sn->pcon.b_kgl  =                    sa->pcon.f0;
sn->pcon.b_kgh  =   sa->pcon.f1  &  *sa->pcon.f0;

/*  e e' sempre dst_b (destino recebe bus B)          */


/* +-------------------------------------------------------------------+ */
/* | definicao do ciclo de maquina                                     | */
/* +-------------------------------------------------------------------+ */

/* MAQUINA DE ESTADOS */
if( f[2] )      sn->pcon.l_prox_est  = sa->pcon.s0;
if( f[3] )      sn->pcon.l_est_atual = sa->pcon.l_prox_est;
sn->pcon.s0 = *( sa->pcon.l_est_atual  | *sa->pcon.t1);
sn->pcon.s1 =    sa->pcon.l_est_atual;

/*
m0      - primeiro (prim.) e unico ciclo, normal, p/ ila e isa
        - f2, f3 e f1 do ciclo normal, nesta ordem

m1      - prim. ciclo de mem, p/ ime ou isr
        - f2, f3 do ciclo normal (prim.), e f1 do ciclo extra (segundo.)

m2      - seg. ciclo de mem, p/ ime ou isr
        - f2 e f3 do ciclo extra (seg.), e f1 do ciclo normal (prim. de novo)

m01 = m0 or m1
        - f2 e f3 do prim. ciclo ou do ciclo unico: independe da instrucao

m02 = m0 or m2
        - f1 do prim. ciclo ou do ciclo unico: independe da instrucao
*/
```

```
sn->pcon.m1   =  ~( ~sa->pcon.s0 |  sa->pcon.s1 );
sn->pcon.m0   =  ~(  sa->pcon.s0 |  sa->pcon.s1 );
sn->pcon.m2   =  ~(  sa->pcon.s0 | ~sa->pcon.s1 );
sn->pcon.m01  =                    ~sa->pcon.s1;
sn->pcon.m02  =     ~sa->pcon.s0;


/* +-------------------------------------------------------------------+ */
/* | definicao do tipo de instrucao                                    | */
/* +-------------------------------------------------------------------+ */

sn->pcon.ila = ~sa->pcon.t1 & ~sa->pcon.t0; /* instrucao logico-aritmetica */
sn->pcon.isa = ~sa->pcon.t1 &  sa->pcon.t0; /* ...de salto                 */
sn->pcon.ime =  sa->pcon.t1 & ~sa->pcon.t0; /* ...de acesso a mem.(ld/st)  */
sn->pcon.isr =  sa->pcon.t1 &  sa->pcon.t0; /* ...de sub-rotina (call)     */


/* +-------------------------------------------------------------------+ */
/* | definicao do ciclo de maquina para cada instrucao                 | */
/* +-------------------------------------------------------------------+ */

sn->pcon.ila0 = sa->pcon.ila & sa->pcon.m0;

sn->pcon.isa0 = sa->pcon.isa & sa->pcon.m0;

sn->pcon.ime1 = sa->pcon.ime & sa->pcon.m1;
sn->pcon.ime2 = sa->pcon.ime & sa->pcon.m2;

sn->pcon.isr1 = sa->pcon.isr & sa->pcon.m1;
sn->pcon.isr2 = sa->pcon.isr & sa->pcon.m2;


/* +-------------------------------------------------------------------+ */
/* | geracao de comandos                                               | */
/* | mnemonico: sx->pcon.pc_bout significa reg. pc recebe bus bout     | */
/* +-------------------------------------------------------------------+ */

sn->pcon.dst_eh_r0 = ~( sa->pcon.enddstb4  |  sa->pcon.enddstb3
                      | sa->pcon.enddstb2  |  sa->pcon.enddstb1
                      | sa->pcon.enddstb0 );

sn->pcon.dst_eh_psw = ~(  sa->pcon.enddstb4  |  sa->pcon.enddstb3
                        |  sa->pcon.enddstb2  |  sa->pcon.enddstb1
                        | ~sa->pcon.enddstb0 );

sn->pcon.dst_eh_pc =   sa->pcon.enddstb4  &  sa->pcon.enddstb3
                     & sa->pcon.enddstb2  &  sa->pcon.enddstb1
                     & sa->pcon.enddstb0;

sn->pcon.ft2_eh_pc =   sa->pcon.endft2b4  &  sa->pcon.endft2b3
                     & sa->pcon.endft2b2  &  sa->pcon.endft2b1
                     & sa->pcon.endft2b0;

sn->pcon.ft2_ig_dst =  ~(sa->pcon.endft2  ^  sa->pcon.enddst);

sn->pcon.usa_ud =     sa->pcon.ila &   sa->pcon.codb4;

sn->pcon.usa_ula =  ( sa->pcon.ila & (~sa->pcon.codb4) )
                  | ( sa->pcon.isa                     )
                  | ( sa->pcon.ime                     )
                  | ( sa->pcon.isr                     );

sn->pcon.load  =  sa->pcon.ime & (~sa->pcon.codb4);

sn->pcon.store =  sa->pcon.ime &   sa->pcon.codb4;

sn->pcon.altft2 = sa->pcon.ime & sa->pcon.codb2;

sn->pcon.preinc = sa->pcon.ime
                & sa->pcon.codb2 &  sa->pcon.codb1 &  sa->pcon.codb0;

sn->pcon.posinc = sa->pcon.ime
                & sa->pcon.codb2 & ~sa->pcon.codb1 &  sa->pcon.codb0;
```

```
sn->pcon.posdec = sa->pcon.ime
            & sa->pcon.codb2 & ~sa->pcon.codb1 & ~sa->pcon.codb0;

#define  ADD      0x00
#define  ADD1     0x01
#define  SUBNC    0x04
#define  SUBRNC   0x08

if(sa->pcon.ila0 & sa->pcon.usa_ud)      sn->pcon.codud = sa->pcon.cod;
if(sa->pcon.ila0 & sa->pcon.usa_ula)     sn->pcon.codula = sa->pcon.cod;
if(sa->pcon.isa0)                       sn->pcon.codula = ADD;
if(sa->pcon.ime1)
{
  if(sa->pcon.preinc)                   sn->pcon.codula = ADD1;
  else                                       sn->pcon.codula = ADD;
};
if(sa->pcon.isr1)          sn->pcon.codula = SUBNC;
if(sa->pcon.ime2)
{
  if(sa->pcon.preinc)                    sn->pcon.codula = ADD1;
  if(sa->pcon.posinc)                    sn->pcon.codula = ADD1;
  if(sa->pcon.posdec)                    sn->pcon.codula = SUBRNC;
};
if(sa->pcon.isr2)          sn->pcon.codula = ADD;

/*cond*/
sn->pcon.cond_v = V;

/*int*/
sn->pcon.int_ac = F;

/*
if (f[3] & sa->pcon.ila0 & ~sa->pcon.dst_eh_pc)
sn->pcon.int_ac = sa->po.int_ext;
*/


/* +-------------------+ */
/* |      FASE: 2      | */
/* +-------------------+ */

 sn->pcon.ba_ft1_f2 =
(sa->pcon.ila0 & ( sa->pcon.a_ft1              ) )|
(sa->pcon.isa0 & ( sa->pcon.a_ft1              ) )|
(sa->pcon.ime1 & ( sa->pcon.a_ft1              ) )|
(sa->pcon.isr2 & ( sa->pcon.a_ft1              ) );

 sn->pcon.ba_r0_f2 =
(sa->pcon.ila0 & ( sa->pcon.a_r0               ) )|
(sa->pcon.isa0 & ( sa->pcon.a_r0               ) )|
(sa->pcon.ime1 & ( sa->pcon.a_r0               ) )|
(sa->pcon.ime2 & ( sa->pcon.altft2             ) )|
(sa->pcon.isr2 & ( sa->pcon.a_r0               ) );

 sn->pcon.ba_dst_f2 =
(sa->pcon.ila0 & ( sa->pcon.a_dst              ) )|
(sa->pcon.isa0 & ( sa->pcon.a_dst              ) )|
(sa->pcon.ime1 & ( sa->pcon.a_dst              ) )|
(sa->pcon.isr1                                 )|
(sa->pcon.isr2 & ( sa->pcon.a_dst              ) );

 sn->pcon.bb_ft2_f2 =
(sa->pcon.ila0 & ( sa->pcon.b_ft2              ) )|
(sa->pcon.isa0 & ( sa->pcon.b_ft2              ) )|
(sa->pcon.ime1 & ( sa->pcon.b_ft2              ) )|
(sa->pcon.isr2 & ( sa->pcon.b_ft2              ) );

 sn->pcon.bb_K_f2 =
(sa->pcon.ila0 & ( sa->pcon.b_k                ) )|
(sa->pcon.isa0 & ( sa->pcon.b_k                ) )|
(sa->pcon.ime1 & ( sa->pcon.b_k                ) )|
(sa->pcon.isr2 & ( sa->pcon.b_k                ) );
```

```
 sn->pcon.bb_r0_f2 =
(sa->pcon.isr1                                  );

 sn->pcon.rua_ba_f2  =
(sa->pcon.m01  & ( sa->pcon.usa_ula             ) )|
(sa->pcon.ime2 & ( sa->pcon.altft2              ) )|
(sa->pcon.isr2                                  );

 sn->pcon.rula_codula_f2 = sa->pcon.rua_ba_f2;

 sn->pcon.rub_bb_f2  =
(sa->pcon.m01  & ( sa->pcon.usa_ula             ) )|
(sa->pcon.isr2                                  );

 sn->pcon.rda_ba_f2  =
(sa->pcon.m01  & ( sa->pcon.usa_ud              ) );

 sn->pcon.rdb_bb_f2    = sa->pcon.rda_ba_f2;
 sn->pcon.rud_codud_f2 = sa->pcon.rda_ba_f2;

/*int*/
 sn->pcon.rd_f2 =
(sa->pcon.m0                                      )|
(sa->pcon.m1   & ( *sa->pcon.int_ac             ) )|
(sa->pcon.ime2 & ( sa->pcon.load                ) );

 sn->pcon.wr_f2 =
(sa->pcon.ime2 & ( sa->pcon.store               ) )|
(sa->pcon.isr2                                    );

/*int*/
 sn->pcon.bout_pc_f2 =
(sa->pcon.m01  & ( *sa->pcon.int_ac               ) )|
(sa->pcon.ime2 & (  sa->pcon.store & sa->pcon.dst_eh_pc) )|
(sa->pcon.isr2 & (  sa->pcon.int_ac               ) );

/*int*/
 sn->pcon.pcinc_bout_f2 =
(sa->pcon.m01  & ( *sa->pcon.int_ac               ) );

/*int*/
 sn->pcon.bout_inc_f2 =
(sa->pcon.isr2 & ( *sa->pcon.int_ac ) );

 sn->pcon.bout_bb_f2 =
(sa->pcon.ime2 & ( sa->pcon.store & (*sa->pcon.dst_eh_pc) ) );

 sn->pcon.bb_dst_f2 =
(sa->pcon.ime2 & ( sa->pcon.store & (*sa->pcon.dst_eh_pc) ) );

 sn->pcon.bsis_bout_f2 =
(sa->pcon.ime2 & ( sa->pcon.store               ) )|
(sa->pcon.isr2                                    );

/* +-------------------------------------+ */
/* | LATCH DOS COMANDOS DA FASE 2        | */
/* +-------------------------------------+ */
/* +--  RD e WR valem nas fases 2 e 3  ---+ */
/* +-------------------------------------+ */

if      ( f[2] )
{       sn->pcon.rd_f2_1  =  sa->pcon.rd_f2;
        sn->pcon.wr_f2_1  =  sa->pcon.wr_f2;
}

/* +------------------+ */
/* |      FASE: 3     | */
/* +------------------+ */

/*int*/
 sn->pcon.proxri_bsis_f3 =
(sa->pcon.m01  & ( *sa->pcon.int_ac ) );
```

```
/*int*/
 sn->pcon.proxri_int_isr_f3 =
(sa->pcon.m0  & ( sa->pcon.int_ac ) );

/*int*/
 sn->pcon.proxri_int_noop_f3 =
(sa->pcon.m1  & ( sa->pcon.int_ac ) );

 sn->pcon.ri_proxri_f3 =
(sa->pcon.m02                          )|
(sa->pcon.isr1 & ( *sa->pcon.cond_v ) );

 sn->pcon.proxK_bsis_f3 =
(sa->pcon.m01             );

 sn->pcon.K_proxK_f3 =
(sa->pcon.m02                         )|
(sa->pcon.isr1 & ( *sa->pcon.cond_v ) );

 sn->pcon.bb_1_f3 =
(sa->pcon.m01                                    )|
(sa->pcon.ime2 & ( sa->pcon.load                 ) )|
(sa->pcon.isr2                                   );

 sn->pcon.rmem_bsis_f3 =
(sa->pcon.ime2 & ( sa->pcon.load                 ) );

 sn->pcon.bb_dst_f3 =
(sa->pcon.ime2 & ( sa->pcon.store & (*sa->pcon.dst_eh_pc) ) );

 sn->pcon.bout_bb_f3 =
(sa->pcon.ime2 & ( sa->pcon.store & (*sa->pcon.dst_eh_pc) ) );

 sn->pcon.bsis_bout_f3 =
(sa->pcon.ime2 & ( sa->pcon.store                ) )|
(sa->pcon.isr2                                   );

/*int*/
 sn->pcon.bout_pc_f3 =
(sa->pcon.ime2 & ( sa->pcon.store & sa->pcon.dst_eh_pc) )|
(sa->pcon.isr2 & (  sa->pcon.int_ac              ) );

/*int*/
 sn->pcon.bout_inc_f3 =
(sa->pcon.isr2 & ( *sa->pcon.int_ac  ) );

/*int*/
 sn->pcon.rd_f3 =
(sa->pcon.m0                                     )|
(sa->pcon.m1   & ( *sa->pcon.int_ac              ) )|
(sa->pcon.ime2 & ( sa->pcon.load                 ) );

 sn->pcon.wr_f3 =
(sa->pcon.ime2 & ( sa->pcon.store                ) )|
(sa->pcon.isr2                                   );
```

```
/* +-------------------------------------+ */
/* | LATCH DOS COMANDOS DA FASE 3        | */
/* +-------------------------------------+ */

if( f[2] )
{
            sn->pcon.proxri_bsis_f3_1  =  sa->pcon.proxri_bsis_f3;
            sn->pcon.proxri_int_isr_f3_1 = sa->pcon.proxri_int_isr_f3;
            sn->pcon.proxri_int_noop_f3_1 = sa->pcon.proxri_int_noop_f3;
       sn->pcon.ri_proxri_f3_1    =  sa->pcon.ri_proxri_f3;
       sn->pcon.proxK_bsis_f3_1   =  sa->pcon.proxK_bsis_f3;
       sn->pcon.K_proxK_f3_1      =  sa->pcon.K_proxK_f3;
       sn->pcon.bb_1_f3_1         =  sa->pcon.bb_1_f3;
       sn->pcon.rmem_bsis_f3_1    =  sa->pcon.rmem_bsis_f3;
       sn->pcon.bb_dst_f3_1       =  sa->pcon.bb_dst_f3;
       sn->pcon.bout_bb_f3_1      =  sa->pcon.bout_bb_f3;
       sn->pcon.bsis_bout_f3_1    =  sa->pcon.bsis_bout_f3;
       sn->pcon.bout_pc_f3_1      =  sa->pcon.bout_pc_f3;
            sn->pcon.bout_inc_f3_1     =  sa->pcon.bout_inc_f3;
            sn->pcon.rd_f3_1           =  sa->pcon.rd_f3;
       sn->pcon.wr_f3_1           =  sa->pcon.wr_f3;
}


/* +------------------+ */
/* |      FASE: 1     | */
/* +------------------+ */

 sn->pcon.bout_bb_f1 =
(sa->pcon.ila0 & ( sa->pcon.dst_eh_pc                                          ))|
(sa->pcon.isa0 & ( sa->pcon.dst_eh_pc & sa->pcon.cond_v                        ))|
(sa->pcon.ime1                                                                  )|
(sa->pcon.ime2 & ( sa->pcon.dst_eh_pc & sa->pcon.load &   sa->pcon.altft2  ))|
(sa->pcon.ime2 & ( sa->pcon.dst_eh_pc & sa->pcon.load & (*sa->pcon.altft2) ))|
(sa->pcon.isr1 & ( sa->pcon.cond_v                                             ))|
(sa->pcon.isr2                                                                  );

 sn->pcon.bout_inc_f1=
(sa->pcon.ila0 & ( *sa->pcon.dst_eh_pc                                         ))|
(sa->pcon.isa0 & ( *(sa->pcon.dst_eh_pc & sa->pcon.cond_v)                     ))|
(sa->pcon.ime2 & (sa->pcon.store
     &   sa->pcon.altft2   & (*sa->pcon.ft2_eh_pc)                             ))|
(sa->pcon.ime2 & (sa->pcon.store
     & (*sa->pcon.altft2 )                                                     ))|
(sa->pcon.ime2 & (sa->pcon.load
     &   sa->pcon.altft2   & (*sa->pcon.dst_eh_pc) & (*sa->pcon.ft2_eh_pc) ))|
(sa->pcon.ime2 & (sa->pcon.load
     & (*sa->pcon.altft2 ) & (*sa->pcon.dst_eh_pc)                             ))|
(sa->pcon.isr1 & ( *sa->pcon.cond_v                                            ));

 sn->pcon.pc_bout_f1 =
(sa->pcon.ila0                                                                  )|
(sa->pcon.isa0                                                                  )|
(sa->pcon.isr1 & (   sa->pcon.cond_v & sa->pcon.dst_eh_pc                  ) )|
(sa->pcon.isr1 & (  *sa->pcon.cond_v                                       ) )|
(sa->pcon.ime2 & ( sa->pcon.store
     &   sa->pcon.altft2                          & (*sa->pcon.ft2_eh_pc) ))|
(sa->pcon.ime2 & ( sa->pcon.store
     & (*sa->pcon.altft2 )                                                 ) )|
(sa->pcon.ime2 & ( sa->pcon.load
     &   sa->pcon.altft2  &   sa->pcon.dst_eh_pc                           ) )|
(sa->pcon.ime2 & ( sa->pcon.load
     &   sa->pcon.altft2  & (*sa->pcon.dst_eh_pc) & (*sa->pcon.ft2_eh_pc) ) )|
(sa->pcon.ime2 & ( sa->pcon.load
     & (*sa->pcon.altft2 )                                                 ) )|
(sa->pcon.isr2                                                                  );

/*int*/
 sn->pcon.bsis_bout_f1 =
(sa->pcon.m0   & ( *sa->pcon.int_ac                    ) )|
(sa->pcon.m1                                              )|
(sa->pcon.m2                                              );
```

```
 sn->pcon.bout_pc_f1 =
(sa->pcon.ime2 & ( sa->pcon.store
    & sa->pcon.altft2  & sa->pcon.ft2_eh_pc                          ) )|
(sa->pcon.ime2 & ( sa->pcon.load
    & sa->pcon.altft2  & sa->pcon.ft2_eh_pc & (*sa->pcon.dst_eh_pc)  ) );

/*int*/
 sn->pcon.ale_f1 =
(sa->pcon.m0  & ( *sa->pcon.int_ac                    ) )|
(sa->pcon.m1                                            )|
(sa->pcon.m2                                            );

 sn->pcon.bb_rmem_f1 =
(sa->pcon.ime2 & ( sa->pcon.load                      ) );

 sn->pcon.bb_ud_f1 =
(sa->pcon.ila0 & ( sa->pcon.usa_ud                    ) );

 sn->pcon.bb_ula_f1 =
(sa->pcon.ila0 & ( sa->pcon.usa_ula                                ) )|
(sa->pcon.isa0 & ( sa->pcon.cond_v                                 ) )|
(sa->pcon.ime1                                                       )|
(sa->pcon.isr1 & ( sa->pcon.cond_v                                 ) )|
(sa->pcon.isr2                                                       );

 sn->pcon.dst_bb_f1 =
(sa->pcon.ila0 & (
    *sa->pcon.dst_eh_pc  &  *sa->pcon.dst_eh_r0                    ) )|
(sa->pcon.isa0 & (
    *sa->pcon.dst_eh_pc  &  *sa->pcon.dst_eh_r0  &  sa->pcon.cond_v ) )|
(sa->pcon.ime2 & (
    *sa->pcon.dst_eh_pc  &  *sa->pcon.dst_eh_r0  &  sa->pcon.load   ) )|
(sa->pcon.isr1 & (
    *sa->pcon.dst_eh_pc  &  *sa->pcon.dst_eh_r0  &  sa->pcon.cond_v ) );

 sn->pcon.ft2_ba_f1 =
(sa->pcon.ime2 & ( sa->pcon.store & sa->pcon.altft2                 ) )|
(sa->pcon.ime2 & ( sa->pcon.load  & sa->pcon.altft2
 &   sa->pcon.dst_eh_pc  & (*sa->pcon.ft2_eh_pc)                   ) )|
(sa->pcon.ime2 & ( sa->pcon.load  & sa->pcon.altft2
 & (*sa->pcon.dst_eh_pc) &   sa->pcon.ft2_eh_pc                    ) )|
(sa->pcon.ime2 & ( sa->pcon.load  & sa->pcon.altft2
 & (*sa->pcon.dst_eh_pc) & (*sa->pcon.ft2_eh_pc) & (*sa->pcon.ft2_ig_dst) ) );

 sn->pcon.ba_ula_f1 =
(sa->pcon.ime2 & ( sa->pcon.store & sa->pcon.altft2                 ) )|
(sa->pcon.ime2 & ( sa->pcon.load  & sa->pcon.altft2
 &   sa->pcon.dst_eh_pc  & (*sa->pcon.ft2_eh_pc)                   ) )|
(sa->pcon.ime2 & ( sa->pcon.load  & sa->pcon.altft2
 & (*sa->pcon.dst_eh_pc) &   sa->pcon.ft2_eh_pc                    ) )|
(sa->pcon.ime2 & ( sa->pcon.load  & sa->pcon.altft2
 & (*sa->pcon.dst_eh_pc) & (*sa->pcon.ft2_eh_pc) & (*sa->pcon.ft2_ig_dst) ) );

 sn->pcon.wr_f1 =
(sa->pcon.ime1 & ( sa->pcon.store                                  ) )|
(sa->pcon.isr1 & ( sa->pcon.cond_v                                 ) );

 sn->pcon.rd_f1 =
(sa->pcon.ime1 & ( sa->pcon.load                                   ) );
```

```
/* +--------------------------------------+ */
/* | LATCH DOS COMANDOS DA FASE 1         | */
/* +--------------------------------------+ */

if      ( f[2] )
{       sn->pcon.bout_bb_f1_l   = sa->pcon.bout_bb_f1;
        sn->pcon.bout_inc_f1_l  = sa->pcon.bout_inc_f1;
        sn->pcon.pc_bout_f1_l   = sa->pcon.pc_bout_f1;
        sn->pcon.bsis_bout_f1_l = sa->pcon.bsis_bout_f1;
        sn->pcon.bout_pc_f1_l   = sa->pcon.bout_pc_f1;
        sn->pcon.ale_f1_l       = sa->pcon.ale_f1;
        sn->pcon.bb_rmem_f1_l   = sa->pcon.bb_rmem_f1;
        sn->pcon.bb_ud_f1_l     = sa->pcon.bb_ud_f1;
        sn->pcon.bb_ula_f1_l    = sa->pcon.bb_ula_f1;
        sn->pcon.dst_bb_f1_l    = sa->pcon.dst_bb_f1;
        sn->pcon.ft2_ba_f1_l    = sa->pcon.ft2_ba_f1;
        sn->pcon.ba_ula_f1_l    = sa->pcon.ba_ula_f1;
        sn->pcon.wr_f1_l        = sa->pcon.wr_f1;
              sn->pcon.rd_f1_l        = sa->pcon.rd_f1;
              sn->pcon.enddst_l       = sa->pcon.enddst;
              sn->pcon.endft1_l       = sa->pcon.endft1;
              sn->pcon.endft2_l       = sa->pcon.endft2;
}

} /* fim de funcao_parte_controle */

/* +-------------------------------------------------------------------+ */
```

```
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* |                                                                   | */
/* |                          +---------+                              | */
/* |                          |  Risco  |                              | */
/* |                          +---------+                              | */
/* |                                                                   | */
/* |                Microprocessador RISC CMOS 32 bits                 | */
/* |                                                                   | */
/* |                Autores: Andre', Dani, Giba, Gil,                  | */
/* |                         GME, Jorge, Junqueira, Luigi,             | */
/* |                         Marcon, Paulo, Soto, Suzim.               | */
/* |                                                                   | */
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                     Risco  [  r-int.c  ]                          | */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                           INTERFACE                               | */
/* +-------------------------------------------------------------------+ */

void funcao_interface()
{
sn->pint.ale =          sa->pcon.ale_f1_1           & f[1];
sn->pint.ba_dst =       sa->pcon.ba_dst_f2          & f[2];
sn->pint.ba_ft1 =       sa->pcon.ba_ft1_f2          & f[2];
sn->pint.ba_r0 =        sa->pcon.ba_r0_f2           & f[2];
sn->pint.ba_ula =       sa->pcon.ba_ula_f1_1        & f[1];
sn->pint.bb_1 =         sa->pcon.bb_1_f3_1          & f[3];
sn->pint.bb_K =         sa->pcon.bb_K_f2            & f[2];

sn->pint.bb_dst =      (sa->pcon.bb_dst_f2          & f[2])|
                       (sa->pcon.bb_dst_f3_1        & f[3]);

sn->pint.bb_ft2 =       sa->pcon.bb_ft2_f2          & f[2];
sn->pint.bb_r0 =        sa->pcon.bb_r0_f2           & f[2];
sn->pint.bb_rmem =      sa->pcon.bb_rmem_f1_1       & f[1];
sn->pint.bb_ud =        sa->pcon.bb_ud_f1_1         & f[1];
sn->pint.bb_ula =       sa->pcon.bb_ula_f1_1        & f[1];

sn->pint.bout_bb =     (sa->pcon.bout_bb_f1_1       & f[1])|
                       (sa->pcon.bout_bb_f2         & f[2])|
                       (sa->pcon.bout_bb_f3_1       & f[3]);

sn->pint.bout_pc =     (sa->pcon.bout_pc_f1_1       & f[1])|
                       (sa->pcon.bout_pc_f2         & f[2])|
                       (sa->pcon.bout_pc_f3_1       & f[3]);

sn->pint.bout_inc =    (sa->pcon.bout_inc_f1_1      & f[1])|
                                (sa->pcon.bout_inc_f2       & f[2])|
                                (sa->pcon.bout_inc_f3_1     & f[3]);

sn->pint.bsis_bout =   (sa->pcon.bsis_bout_f1_1     & f[1])|
                       (sa->pcon.bsis_bout_f2       & f[2])|
                       (sa->pcon.bsis_bout_f3_1     & f[3]);

sn->pint.proxK_bsis =   sa->pcon.proxK_bsis_f3_1    & f[3];
sn->pint.K_proxK =      sa->pcon.K_proxK_f3_1       & f[3];
sn->pint.dst_bb =       sa->pcon.dst_bb_f1_1        & f[1];
sn->pint.ft2_ba =       sa->pcon.ft2_ba_f1_1        & f[1];
sn->pint.pcinc_bout =   sa->pcon.pcinc_bout_f2      & f[2];
sn->pint.pc_bout =      sa->pcon.pc_bout_f1_1       & f[1];

sn->pint.proxri_bsis     = sa->pcon.proxri_bsis_f3_1      & f[3];
sn->pint.proxri_int_isr  = sa->pcon.proxri_int_isr_f3_1   & f[3];
sn->pint.proxri_int_noop = sa->pcon.proxri_int_noop_f3_1  & f[3];

sn->pint.rda_ba =       sa->pcon.rda_ba_f2          & f[2];
sn->pint.rdb_bb =       sa->pcon.rdb_bb_f2          & f[2];
```

```
sn->pint.rd =          (sa->pcon.rd_f1_1          & f[1])|
                       (sa->pcon.rd_f2_1          & f[2])|
                       (sa->pcon.rd_f3_1          & f[3]);

sn->pint.ri_proxri =    sa->pcon.ri_proxri_f3_1   & f[3];
sn->pint.rmem_bsis =    sa->pcon.rmem_bsis_f3_1   & f[3];
sn->pint.rua_ba    =    sa->pcon.rua_ba_f2        & f[2];
sn->pint.rub_bb    =    sa->pcon.rub_bb_f2        & f[2];
sn->pint.rud_codud   = sa->pcon.rud_codud_f2      & f[2];
sn->pint.rula_codula = sa->pcon.rula_codula_f2    & f[2];

sn->pint.wr =          (sa->pcon.wr_f1_1          & f[1])|
                       (sa->pcon.wr_f2_1          & f[2])|
                       (sa->pcon.wr_f3_1          & f[3]);
}

/* +-------------------------------------------------------------------+ */
```

```
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* |                                                                   | */
/* |                          +---------+                              | */
/* |                          |  Risco  |                              | */
/* |                          +---------+                              | */
/* |                                                                   | */
/* |              Microprocessador RISC CMOS 32 bits                   | */
/* |                                                                   | */
/* |              Autores: Andre', Dani, Giba, Gil,                    | */
/* |                       GME, Jorge, Junqueira, Luigi,               | */
/* |                       Marcon, Paulo, Soto, Suzim.                 | */
/* |                                                                   | */
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                     Risco [   r-po.c   ]                          | */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                          PARTE OPERATIVA                          | */
/* +-------------------------------------------------------------------+ */

void funcao_po(void);
void funcao_constante(void);
void funcao_banco_registradores(void);
void funcao_ula2(void);
void funcao_ula1(void);
void funcao_pla_ula(void);
void funcao_ula0(void);

void funcao_parte_operativa()
{
        funcao_po();
        funcao_constante();
        funcao_banco_registradores();
        funcao_ula2();
}

void funcao_po()
{
        if      (sa->pint.bout_pc)
                sn->po.bout   =  sa->po.PC;
        if      (sa->pint.pcinc_bout)
                sn->po.pcinc  =  sa->po.bout;
        sn->po.inc  =  sa->po.pcinc  +  1;

        if      (sa->pint.bb_1)
                sn->po.bb  =  V; /* pre-carga adiantada */

        if      (sa->pint.bout_bb)
                sn->po.bout  =  sa->po.bb;
        if      (sa->pint.bout_inc)
                sn->po.bout  =  sa->po.inc;

        if      (sa->pint.pc_bout)
                sn->po.PC  =  sa->po.bout;

        if      (sa->pint.bsis_bout)
                sn->po.bsis  =  sa->po.bout;

        if      (sa->pint.rmem_bsis)
                sn->po.rmem  =  sa->po.bsis;

        if      (sa->pint.bb_rmem)
                sn->po.bb  =  sa->po.rmem;
}
```

```
void funcao_constante()
{
        if      (sa->pint.proxK_bsis)
                        sn->po.proxK = sa->po.bsis;

        if      (sa->pint.K_proxK)
                        sn->po.K = sa->po.proxK;


#define    CAMPOKP          0x7ffL
#define    CAMPOKG         0x1ffffL

#define    EXT_1_KP       0xfffffc00L
#define    EXT_1_KGLO     0xffff0000L
#define    EXT_1_KGHI     0x0000ffffL

              sn->po.sinal_kp = ( (sa->po.K & BIT10) ? V : F );
              sn->po.sinal_kg = ( (sa->po.K & BIT16) ? V : F );

        if (sa->pcon.b_kp)
             {
             if (sa->po.sinal_kp)
                  sn->po.K_ext = (sa->po.K & CAMPOKP) | EXT_1_KP;
             else sn->po.K_ext =  sa->po.K & CAMPOKP;
        };

        if (sa->pcon.b_kgl)
             {
             if (sa->po.sinal_kg)
                  sn->po.K_ext = (sa->po.K & CAMPOKG) | EXT_1_KGLO;
             else sn->po.K_ext =  sa->po.K & CAMPOKG;
             };

        if (sa->pcon.b_kgh)
             {
             if (sa->po.sinal_kg)
                  sn->po.K_ext = ((sa->po.K & CAMPOKG)<<16) | EXT_1_KGHI;
             else sn->po.K_ext = ((sa->po.K & CAMPOKG)<<16);
        };

        if      (sa->pint.bb_K)
                sn->po.bb  =  sa->po.K_ext;

}


void funcao_banco_registradores()
{
             sn->po.R00  =  0L;

        /* leitura - ciclo normal */
        if      (sa->pint.ba_ft1)
                sn->po.ba  =  sa->po.breg[ sa->pcon.endft1_1 ];
        if      (sa->pint.ba_r0) /* tambem para 2do ciclo de mem com altft2 */
                sn->po.ba  =  sa->po.R00;
        if      (sa->pint.ba_dst)  /* tambem para 1ro ciclo de sub-rot */
                sn->po.ba  =  sa->po.breg[ sa->pcon.enddst_1 ];

        /* leitura - ciclo normal */
        if      (sa->pint.bb_ft2)
                sn->po.bb  =  sa->po.breg[ sa->pcon.endft2_1 ];
              /* bb_K esta' na funcao_constante */

        if      (sa->pint.bb_r0) /* leitura - para 1ro ciclo de sub-rot */
                sn->po.bb  =  sa->po.R00;

        if      (sa->pint.bb_dst) /* no store */
                sn->po.bb  =  sa->po.breg[ sa->pcon.enddst_1 ];

        if      (sa->pint.ft2_ba) /* se altera ft2 */
                sn->po.breg[ sa->pcon.endft2_1 ]  =  sa->po.ba;
```

```
        /* escrita - ciclo normal */
        if      (sa->pint.dst_bb) /* escrita do resultado */
                sn->po.breg[ sa->pcon.enddst_l ]  =  sa->po.bb;


}

void funcao_pla_ula()
{
#define   C4   ( ( sa->po.rula  & BIT4 )  ?  V  :  F  )
#define   C3   ( ( sa->po.rula  & BIT3 )  ?  V  :  F  )
#define   C2   ( ( sa->po.rula  & BIT2 )  ?  V  :  F  )
#define   C1   ( ( sa->po.rula  & BIT1 )  ?  V  :  F  )
#define   C0   ( ( sa->po.rula  & BIT0 )  ?  V  :  F  )

/*
        Instrucao       Codigo (.rula)  Operacao
        ---------       ------          --------
        and             01111           ula  =   a  &  b
        or              01110           ula  =   a  |  b
        xor             01101           ula  =   a  ^  b
        .               01100           ula  =   ?

        subrc           01011           ula  =  ~a  +  b  + ~carry
        subrcnot        01010           ula  =  ~a  +  b  +  carry
        subr            01001           ula  =  ~a  +  b  +  1
        subrnc          01000           ula  =  ~a  +  b  +  0

        subc            00111           ula  =   a  + ~b  + ~carry
        subcnot         00110           ula  =   a  + ~b  +  carry
        sub             00101           ula  =   a  + ~b  +  1
        subnc           00100           ula  =   a  + ~b  +  0

        addcnot         00011           ula  =   a  +  b  + ~carry
        addc            00010           ula  =   a  +  b  +  carry
        add1            00001           ula  =   a  +  b  +  1
        add             00000           ula  =   a  +  b  +  0
*/

        sn->po.log   =  ~C4  &   ( C3  &  C2 );
        sn->po.arit  =  ~C4  &  ~( C3  &  C2 );

        sn->po.and  =  sa->po.log  &   C1  &   C0;
        sn->po.or   =  sa->po.log  &   C1  &  ~C0;
        sn->po.xor  =  sa->po.log  &  ~C1  &   C0;

        sn->po.add    =  sa->po.arit;
        sn->po.inv_a  =  sa->po.arit  &  C3;
        sn->po.inv_b  =  sa->po.arit  &  C2;
        sn->po.com_c  =  sa->po.arit  &  C1;
        sn->po.inv_c  =  sa->po.arit  &  C0;
}

void funcao_ula0()
{
        if      (sa->po.sel_and)
                {
                        sn->po.ula  =  sa->po.a  &  sa->po.b;
                        /* sn->po.cout  nao altera */
                        /* sn->po.overf nao altera */
                };

        if      (sa->po.sel_or)
                {
                        sn->po.ula  =  sa->po.a  |  sa->po.b;
                        /* sn->po.cout  nao altera */
                        /* sn->po.overf nao altera */
                };
```

```
        if      (sa->po.sel_xor)
                {
                        sn->po.ula  =  sa->po.a  ^  sa->po.b;
                        /* sn->po.cout  nao altera */
                        /* sn->po.overf nao altera */
                };

        if      (sa->po.sel_add)
                {
                        unsigned char vcout,i,va,vb,vcin;
                        unsigned long bit_i;

                        sn->po.ula =  sa->po.a  +
                                      sa->po.b  +
                                      ( sa->po.cin ? 1 : 0 );
                        vcout = sa->po.cin;
                        bit_i = 1L;
                        for     (  i=0;  i<=31;  i=i+1  )
                                {
                                        va = ( (sa->po.a & bit_i) ? V : F );
                                        vb = ( (sa->po.b & bit_i) ? V : F );
                                        vcin = ( (vcout) ? V : F );
                                        vcout = ( va  &  vb   ) |
                                                ( va  &  vcin ) |
                                                ( vb  &  vcin );
                                        bit_i = bit_i << 1;
                                };
                        /*      vcout = cout do bit 31,                         */
                        /*      vcin  = cout do bit 30 = cin do bit 31          */
                        sn->po.cout  = ( vcout ? V : F );
                        sn->po.overf = ( vcout ? V : F ) ^ ( vcin ? V : F );

                };

        sn->po.negat  =  ( ( sa->po.ula < 0 ) ? V : F );
        sn->po.zero   =  (   sa->po.ula       ? F : V  );
}

void funcao_ula1()
{
        funcao_pla_ula();
        sn->po.a  =  ( sa->po.inv_a  ?  *sa->po.rua  :  sa->po.rua );
        sn->po.b  =  ( sa->po.inv_b  ?  *sa->po.rub  :  sa->po.rub );
        sn->po.cin  =  ( sa->po.inv_c  ?
                                ( sa->po.com_c  ?  *sa->po.c  :  1 )  :
                                ( sa->po.com_c  ?   sa->po.c  :  0 )
         );
        sn->po.sel_and  =  sa->po.and;
        sn->po.sel_or   =  sa->po.or;
        sn->po.sel_xor  =  sa->po.xor;
        sn->po.sel_add  =  sa->po.add;
        funcao_ula0();
}

void funcao_ula2()
{
        if      (sa->pint.rula_codula)
                sn->po.rula  =  sa->pcon.codula;
        if      (sa->pint.rua_ba)
                sn->po.rua   =  sa->po.ba;
        if      (sa->pint.rub_bb)
                sn->po.rub   =  sa->po.bb;
        funcao_ula1();
        if      (sa->pint.bb_ula)
                sn->po.bb    =  sa->po.ula;
        if      (sa->pint.ba_ula)
                sn->po.ba    =  sa->po.ula;

}

/* +-----------------------------------------------------------------+ */
```

```
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* |                                                                   | */
/* |                            +---------+                            | */
/* |                            |  Risco  |                            | */
/* |                            +---------+                            | */
/* |                                                                   | */
/* |                Microprocessador RISC CMOS 32 bits                 | */
/* |                                                                   | */
/* |                Autores: Andre', Dani, Giba, Gil,                  | */
/* |                         GME, Jorge, Junqueira, Luigi,             | */
/* |                         Marcon, Paulo, Soto, Suzim.               | */
/* |                                                                   | */
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                      Risco  [  r-mem.c  ]                         | */
/* +-------------------------------------------------------------------+ */

/* +-------------------------------------------------------------------+ */
/* |                             MEMORIA                               | */
/* +-------------------------------------------------------------------+ */

/* funcao que converte campos decimais da instr em um hexa de 32 bits */

long unsigned instr(char *,  char, char *,  char *,   char, char, char, char);
long unsigned instr(tipo,    cod,  apsw,    form,     dst,  ft1,  ft2,  resto)
              char tipo[4], cod,  apsw[2], form[12], dst,  ft1,  ft2,  resto;
{
unsigned char       t10,       aps,    f10,ss2;
long unsigned valor;

t10=100;
if (!strcmp(tipo,"ila")) t10=0;
if (!strcmp(tipo,"isa")) t10=1;
if (!strcmp(tipo,"ime")) t10=2;
if (!strcmp(tipo,"isr")) t10=3;

aps=100;
if (!strcmp(apsw,"n")) aps=0;
if (!strcmp(apsw,"a")) aps=1;

f10=100;
ss2=100;
if ( !strcmp(form,"dst-ft1-ft2") ) { f10=0; ss2=0; };
if ( !strcmp(form,"dst-ft1-kp")  ) { f10=0; ss2=1; };
if ( !strcmp(form,"dst-r0-kgl")  ) { f10=1; ss2=0; };
if ( !strcmp(form,"dst-dst-kgh") ) { f10=2; ss2=0; };
if ( !strcmp(form,"dst-dst-kgl") ) { f10=3; ss2=0; };

if (t10>3)    return(0);
if (cod>31)   return(0);
if (aps>1)    return(0);
if (f10>3 )   return(0);
if (dst>31)   return(0);
if (ft1>31)   return(0);
if (ss2>1)    return(0);
if (ft2>31)   return(0);
if (resto>63) return(0);

valor=0L;
```

```
/*       contagem dos parenteses...       */
/*       4321    11   1  2      3      4  */
valor =  ((((long)(t10)  )  &3L ) << 30)
              |((((long)(cod)   )  &31L) << 25)
              |((((long)(aps)   )  &1L ) << 24)
              |((((long)(ft0)   )  &3L ) << 22)
              |((((long)(dst)   )  &31L) << 17)
              |((((long)(ft1)   )  &31L) << 12)
              |((((long)(ss2)   )  &1L ) << 11)
              |((((long)(ft2)   )  &31L) << 6 )
              |((((long)(resto))  &63L)      );

return(valor);
}

void funcao_memoria()
{
if      (sa->pint.ale)
             sn->mem.latch_end  =  sa->po.bsis;
if      (sa->pint.rd & ~sa->pint.ale)
             sn->po.bsis = sa->mem.ram[(unsigned int)sa->mem.latch_end];
if      (sa->pint.wr & ~sa->pint.ale)
             sn->mem.ram[(unsigned int)sa->mem.latch_end] = sa->po.bsis;

/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* +--  inicio do programa  ------------------------------------------+ */

/*
   Este programa apenas executa 3 classes de instrucao: log-arit.,
sub-rotina e memoria.
   Em 2 inicializa o SP (com 20), em 4 vai para a sub-rotina (em 50),
e em 52 retorna da sub-rot para o end do SP (20). Todos os numeros em decimal.
*/

/*                    tipo, cod,apsw,  form,       dst,ft1,ft2,resto */
sn->mem.ram[ 0]=01;
sn->mem.ram[ 1]=11;
sn->mem.ram[ 2]=instr( "ila",  0, "n", "dst-ft1-kp",  30, 0, 0, 20);
sn->mem.ram[ 3]=31;
sn->mem.ram[ 4]=instr( "isr",  0, "n", "dst-ft1-kp",  30, 0, 0, 50);
sn->mem.ram[ 5]=51;
sn->mem.ram[ 6]=61;

sn->mem.ram[18]=181;
sn->mem.ram[19]=191;
sn->mem.ram[20]=201;
sn->mem.ram[21]=211;
sn->mem.ram[21]=221;
sn->mem.ram[23]=231;
sn->mem.ram[23]=241;

sn->mem.ram[49]=491;
sn->mem.ram[50]=501;
sn->mem.ram[51]=511;
sn->mem.ram[52]=instr( "ime",  5, "n", "dst-ft1-ft2", 31, 0, 30, 0);
sn->mem.ram[53]=531;
sn->mem.ram[54]=541;
sn->mem.ram[55]=551;

/* +--  fim do programa  ------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */
/* +-------------------------------------------------------------------+ */

}

/* +-------------------------------------------------------------------+ */
```

ANEXO A-2

Descrição algorítmica

```
#define verd  1
#define falso 0

/* ************************************ */
/* FASE 1 - E                           */

se la
entao ( se dst=pc
        entao bout := bb
        senao bout := inc;
        pc := bout
      );

se sl
entao ( se ( (dst=pc) e cond)
        entao bout := bb
        senao bout := inc;
        pc := bout
      );

se me
entao ( se ( (dst=pc)  e load)
        entao ( bout := bb;
                bout := inc
              )
        senao ( se ( altera_ft2 e (ft2=pc) )
                entao bout := pc
                senao ( bout := inc;
                        pc := bout
                      )
              )
      );

se sr
entao ( se cond
        entao bout := bb
        senao bout  := inc;
        pc := bout
      );

bsis := bout;


/* ************************************ */
/* FASE 1 - D  (decodificacao)          */

decodifica();


/* ************************************ */
/* FASE 1 - Em (escrita end. dado mem.)  */

se me
entao ( bb := sai_ula;
        bout := bb;
        bsis := bout;
        se altera_ft2
        entao ( ba := r0;
                rua := ba;
                ula := op_ula
              )
      );

se sr
entao ( calcula_op_ula;
        bb := sai_ula;
        bout := bb;
        bsis := bout;
        dst := bb
      );
```

```
/* ************************************ */
/* FASE 1 - W (escrita do resultado)    */

se la
entao ( se usa_ula
        entao bb := sai_ula
        senao bb := sai_ud;
        se (dst~=pc)
        entao dst := bb
      );

se sl
entao ( se cond
        entao ( bb := sai_ula;
                se (dst~=pc)
                entao dst := bb
              )
      );

se me
entao ( se load
        entao ( bb := rmem;
                se (dst~=pc)
                entao dst := bb
              )
      );

se (sr e cond)
entao bb := sai_ula;


/* ************************************ */
/* FASE 2 - Ia (leitura da instrucao)   */

bout := pc;
pcinc := bout;


/* ************************************ */
/* FASE 2 - R (leitura dos operandos)   */

ba := ft1;
bb := fb;
se usa_ula
entao ( rua := ba;
        rub := bb;
        ula := op_ula
      )
senao ( rda := ba;
        rdb := bb;
        ud := op_ud
      );


/* ************************************ */
/* FASE 2 - Ma (acesso dado memoria)    */

se me
entao ( se store
        entao ( bb := dst;
                bout := bb;
                bsis := bout
              );
        se altera_ft2
        entao ba := verd
      );
```

```
se sr
entao ( bout := pc;
        bsis := bout;
        ba := ft1;
        bb := fb;
        rua := ba;
        rub := bb;
        ula := op_ula
      );

/* ************************************ */
/* FASE 3 - Ib (leitura da instrucao)   */

in := bsis;
imed := bsis;

/* ************************************ */
/* FASE 3 - O (operacao)                */

bb := verd;

/* ************************************ */
/* FASE 3 - Mb (acesso dado memoria)    */

se me
entao ( se store
        entao ( bb := dst;
                bout := bb;
                bsis := bout
              )
        senao ( rmem := bsis; /* load */
                bb := verd
              );
        se altera_ft2
        entao ( ba := sai2_ula;
                se ( ft2=pc )
                entao pc := ba
                senao ft2 := ba
              );
        dec_breg := end_dst
      );

se sr
entao ( bout := pc;
        bsis := bout;
        bb := verd;
        dec_breg := end_dst;
      );
```

ANEXO A-3

Descrição Spice dos blocos

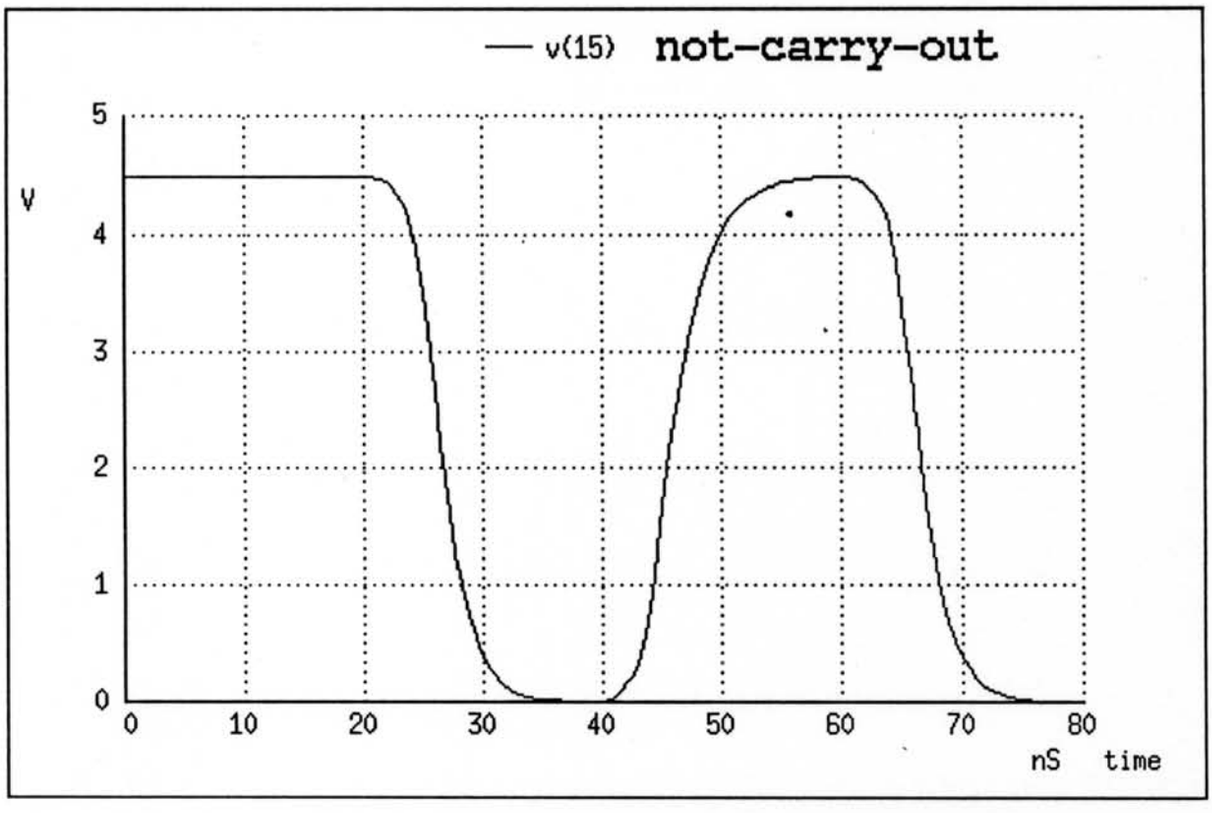
```
* CIRCUITO: ula2.txt
* TECNOLOGIA: cmos12.tec  TIPO CMOS
*
* Modelos dos transistores worst
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P
*
.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P

* *
* SUBCIRCUITO CORRESPONDENTE A CELULA  ula2.txt
*
* Transistores tipo NMOS: 54
* Transistores tipo PMOS: 54
* No    0 = gnd
* No    1 = vdd2
* No    6 = vdd1
* No    9 = a0
* No   10 = b0
* No   11 = b1
* No   12 = a1
* No   13 = bi0
* No   14 = axorb1
* No   15 = coutneg1
* No   16 = aorb0neg
* No   17 = aandb1neg
* No   18 = a+b+c0
* No   20 = laritneg
* No   21 = lxorneg1
* No   22 = s0neg
* No   23 = zout
* No   24 = bi1
* No   25 = aandb0
* No   26 = aorb0
* No   27 = cin0
* No   28 = lorneg1
* No   29 = aorb1neg
* No   30 = landneg1
* No   31 = aandb0neg
* No   32 = cinneg0
* No   34 = laritneg1
* No   35 = axorb0neg
* No   36 = s1
* No   37 = ai0neg
* No   38 = ai1neg
* No   39 = bi0neg
* No   40 = bi1neg
* No   41 = aandb1
* No   42 = aorb1
* No   43 = cin1
* No   44 = lor
* No   45 = lorneg
* No   46 = land
* No   47 = landneg
* No   48 = axorb0
* No   49 = cinneg1
* No   50 = axorb1neg
* No   51 = a+b+c1
* No   52 = larit
* No   53 = lxor
```

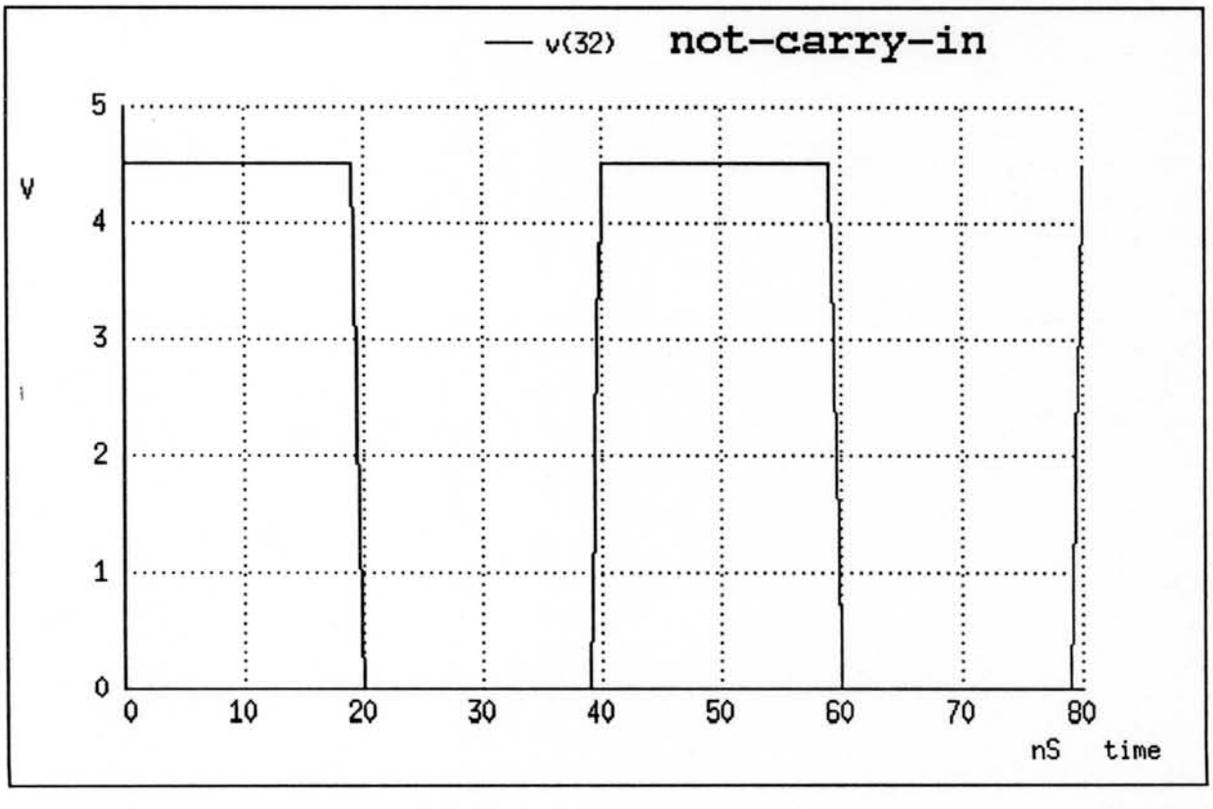
```
* No   54 = lxorneg
* No   55 = s0
* No   56 = zinnands0
* No   57 = zin
* No   58 = slneg

VB 39 0 dc 0
VA 37 0 dc 4.5
VC 32 0 PULSE(4.5 0 19n 1n 1n 19n 40n)

VDD 6 0 DC 4.5
VDD2 1 0 DC 4.5
VZ 57 0 dc 0
va1 38 0 dc 0
vb1 40 0 dc 4.5
VT 52 0 DC 4.5
VX 53 0 DC 0
VO 44 0 DC 0
VD 46 0 DC 0

cs0 55 0 200f
cs1 36 0 200f
cc 15 0 300f
cz 23 0 200f

MN1      5   58   23    0 NMOS L=1.2U W=3.0U AD=33P AS=10P PD=28U PS=13U
MP2      4   57    6    1 PMOS L=1.2U W=7.5U AD=83P AS=44P PD=37U PS=47U
MN3     56   57    0    0 NMOS L=1.2U W=3.0U AD=19P AS=21P PD=19U PS=20U
MP4     23   58    1    1 PMOS L=1.2U W=7.5U AD=42P AS=53P PD=27U PS=29U
MN5      5   56    0    0 NMOS L=1.2U W=3.0U AD=33P AS=70P PD=28U PS=66U
MP6     56   55    4    1 PMOS L=1.2U W=7.5U AD=60P AS=83P PD=31U PS=37U
MP7     23   56    1    1 PMOS L=1.2U W=7.5U AD=42P AS=53P PD=27U PS=29U
MN8     56   55    0    0 NMOS L=1.2U W=3.0U AD=19P AS=70P PD=19U PS=66U
MP9     55   22    6    1 PMOS L=1.2U W=37.5U AD=179P AS=715P PD=84U PS=113U
MN10    55   22    0    0 NMOS L=1.2U W=15.0U AD=89P AS=341P PD=43U PS=75U
MN11    36   58    0    0 NMOS L=1.2U W=15.0U AD=71P AS=341P PD=39U PS=75U
MP12    36   58    1    1 PMOS L=1.2U W=37.5U AD=231P AS=269P PD=87U PS=89U
MN13    50   53   58    0 NMOS L=1.2U W=6.0U AD=27P AS=29P PD=21U PS=24U
MP14    58   21   50    1 PMOS L=1.2U W=15.0U AD=67P AS=89P PD=38U PS=43U
MN15    22   53   35    0 NMOS L=1.2U W=6.0U AD=21P AS=52P PD=19U PS=32U
MP16    35   54   22    1 PMOS L=1.2U W=15.0U AD=71P AS=89P PD=39U PS=43U
MP17    21   53    1    1 PMOS L=1.2U W=15.0U AD=85P AS=107P PD=42U PS=44U
MN18    54   53    0    0 NMOS L=1.2U W=6.0U AD=30P AS=136P PD=25U PS=57U
MP19    54   53    6    1 PMOS L=1.2U W=15.0U AD=71P AS=278P PD=39U PS=68U
MN20    21   53    0    0 NMOS L=1.2U W=6.0U AD=28P AS=136P PD=21U PS=57U
MP21    33   20   22    1 PMOS L=1.2U W=15.0U AD=71P AS=89P PD=39U PS=43U
MN22    22   52   33    0 NMOS L=1.2U W=6.0U AD=21P AS=52P PD=19U PS=32U
MP23    58   34   19    1 PMOS L=1.2U W=15.0U AD=67P AS=89P PD=38U PS=43U
MN24    19   52   58    0 NMOS L=1.2U W=6.0U AD=27P AS=29P PD=21U PS=24U
MP25    34   52    1    1 PMOS L=1.2U W=15.0U AD=85P AS=107P PD=42U PS=44U
MN26    34   52    0    0 NMOS L=1.2U W=6.0U AD=28P AS=136P PD=21U PS=57U
MP27    20   52    6    1 PMOS L=1.2U W=15.0U AD=71P AS=278P PD=39U PS=68U
MN28    20   52    0    0 NMOS L=1.2U W=6.0U AD=30P AS=136P PD=25U PS=57U
MN29    33   18    0    0 NMOS L=1.2U W=6.0U AD=30P AS=136P PD=25U PS=57U
MP30    33   18    6    1 PMOS L=1.2U W=15.0U AD=71P AS=278P PD=39U PS=68U
MN31    19   51    0    0 NMOS L=1.2U W=6.0U AD=28P AS=136P PD=21U PS=57U
MP32    19   51    1    1 PMOS L=1.2U W=15.0U AD=85P AS=107P PD=42U PS=44U
MN33    18   48   32    0 NMOS L=1.2U W=9.0U AD=32P AS=82P PD=25U PS=38U
MP34    18   35   32    1 PMOS L=1.2U W=22.5U AD=107P AS=143P PD=54U PS=58U
MN35    49   14   51    0 NMOS L=1.2U W=9.0U AD=41P AS=48P PD=27U PS=30U
MP36    49   50   51    1 PMOS L=1.2U W=22.5U AD=107P AS=143P PD=54U PS=58U
MN37    18   32   48    0 NMOS L=1.2U W=9.0U AD=32P AS=82P PD=25U PS=38U
MP38    18   32   35    1 PMOS L=1.2U W=22.5U AD=107P AS=143P PD=54U PS=58U
MN39    14   49   51    0 NMOS L=1.2U W=9.0U AD=41P AS=48P PD=27U PS=30U
MP40    50   49   51    1 PMOS L=1.2U W=22.5U AD=107P AS=143P PD=54U PS=58U
MP41    50   14    1    1 PMOS L=1.2U W=30.0U AD=179P AS=215P PD=72U PS=74U
```

```
MN42    50   14    0    0 NMOS L=1.2U W=12.0U AD=57P AS=273P PD=33U PS=69U
MP43    35   48    6    1 PMOS L=1.2U W=30.0U AD=143P AS=566P PD=69U PS=98U
MN44    35   48    0    0 NMOS L=1.2U W=12.0U AD=70P AS=273P PD=37U PS=69U
MN45    17   46   58    0 NMOS L=1.2U W=6.0U AD=27P AS=30P PD=21U PS=25U
MP46    58   30   17    1 PMOS L=1.2U W=15.0U AD=67P AS=89P PD=38U PS=43U
MN47    22   46   31    0 NMOS L=1.2U W=6.0U AD=28P AS=23P PD=21U PS=22U
MP48    31   47   22    1 PMOS L=1.2U W=15.0U AD=71P AS=89P PD=39U PS=43U
MP49    30   46    1    1 PMOS L=1.2U W=15.0U AD=85P AS=107P PD=42U PS=44U
MP50    47   46    6    1 PMOS L=1.2U W=15.0U AD=71P AS=278P PD=39U PS=68U
MN51    47   46    0    0 NMOS L=1.2U W=6.0U AD=30P AS=136P PD=25U PS=57U
MN52    30   46    0    0 NMOS L=1.2U W=6.0U AD=28P AS=136P PD=21U PS=57U
MP53    58   28   29    1 PMOS L=1.2U W=15.0U AD=67P AS=89P PD=38U PS=43U
MN54    29   44   58    0 NMOS L=1.2U W=6.0U AD=27P AS=30P PD=21U PS=25U
MN55    22   44   16    0 NMOS L=1.2U W=6.0U AD=28P AS=23P PD=21U PS=22U
MP56    16   45   22    1 PMOS L=1.2U W=15.0U AD=71P AS=89P PD=39U PS=43U
MP57    28   44    1    1 PMOS L=1.2U W=15.0U AD=85P AS=107P PD=42U PS=44U
MN58    28   44    0    0 NMOS L=1.2U W=6.0U AD=28P AS=136P PD=21U PS=57U
MN59    45   44    0    0 NMOS L=1.2U W=6.0U AD=30P AS=136P PD=25U PS=57U
MP60    45   44    6    1 PMOS L=1.2U W=15.0U AD=71P AS=278P PD=39U PS=68U
MN61    49   32   31    0 NMOS L=1.2U W=4.5U AD=21P AS=19P PD=18U PS=19U
MP62    16   32   49    1 PMOS L=1.2U W=11.3U AD=53P AS=67P PD=32U PS=35U
MN63    17   49   15    0 NMOS L=1.2U W=4.5U AD=20P AS=25P PD=18U PS=22U
MP64    15   49   29    1 PMOS L=1.2U W=11.3U AD=53P AS=67P PD=32U PS=35U
MP65    31   27   49    1 PMOS L=1.2U W=11.3U AD=53P AS=67P PD=32U PS=35U
MN66    49   27   16    0 NMOS L=1.2U W=4.5U AD=21P AS=19P PD=18U PS=19U
MN67    29   43   15    0 NMOS L=1.2U W=4.5U AD=20P AS=25P PD=18U PS=22U
MP68    15   43   17    1 PMOS L=1.2U W=11.3U AD=50P AS=67P PD=31U PS=35U
MP69    27   32    6    1 PMOS L=1.2U W=30.0U AD=143P AS=566P PD=69U PS=98U
MN70    27   32    0    0 NMOS L=1.2U W=12.0U AD=70P AS=273P PD=37U PS=69U
MN71    43   49    0    0 NMOS L=1.2U W=12.0U AD=57P AS=273P PD=33U PS=69U
MP72    43   49    1    1 PMOS L=1.2U W=30.0U AD=179P AS=215P PD=72U PS=74U
MP73    16   26    6    1 PMOS L=1.2U W=22.5U AD=107P AS=427P PD=54U PS=83U
MN74    16   26    0    0 NMOS L=1.2U W=9.0U AD=50P AS=205P PD=31U PS=63U
MP75    29   42    1    1 PMOS L=1.2U W=22.5U AD=137P AS=161P PD=57U PS=59U
MN76    29   42    0    0 NMOS L=1.2U W=9.0U AD=43P AS=205P PD=27U PS=63U
MP77    31   25    6    1 PMOS L=1.2U W=22.5U AD=107P AS=427P PD=54U PS=83U
MN78    31   25    0    0 NMOS L=1.2U W=9.0U AD=50P AS=205P PD=31U PS=63U
MN79    17   41    0    0 NMOS L=1.2U W=9.0U AD=43P AS=205P PD=27U PS=63U
MP80    17   41    1    1 PMOS L=1.2U W=22.5U AD=137P AS=161P PD=57U PS=59U
MP81    26   39    6    1 PMOS L=1.2U W=15.0U AD=71P AS=278P PD=39U PS=68U
MP82    42   40    1    1 PMOS L=1.2U W=15.0U AD=85P AS=107P PD=42U PS=44U
MN83     3   40   42    0 NMOS L=1.2U W=6.0U AD=122P AS=44P PD=52U PS=29U
MN84     2   39   26    0 NMOS L=1.2U W=6.0U AD=122P AS=44P PD=52U PS=29U
MP85    26   37    6    1 PMOS L=1.2U W=15.0U AD=71P AS=278P PD=39U PS=68U
MP86    42   38    1    1 PMOS L=1.2U W=15.0U AD=85P AS=107P PD=42U PS=44U
MN87     3   38    0    0 NMOS L=1.2U W=6.0U AD=122P AS=105P PD=52U PS=81U
MN88     2   37    0    0 NMOS L=1.2U W=6.0U AD=122P AS=105P PD=52U PS=81U
MP89     8   40   41    1 PMOS L=1.2U W=15.0U AD=275P AS=125P PD=67U PS=47U
MP90     7   39   25    1 PMOS L=1.2U W=15.0U AD=278P AS=125P PD=67U PS=47U
MN91    41   40    0    0 NMOS L=1.2U W=6.0U AD=28P AS=136P PD=21U PS=57U
MN92    25   39    0    0 NMOS L=1.2U W=6.0U AD=30P AS=136P PD=25U PS=57U
MP93     8   38    1    1 PMOS L=1.2U W=18.0U AD=275P AS=272P PD=67U PS=75U
MP94     6   37    7    1 PMOS L=1.2U W=15.0U AD=247P AS=278P PD=74U PS=67U
MN95    25   37    0    0 NMOS L=1.2U W=6.0U AD=30P AS=136P PD=25U PS=57U
MN96    41   38    0    0 NMOS L=1.2U W=6.0U AD=28P AS=136P PD=21U PS=57U
MP97    48   39   37    1 PMOS L=1.2U W=22.5U AD=107P AS=143P PD=54U PS=58U
MN98    37   13   48    0 NMOS L=1.2U W=9.0U AD=32P AS=82P PD=25U PS=38U
MN99    14   24   38    0 NMOS L=1.2U W=9.0U AD=41P AS=48P PD=27U PS=30U
MP100   38   40   14    1 PMOS L=1.2U W=22.5U AD=107P AS=143P PD=54U PS=58U
MP101   48   37   39    1 PMOS L=1.2U W=22.5U AD=107P AS=143P PD=54U PS=58U
MN102   13   37   48    0 NMOS L=1.2U W=9.0U AD=32P AS=82P PD=25U PS=38U
MN103   14   38   24    0 NMOS L=1.2U W=9.0U AD=41P AS=48P PD=27U PS=30U
MP104   40   38   14    1 PMOS L=1.2U W=22.5U AD=107P AS=143P PD=54U PS=58U
MP105   24   40    1    1 PMOS L=1.2U W=22.5U AD=137P AS=161P PD=57U PS=59U
MN106   24   40    0    0 NMOS L=1.2U W=9.0U AD=43P AS=205P PD=27U PS=63U
MP107   13   39    6    1 PMOS L=1.2U W=22.5U AD=107P AS=427P PD=54U PS=83U
```

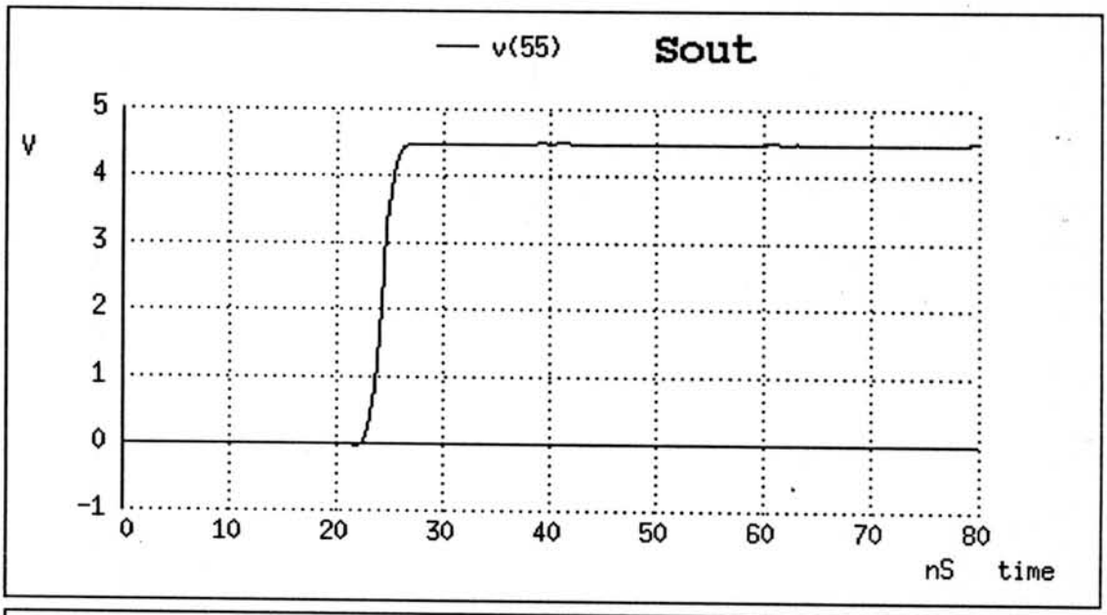
```
MN108   13   39    0    0 NMOS L=1.2U W=9.0U AD=50P AS=205P PD=31U PS=63U

.tran 1n 80n
.opt temp=125
.END
```

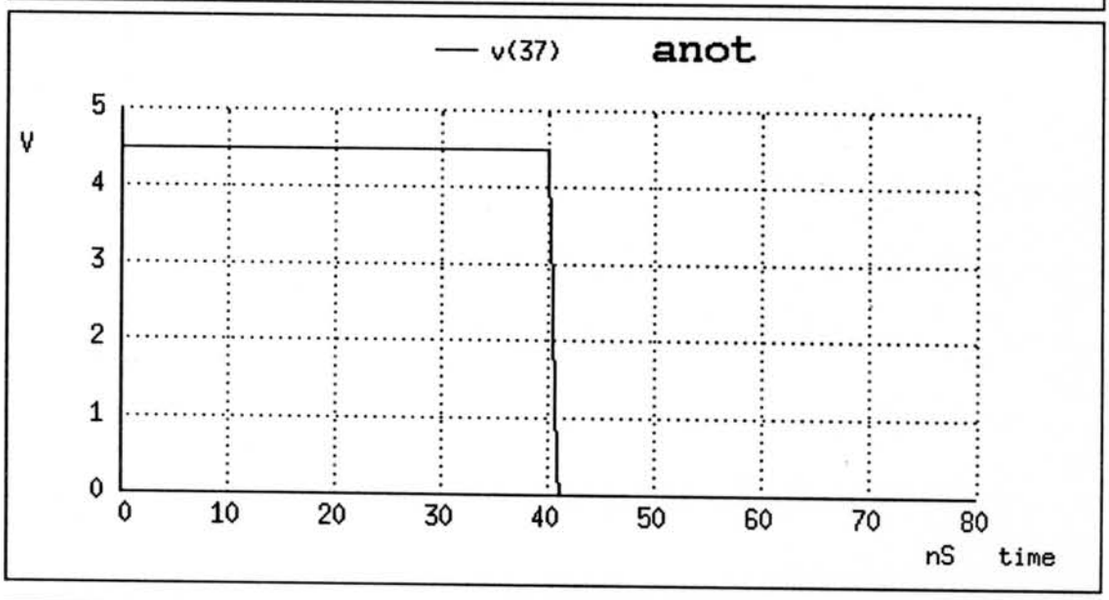
CARRY

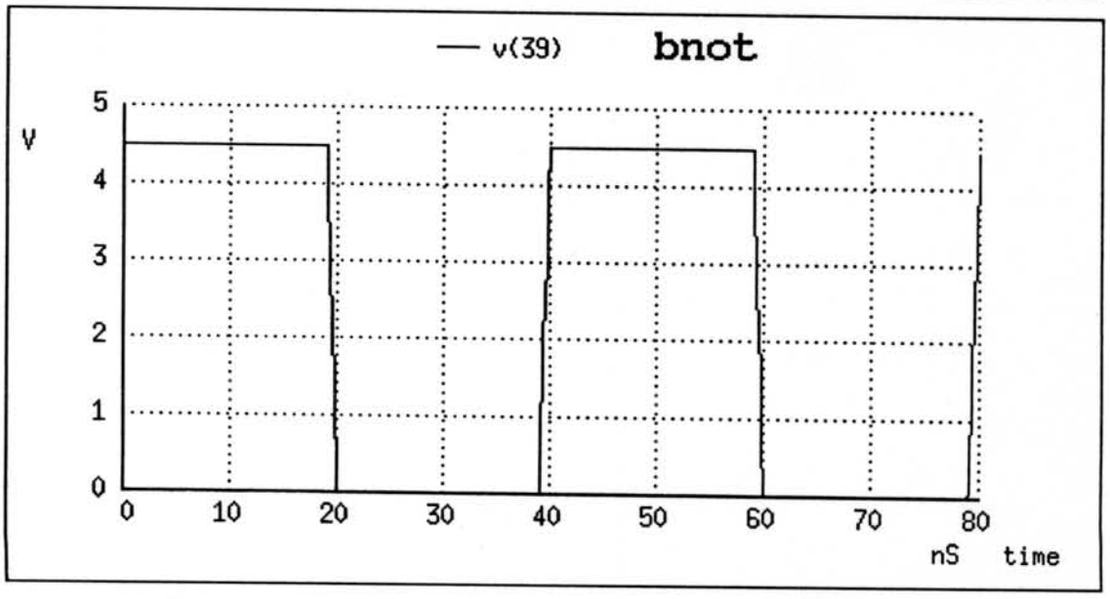

v(15) not-carry-out
V
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
nS time


v(32) not-carry-in
V
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
nS time

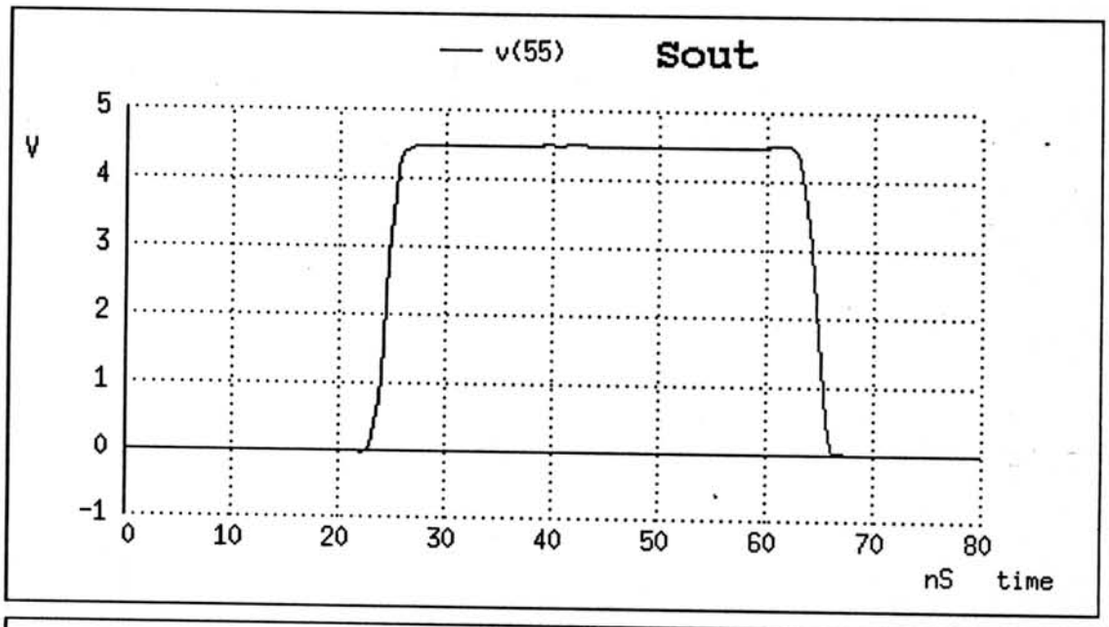
OR

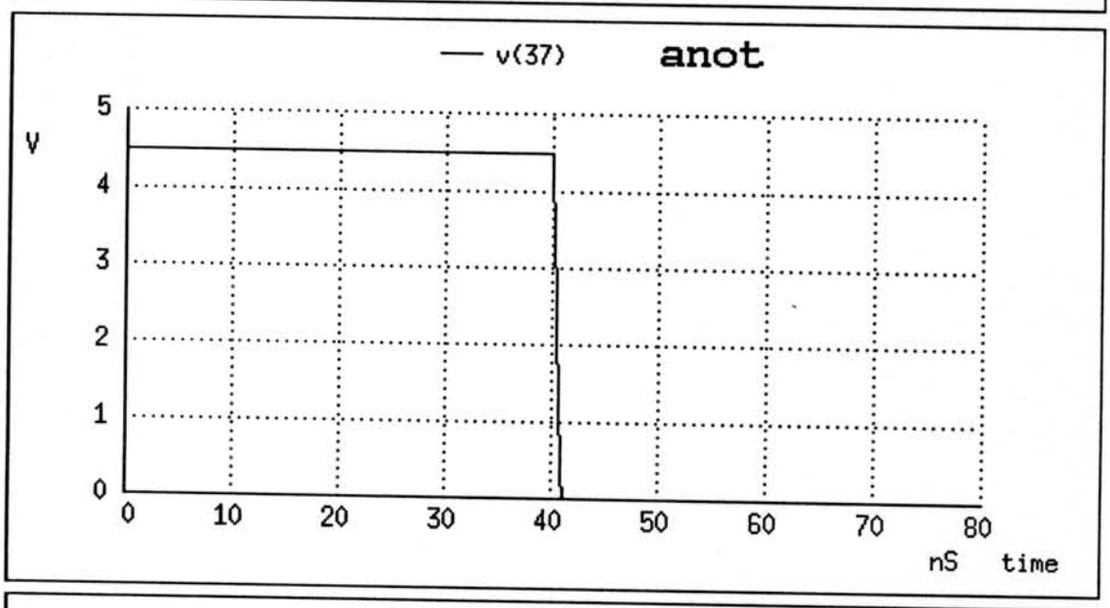

— v(55)
Sout
V
5
4
3
2
1
0
-1
0
10
20
30
40
50
60
70
80
nS
time

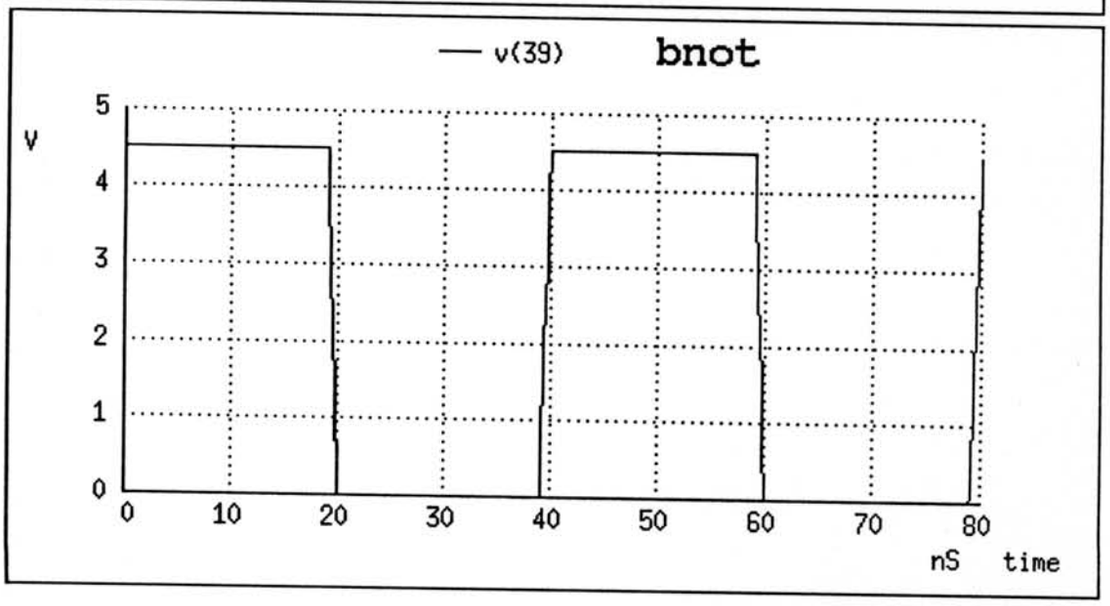

— v(37)
anot
V
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
nS
time


— v(39)
bnot
V
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
nS
time

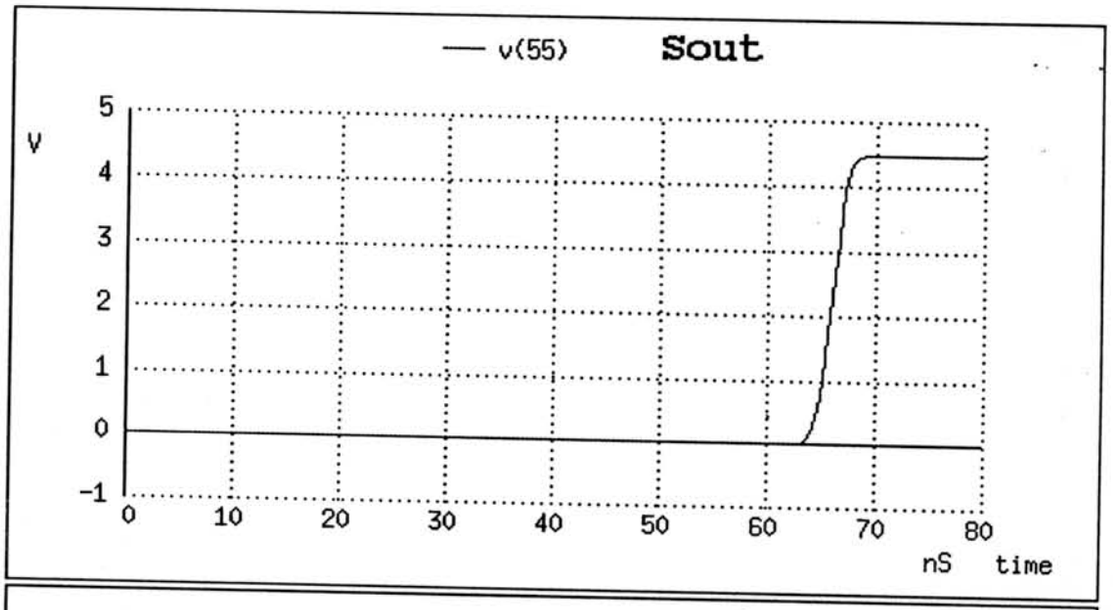
XOR

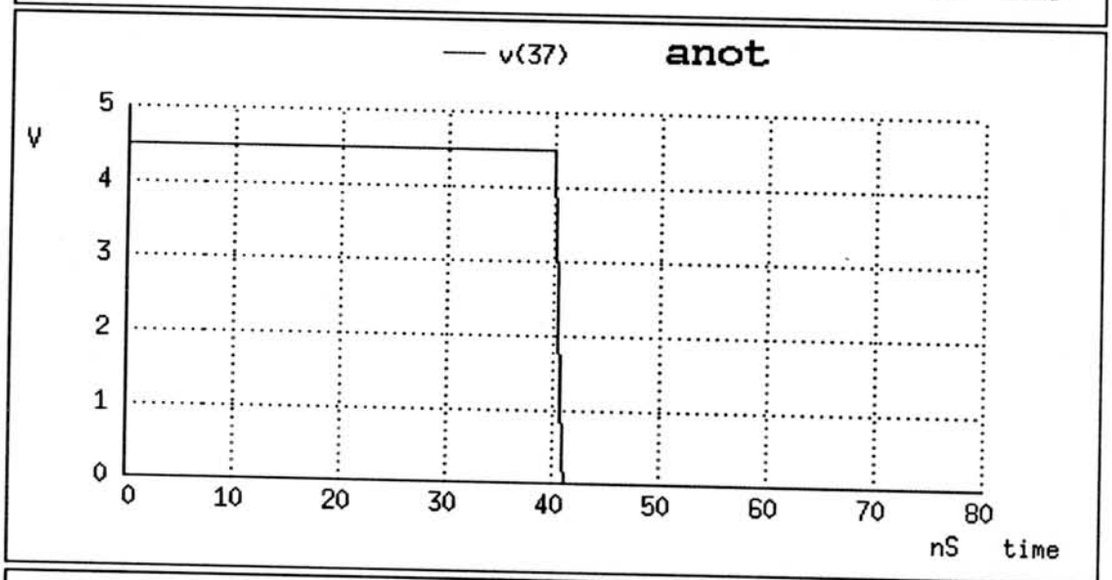

— v(55)
Sout
V
5
4
3
2
1
0
-1
0
10
20
30
40
50
60
70
80
nS
time

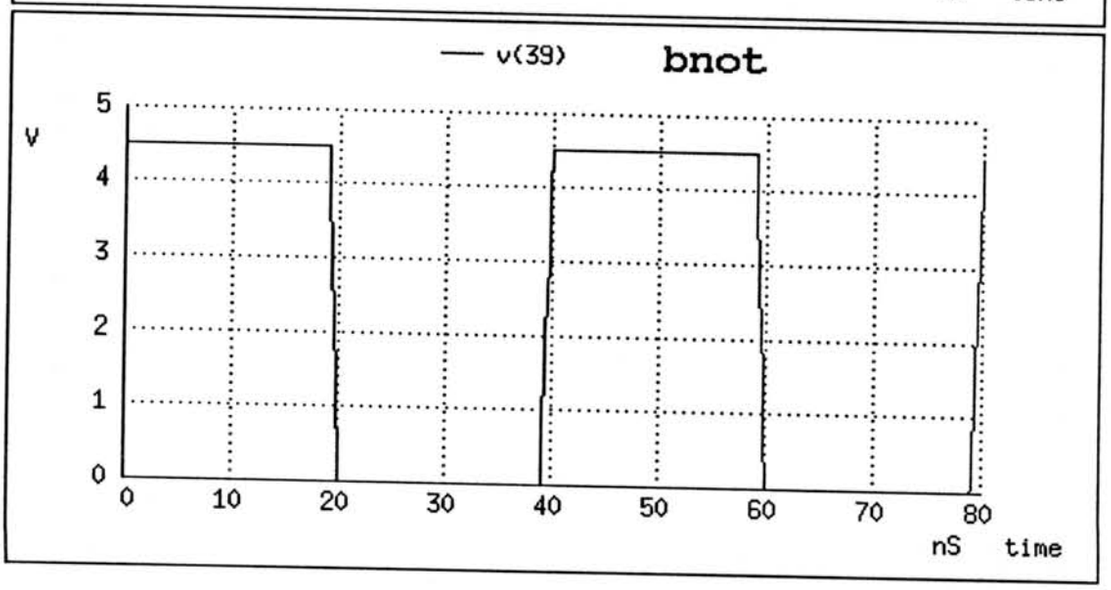

— v(37)
anot
V
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
nS
time


— v(39)
bnot
V
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
nS
time

AND


v(55)
Sout
V
5
4
3
2
1
0
-1
0
10
20
30
40
50
60
70
80
nS
time

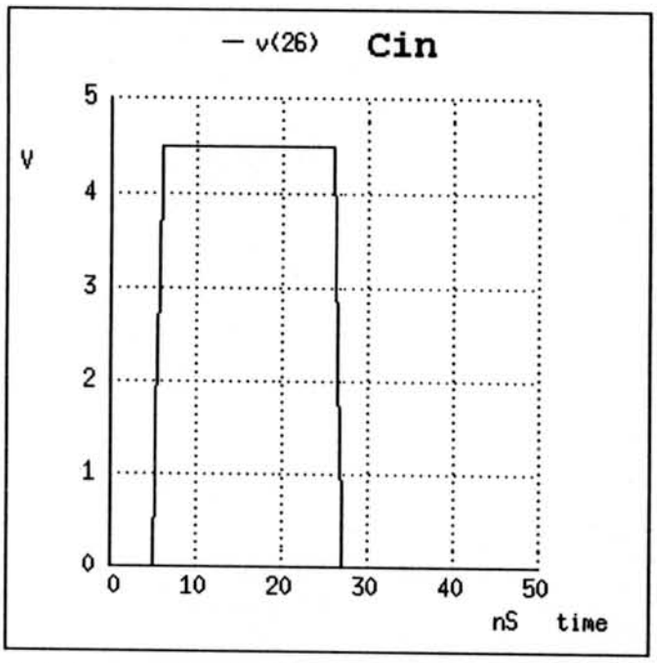

v(37)
anot
V
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
nS
time

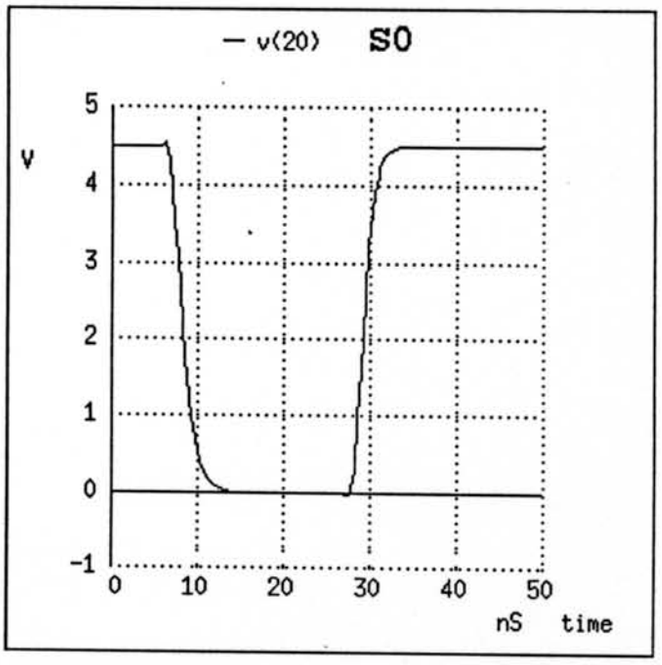

v(39)
bnot
V
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
nS
time

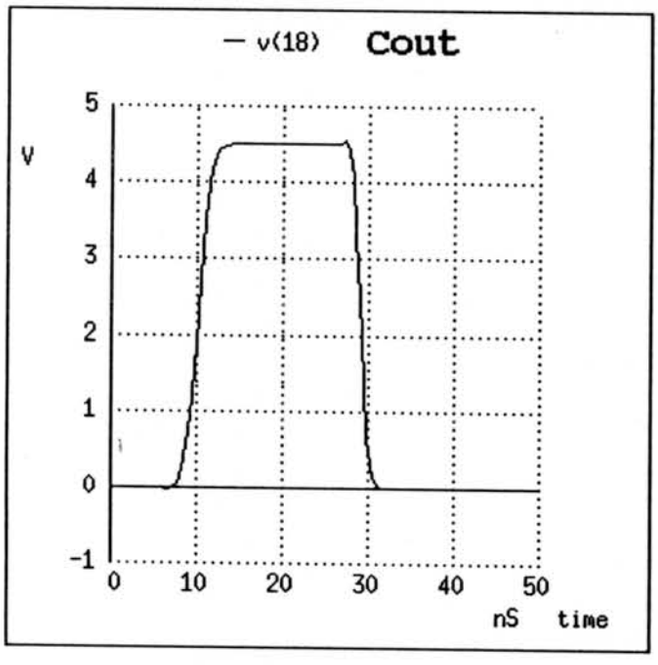
```
* CIRCUITO: ipc.txt
* TECNOLOGIA: cmos12.tec  TIPO CMOS
*
* Modelos dos transistores worst
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P

.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P

* *
* SUBCIRCUITO CORRESPONDENTE A CELULA  ipc.txt
*
* Transistores tipo NMOS: 22
* Transistores tipo PMOS: 22
* No    0 = gnd
* No    1 = vdd2
* No   12 = a1
* No   13 = ai1
* No   14 = b1
* No   15 = b0
* No   16 = a0
* No   17 = ai0
* No   18 = cout
* No   19 = vdd1
* No   20 = s0
* No   22 = ncout
* No   23 = s1
* No   26 = cin
* No   27 = i1
* No   28 = i0
* No   29 = ncin

VDD2 1 0 DC 4.5
VDD1 19 0 DC 4.5
Vnc 29 0 pulse(4.5 0 5n 1n 1n 20n 50n)
Vci 26 0 pulse(0 4.5 5n 1n 1n 20n 50n)

Va0 28 0 dc 4.5
Va1 27 0 dc 4.5

Cs0 20 0 200f
Cs1 23 0 200f
Cco 18 0 220f
Cnc 22 0 150f

MN1     10   34   23    0 NMOS L=1.2U W=12.0U AD=568P AS=39P PD=118U PS=31U
MP2     23   34    1    1 PMOS L=1.2U W=30.0U AD=152P AS=107P PD=70U PS=67U
MN3      9   33   20    0 NMOS L=1.2U W=12.0U AD=467P AS=214P PD=101U PS=61U
MN4      8   32   10    0 NMOS L=1.2U W=12.0U AD=467P AS=568P PD=101U PS=118U
MP5     20   31   19    1 PMOS L=1.2U W=30.0U AD=188P AS=107P PD=72U PS=67U
MP6     20   33   19    1 PMOS L=1.2U W=30.0U AD=188P AS=102P PD=72U PS=67U
MP7     23   30    1    1 PMOS L=1.2U W=30.0U AD=188P AS=129P PD=72U PS=69U
MP8     23   32    1    1 PMOS L=1.2U W=30.0U AD=188P AS=107P PD=72U PS=67U
MN9      9   31    0    0 NMOS L=1.2U W=12.0U AD=467P AS=293P PD=101U PS=116U
MN10     8   30    0    0 NMOS L=1.2U W=12.0U AD=467P AS=293P PD=101U PS=116U
MN11     7   29   33    0 NMOS L=1.2U W=9.0U AD=234P AS=71P PD=70U PS=35U
MP12    33   28   19    1 PMOS L=1.2U W=22.5U AD=141P AS=80P PD=57U PS=52U
MP13    33   29   19    1 PMOS L=1.2U W=22.5U AD=141P AS=78P PD=57U PS=52U
MN14     7   28    0    0 NMOS L=1.2U W=9.0U AD=234P AS=96P PD=70U PS=45U
MN15     6   29   32    0 NMOS L=1.2U W=9.0U AD=283P AS=66P PD=80U PS=33U
```

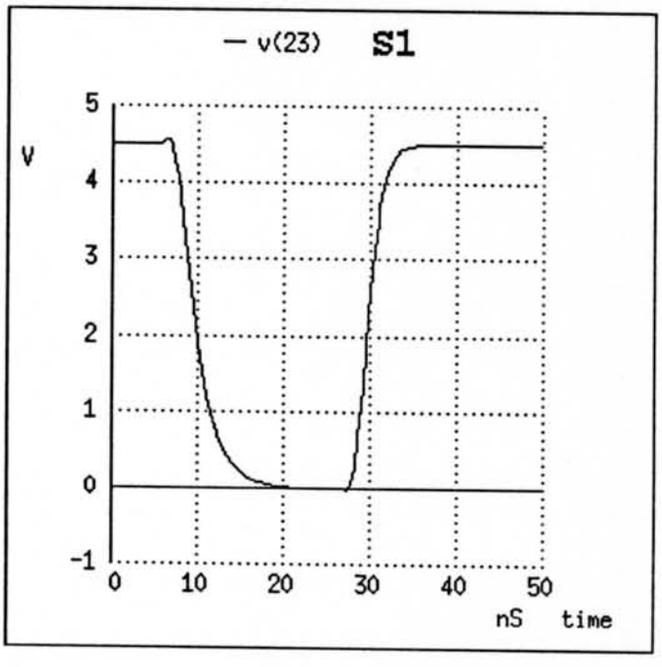
```
MP16    32   29    1    1 PMOS L=1.2U W=22.5U AD=141P AS=80P PD=57U PS=52U
MP17    32   27    1    1 PMOS L=1.2U W=22.5U AD=141P AS=99P PD=57U PS=54U
MN18     5   26   31    0 NMOS L=1.2U W=9.0U AD=267P AS=71P PD=77U PS=35U
MN19     6   27    0    0 NMOS L=1.2U W=9.0U AD=283P AS=361P PD=80U PS=120U
MP20    31   25   19    1 PMOS L=1.2U W=22.5U AD=141P AS=80P PD=57U PS=52U
MP21    31   26   19    1 PMOS L=1.2U W=22.5U AD=141P AS=78P PD=57U PS=52U
MN22     5   25    0    0 NMOS L=1.2U W=9.0U AD=267P AS=96P PD=77U PS=45U
MN23     4   27   34    0 NMOS L=1.2U W=9.0U AD=283P AS=66P PD=80U PS=33U
MP24    34   25    1    1 PMOS L=1.2U W=22.5U AD=141P AS=99P PD=57U PS=54U
MP25    34   27    1    1 PMOS L=1.2U W=22.5U AD=141P AS=80P PD=57U PS=52U
MN26     4   25    0    0 NMOS L=1.2U W=9.0U AD=283P AS=216P PD=80U PS=88U
MN27    25   28    0    0 NMOS L=1.2U W=6.0U AD=187P AS=46P PD=77U PS=27U
MP28    25   28   19    1 PMOS L=1.2U W=15.0U AD=53P AS=48P PD=37U PS=37U
MP29    18   22   19    1 PMOS L=1.2U W=30.0U AD=107P AS=102P PD=67U PS=67U
MN30    18   22    0    0 NMOS L=1.2U W=12.0U AD=383P AS=93P PD=89U PS=39U
MP31    30   28    1    1 PMOS L=1.2U W=22.5U AD=116P AS=80P PD=55U PS=52U
MN32    30   28    3    0 NMOS L=1.2U W=9.0U AD=41P AS=156P PD=27U PS=52U
MN33     2   26    3    0 NMOS L=1.2U W=9.0U AD=251P AS=156P PD=73U PS=52U
MP34    30   26    1    1 PMOS L=1.2U W=22.5U AD=141P AS=80P PD=57U PS=52U
MP35    30   24    1    1 PMOS L=1.2U W=22.5U AD=141P AS=99P PD=57U PS=54U
MN36     2   24    0    0 NMOS L=1.2U W=9.0U AD=251P AS=41P PD=73U PS=27U
MP37    22   26   19    1 PMOS L=1.2U W=22.5U AD=80P AS=78P PD=52U PS=52U
MN38    11   26   22    0 NMOS L=1.2U W=9.0U AD=308P AS=26P PD=86U PS=25U
MN39    21   28   11    0 NMOS L=1.2U W=9.0U AD=426P AS=308P PD=112U PS=86U
MP40    22   28   19    1 PMOS L=1.2U W=22.5U AD=141P AS=78P PD=57U PS=52U
MP41    19   27   22    1 PMOS L=1.2U W=22.5U AD=80P AS=141P PD=52U PS=57U
MN42    21   27    0    0 NMOS L=1.2U W=9.0U AD=426P AS=26P PD=112U PS=25U
MN43    24   27    0    0 NMOS L=1.2U W=6.0U AD=21P AS=192P PD=19U PS=79U
MP44    24   27    1    1 PMOS L=1.2U W=15.0U AD=71P AS=53P PD=40U PS=37U

.tran 1n 50n
.opt temp=125
.END
```

IPC


— v(26) Cin
V
5
4
3
2
1
0
0 10 20 30 40 50
nS time


— v(20) S0
V
5
4
3
2
1
0
-1
0 10 20 30 40 50
nS time


— v(18) Cout
V
5
4
3
2
1
0
-1
0 10 20 30 40 50
nS time


— v(23) S1
V
5
4
3
2
1
0
-1
0 10 20 30 40 50
nS time

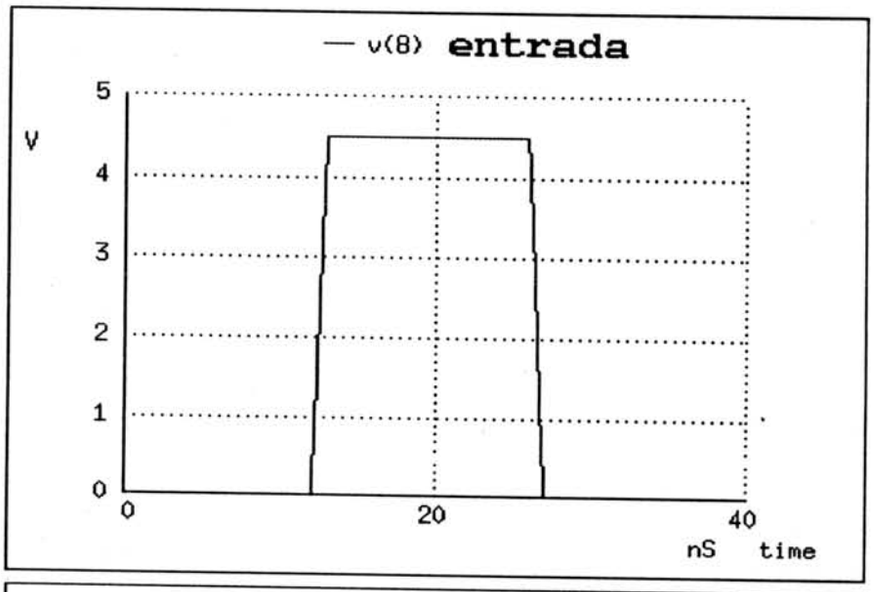
```
*
* CIRCUITO: mux.cel
* TECNOLOGIA: cmos12.tec  TIPO CMOS
*
* Modelos dos transistores worst
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P
*
.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P
* *
* SUBCIRCUITO CORRESPONDENTE A CELULA  mux.cel
*
* Transistores tipo NMOS: 12
* Transistores tipo PMOS: 12
* No    0 = gnd
* No    1 = vdd1
* No    4 = vdd2
* No    6 = i0
* No    8 = a0
* No    9 = s0
* No   10 = b0
* No   12 = i1
* No   14 = a1
* No   15 = s1
* No   16 = b1
* No   17 = li
* No   18 = la
* No   21 = lb

vdd1 1 0 dc 4.5
va0 8 0 pulse(0 4.5 12n 1n 1n 13n 40n)
vb0 10 0 dc 0
vi0 6 0 dc 0
vla 18 0 pulse(4.5 0 10n 1n 1n 3n 15n)
vlb 21 0 dc 0
vli 17 0 dc 0

c0 9 0 300f

MN1     9  21  10   0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP2    10  20   9   1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP3    15  19  16   1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN4    16  21  15   0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN5    20  21   0   0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MP6    20  21   1   1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP7     4  21  19   1 PMOS L=1.2U W=18.3U AD=131P AS=110P PD=50U PS=49U
MN8    19  21   0   0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN9     9  18   8   0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP10    8   7   9   1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP11   15  13  14   1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN12   14  18  15   0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN13    7  18   0   0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MP14    7  18   1   1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP15    4  18  13   1 PMOS L=1.2U W=18.3U AD=131P AS=110P PD=50U PS=49U
MN16   13  18   0   0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN17    9  17   6   0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP18    6   5   9   1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP19   15  11  12   1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN20   12  17  15   0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN21    5  17   0   0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
```

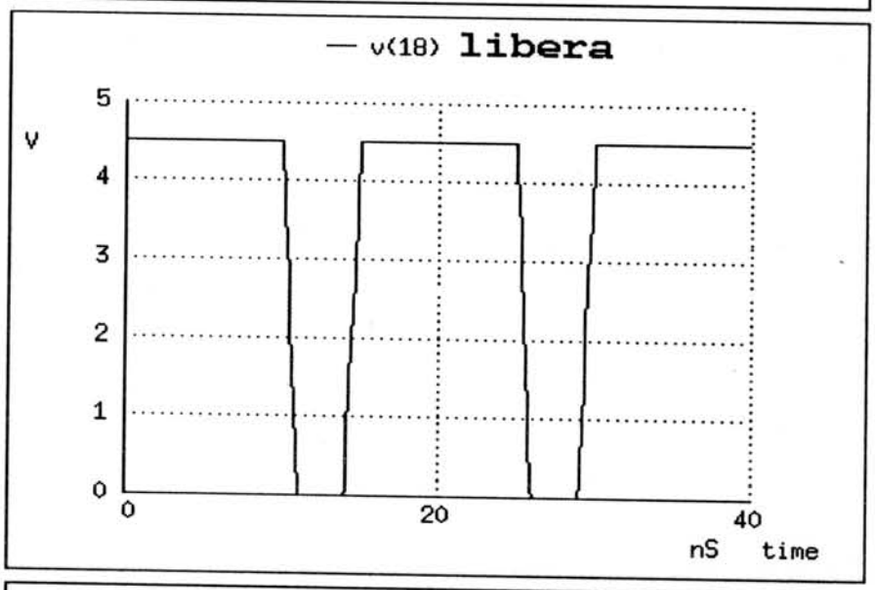
```
MP22     5   17    1    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP23     4   17   11    1 PMOS L=1.2U W=18.3U AD=131P AS=110P PD=50U PS=49U
MN24    11   17    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U

.tran 1n 40n
.opt temp=125
.END
```

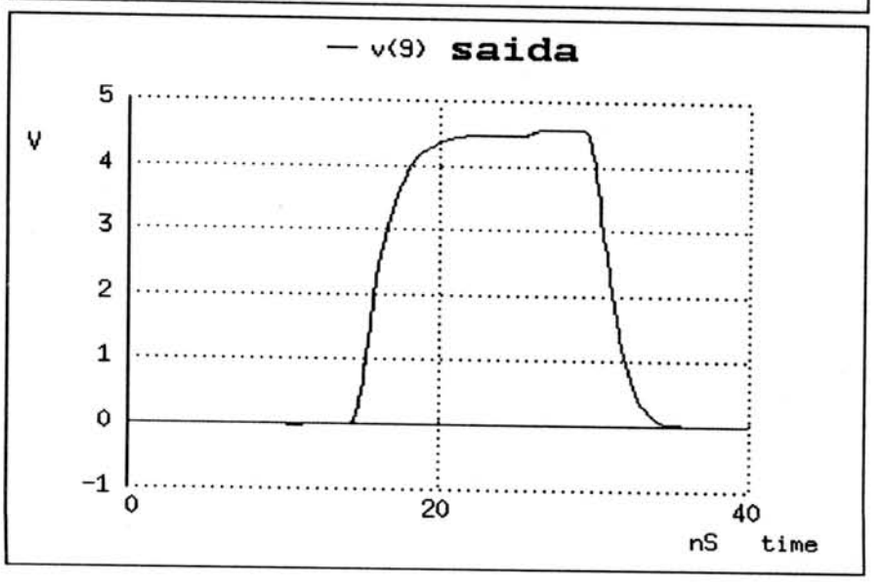
```
* CIRCUITO: mux.cel
* TECNOLOGIA: cmos12.tec  TIPO CMOS
*
* Modelos dos transistores worst
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P
*
.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P
* *
* SUBCIRCUITO CORRESPONDENTE A CELULA  mux.cel
*
* Transistores tipo NMOS: 12
* Transistores tipo PMOS: 12
* No    0 = gnd
* No    1 = vdd1
* No    4 = vdd2
* No    6 = i0
* No    8 = a0
* No    9 = s0
* No   10 = b0
* No   12 = i1
* No   14 = a1
* No   15 = s1
* No   16 = b1
* No   17 = li
* No   18 = la
* No   21 = lb

vdd1 1 0 dc 4.5
va0 8 0 pulse(0 4.5 12n 1n 1n 13n 40n)
vb0 10 0 dc 0
vi0 6 0 dc 0
vla 18 0 pulse(4.5 0 10n 1n 1n 3n 15n)
vlb 21 0 dc 0
vli 17 0 dc 0

c0 9 0 300f

MN1     9   21   10    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP2    10   20    9    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP3    15   19   16    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN4    16   21   15    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN5    20   21    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MP6    20   21    1    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP7     4   21   19    1 PMOS L=1.2U W=18.3U AD=131P AS=110P PD=50U PS=49U
MN8    19   21    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN9     9   18    8    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP10    8    7    9    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP11   15   13   14    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN12   14   18   15    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN13    7   18    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MP14    7   18    1    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP15    4   18   13    1 PMOS L=1.2U W=18.3U AD=131P AS=110P PD=50U PS=49U
MN16   13   18    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN17    9   17    6    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP18    6    5    9    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP19   15   11   12    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN20   12   17   15    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN21    5   17    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MP22    5   17    1    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
```

```
MP23     4   17   11    1 PMOS L=1.2U W=18.3U AD=131P AS=110P PD=50U PS=49U
MN24    11   17    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U

.tran 1n 40n
.opt temp=125
.END
```

MUX


— v(8) entrada
V
5
4
3
2
1
0
0
20
40
nS time

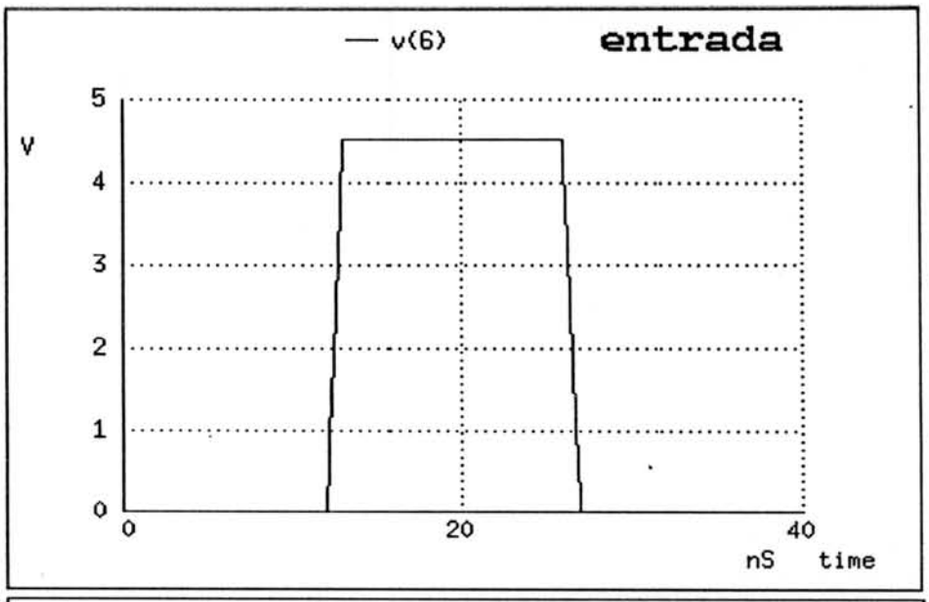

— v(18) libera
V
5
4
3
2
1
0
0
20
40
nS time

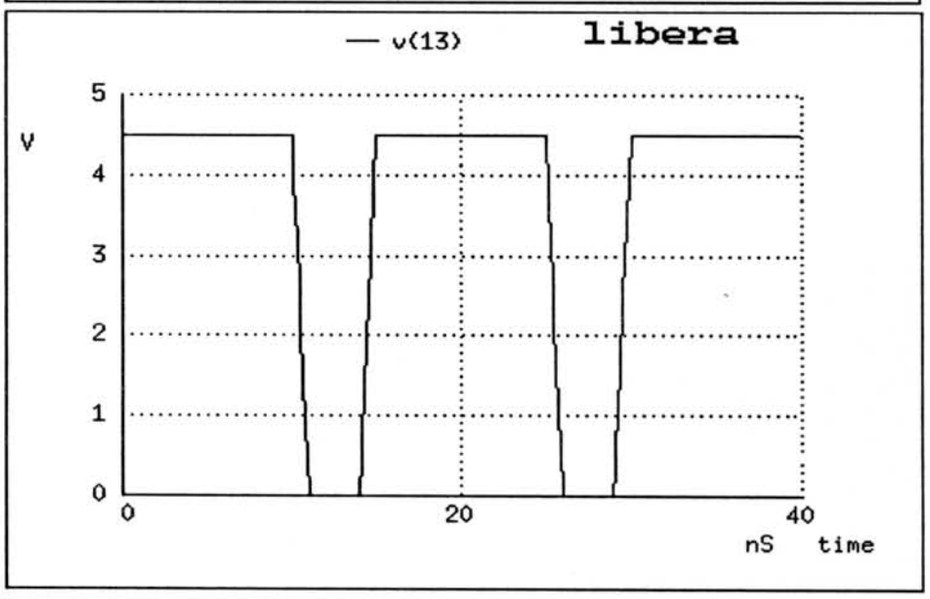

— v(9) saida
V
5
4
3
2
1
0
-1
0
20
40
nS time

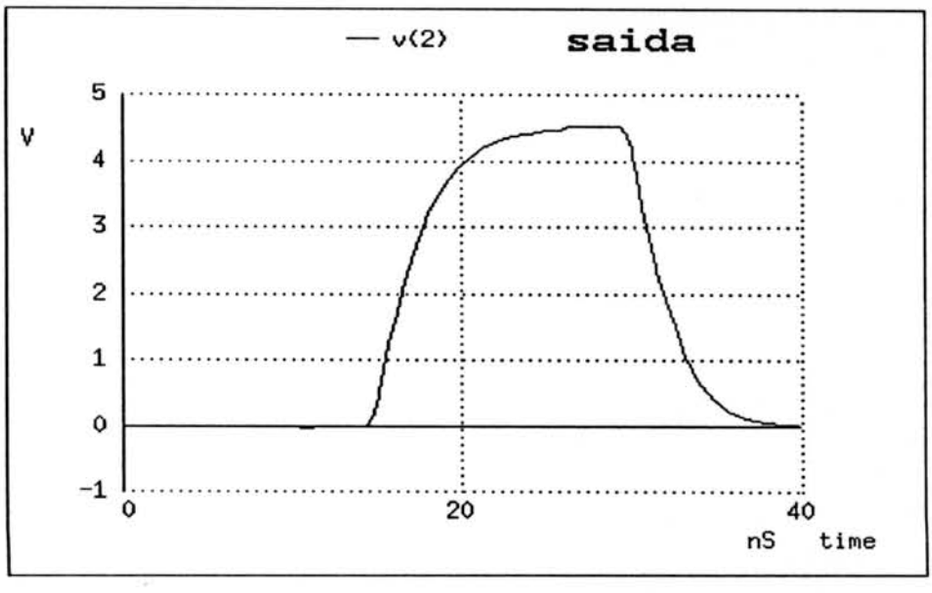
```
* CIRCUITO: muxipc.cel
* TECNOLOGIA: cmos12.tec  TIPO CMOS
*
* Modelos dos transistores worst
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P
*
.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P
* *
* SUBCIRCUITO CORRESPONDENTE A CELULA  muxipc.cel
*
* Transistores tipo NMOS: 8
* Transistores tipo PMOS: 8
* No    0 = gnd
* No    1 = vdd2
* No    2 = a0
* No    3 = vdd1
* No    4 = b1
* No    5 = a1
* No    6 = ai0
* No    7 = b0
* No    8 = ai1
* No   13 = la
* No   14 = lb

vdd1 3 0 dc 4.5
vdd2 1 0 dc 4.5
vai0 6 0 pulse(0 4.5 12n 1n 1n 13n 40n)
vla 13 0 pulse(4.5 0 10n 1n 1n 3n 15n)
vlb 14 0 dc 0

ca 2 0 .8p
cb 7 0 .8p

MP1     12   14    3    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP2     11   13    3    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP3      6   12    7    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP4      6   11    2    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MN5      7   14    6    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MN6      2   13    6    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MN7     12   14    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MN8     11   13    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MN9     10   14    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN10     9   13    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN11     8   14    4    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN12     8   13    5    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MP13     4   10    8    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MP14     5    9    8    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MP15    10   14    1    1 PMOS L=1.2U W=18.3U AD=110P AS=131P PD=49U PS=50U
MP16     9   13    1    1 PMOS L=1.2U W=18.3U AD=110P AS=131P PD=49U PS=50U

.tran 1n 40n
.opt temp=125
.END
```

```
*
* CIRCUITO: muxipc.cel
* TECNOLOGIA: cmos12.tec  TIPO CMOS
*
* Modelos dos transistores worst
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P
*
.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P
* *
* SUBCIRCUITO CORRESPONDENTE A CELULA  muxipc.cel
*
* Transistores tipo NMOS: 8
* Transistores tipo PMOS: 8
* No    0 = gnd
* No    1 = vdd2
* No    2 = a0
* No    3 = vdd1
* No    4 = b1
* No    5 = a1
* No    6 = ai0
* No    7 = b0
* No    8 = ai1
* No   13 = la
* No   14 = lb

vdd1 3 0 dc 4.5
vdd2 1 0 dc 4.5
vai0 6 0 pulse(0 4.5 12n 1n 1n 13n 40n)
vla 13 0 pulse(4.5 0 10n 1n 1n 3n 15n)
vlb 14 0 dc 0

ca 2 0 .8p
cb 7 0 .8p

MP1     12   14    3    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP2     11   13    3    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP3      6   12    7    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP4      6   11    2    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MN5      7   14    6    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MN6      2   13    6    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MN7     12   14    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MN8     11   13    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MN9     10   14    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN10     9   13    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN11     8   14    4    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN12     8   13    5    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MP13     4   10    8    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MP14     5    9    8    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MP15    10   14    1    1 PMOS L=1.2U W=18.3U AD=110P AS=131P PD=49U PS=50U
MP16     9   13    1    1 PMOS L=1.2U W=18.3U AD=110P AS=131P PD=49U PS=50U

.tran 1n 40n
.opt temp=125
.END
```

MUXIPC

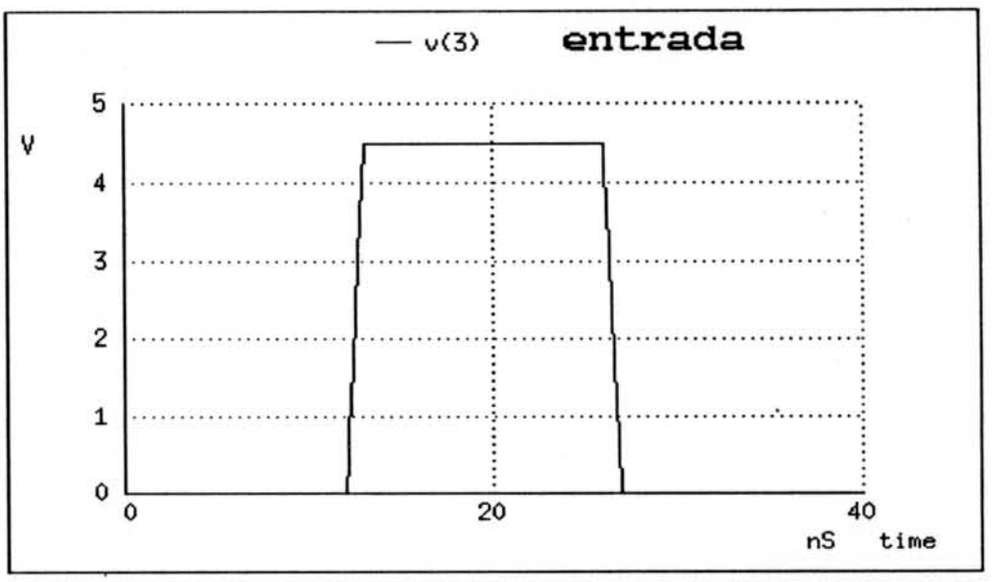

— v(6)
entrada
V
5
4
3
2
1
0
0
20
40
nS
time

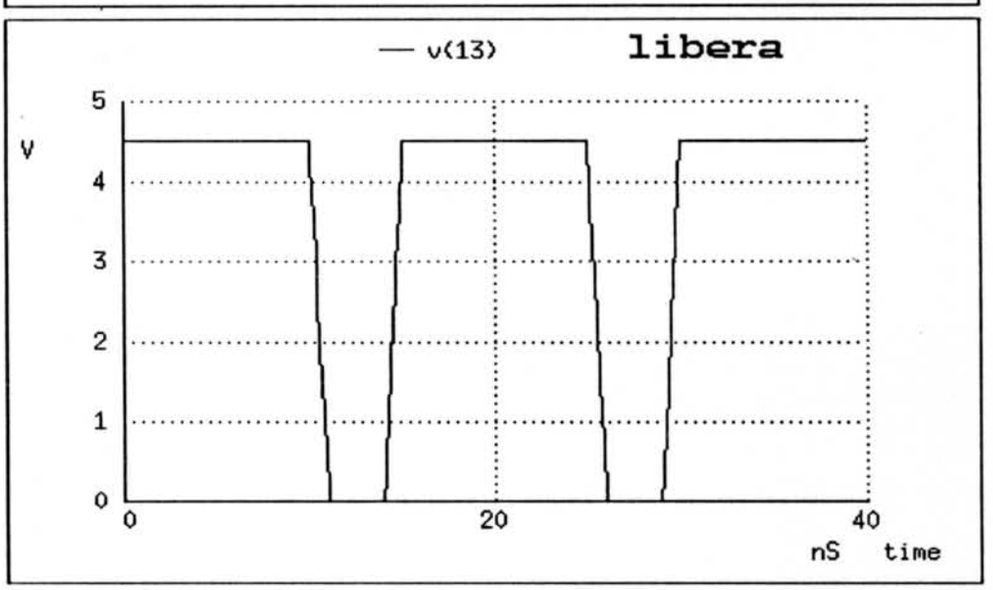

— v(13)
libera
V
5
4
3
2
1
0
0
20
40
nS
time

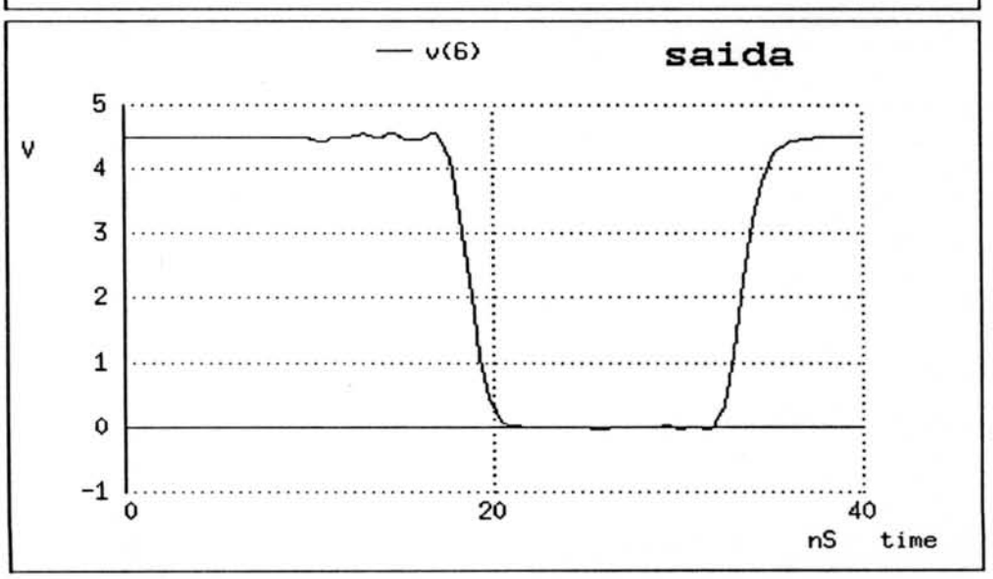

— v(2)
saida
V
5
4
3
2
1
0
-1
0
20
40
nS
time

```
* CIRCUITO: Rau.cel
* TECNOLOGIA: cmos12.tec  TIPO CMOS
*
* Modelos dos transistores worst
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P
*
.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P
* *
* SUBCIRCUITO CORRESPONDENTE A CELULA  Rau.cel
*
* Transistores tipo NMOS: 12
* Transistores tipo PMOS: 14
* No    0 = gnd
* No    1 = vdd2
* No    2 = b0
* No    3 = a0
* No    5 = vdd1
* No    6 = ai0
* No    7 = ai1
* No    9 = a1
* No   11 = b1
* No   13 = 1

vdd1 5 0 dc 4.5
va0 3 0 pulse(0 4.5 12n 1n 1n 13n 40n)
vl 13 0 pulse(4.5 0 10n 1n 1n 3n 15n)

c0 6 0 300f

MN1     7  17   0   0 NMOS L=1.2U W=12.0U AD=57P AS=129P PD=33U PS=45U
MP2     7  17   1   1 PMOS L=1.2U W=14.6U AD=122P AS=105P PD=45U PS=43U
MP3     7  17   1   1 PMOS L=1.2U W=14.6U AD=122P AS=88P PD=45U PS=42U
MP4     6  16   5   1 PMOS L=1.2U W=29.2U AD=274P AS=158P PD=95U PS=81U
MN5     6  16   0   0 NMOS L=1.2U W=12.0U AD=70P AS=129P PD=37U PS=45U
MP7    16   4   5   1 PMOS L=1.2U W=14.6U AD=137P AS=79P PD=66U PS=51U
MP8    17  15   1   1 PMOS L=1.2U W=7.3U AD=61P AS=44P PD=31U PS=27U
MN10   16   4   0   0 NMOS L=1.2U W=6.0U AD=30P AS=64P PD=25U PS=33U
MN11   17  15   0   0 NMOS L=1.2U W=6.0U AD=28P AS=64P PD=21U PS=33U
MP12   17  15   1   1 PMOS L=1.2U W=7.3U AD=61P AS=52P PD=31U PS=28U
MP13    4  10   5   1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN14    4  10   0   0 NMOS L=1.2U W=7.5U AD=44P AS=80P PD=28U PS=36U
MN15   15  14   0   0 NMOS L=1.2U W=7.5U AD=35P AS=80P PD=24U PS=36U
MP16   15  14   1   1 PMOS L=1.2U W=18.3U AD=110P AS=131P PD=49U PS=50U
MP17   17  13  14   1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN18   14  12  17   0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN19   16   8  10   0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP20   10  13  16   1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP21    3   8  10   1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MN22   10  13   3   0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MN23    9  13  14   0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MP24   14  12   9   1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MP25    8  13   5   1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN26    8  13   0   0 NMOS L=1.2U W=7.5U AD=44P AS=80P PD=28U PS=36U
MN27   12  13   0   0 NMOS L=1.2U W=7.5U AD=35P AS=80P PD=24U PS=36U
MP28   12  13   1   1 PMOS L=1.2U W=18.3U AD=110P AS=131P PD=49U PS=50U

.tran 1n 40n
.opt temp=125
.END
```

Rau


— v(3)
entrada
V
5
4
3
2
1
0
0
20
40
nS
time

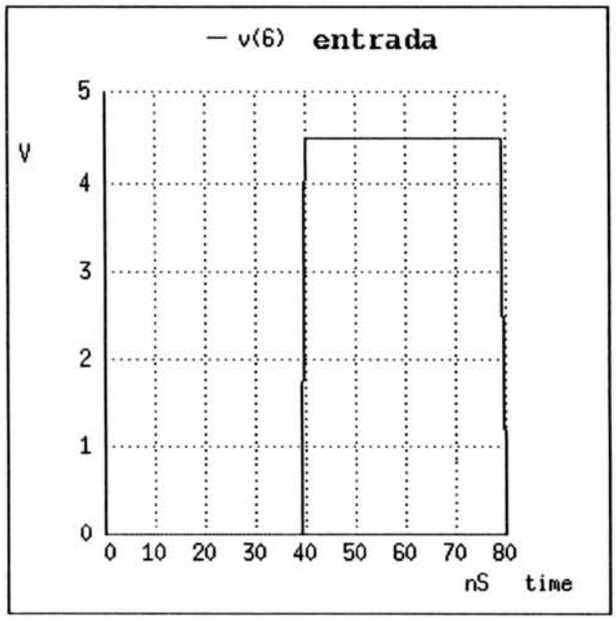

— v(13)
libera
V
5
4
3
2
1
0
0
20
40
nS
time

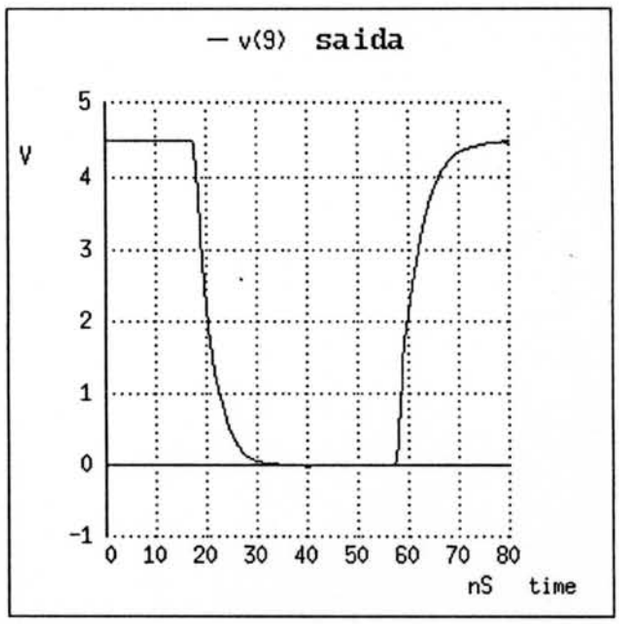

— v(6)
saida
V
5
4
3
2
1
0
-1
0
20
40
nS
time

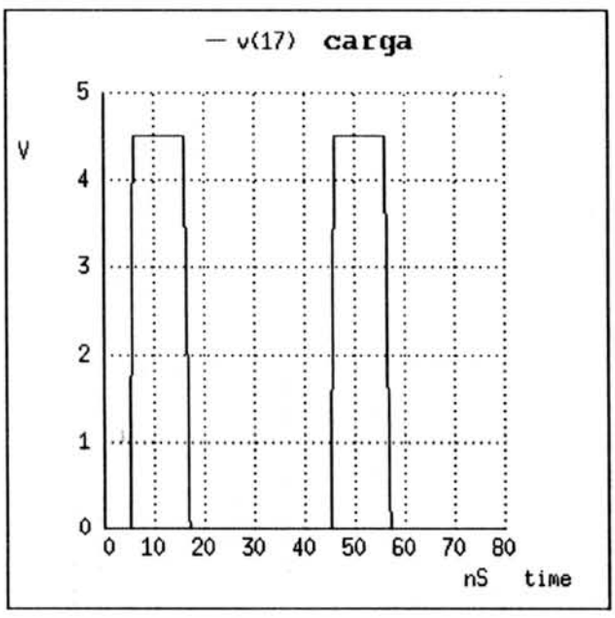
```
*
* CIRCUITO: RAC.cel
* TECNOLOGIA: cmos12.tec  TIPO CMOS
*
* Modelos dos transistores worst
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P
*
.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P
* *
* SUBCIRCUITO CORRESPONDENTE A CELULA  RAC.cel
*
* Transistores tipo NMOS: 14
* Transistores tipo PMOS: 15
* No    0 = gnd
* No    1 = vdd1
* No    2 = bi1
* No    3 = b0
* No    4 = bi0
* No    6 = ai0
* No    8 = b1
* No    9 = a0
* No   10 = a1
* No   11 = ai1
* No   14 = vdd2
* No   17 = cg
* No   21 = l

vdd1 1 0 dc 4.5
va0 6 0 pulse(0 4.5 39n 1n 1n 39n 80n)
vcg 17 0 pulse(0 4.5 5n 1n 1n 10n 40n)
vl 21 0 pulse(0 4.5 17n 1n 1n 21n 40n)

c0 9 0 .8p

r2 7 1 10meg
r3 9 1 10meg
r4 12 1 10meg
r5 13 1 10meg
r6 15 1 10meg
r8 19 1 10meg
r9 18 1 10meg
r10 19 1 10meg

MP1     10   22   13    1 PMOS L=1.2U W=21.9U AD=157P AS=141P PD=58U PS=56U
MN2     13   21   10    0 NMOS L=1.2U W=9.0U AD=43P AS=108P PD=27U PS=42U
MN3      9   21   15    0 NMOS L=1.2U W=9.0U AD=43P AS=108P PD=27U PS=42U
MP4     15   20    9    1 PMOS L=1.2U W=21.9U AD=157P AS=141P PD=58U PS=56U
MP5     14   21   22    1 PMOS L=1.2U W=21.9U AD=157P AS=141P PD=58U PS=56U
MN6     22   21    0    0 NMOS L=1.2U W=9.0U AD=43P AS=108P PD=27U PS=42U
MN7     20   21    0    0 NMOS L=1.2U W=9.0U AD=50P AS=108P PD=31U PS=42U
MP8     20   21    1    1 PMOS L=1.2U W=21.9U AD=105P AS=170P PD=53U PS=59U
MP9     15    7    1    1 PMOS L=1.2U W=14.6U AD=137P AS=70P PD=47U PS=38U
MP10    13   19   14    1 PMOS L=1.2U W=29.2U AD=245P AS=193P PD=91U PS=85U
MP11    15    7    1    1 PMOS L=1.2U W=14.6U AD=137P AS=107P PD=47U PS=44U
MN12    15    7    0    0 NMOS L=1.2U W=12.0U AD=70P AS=145P PD=37U PS=48U
MN13    13   19    0    0 NMOS L=1.2U W=12.0U AD=57P AS=145P PD=33U PS=48U
MP15    14   12   19    1 PMOS L=1.2U W=18.3U AD=131P AS=110P PD=50U PS=49U
MN16    19   12    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN17     7   18    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
```

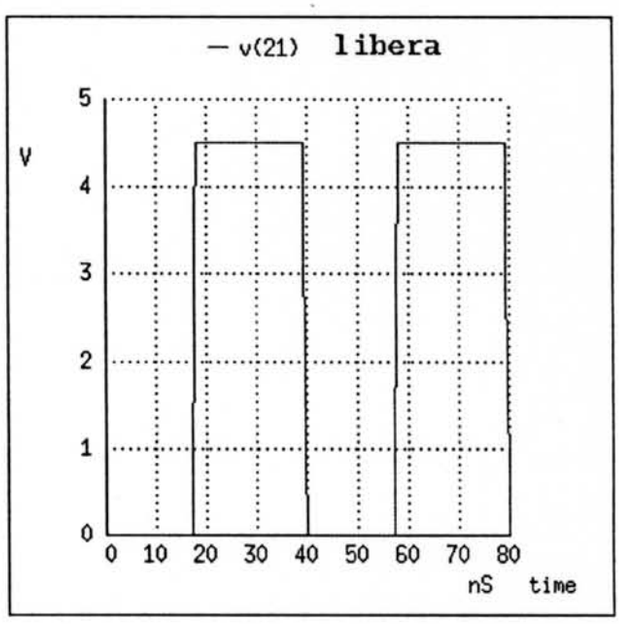
```
MP18     7   18    1    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U
MP19    13   17   12    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN20    12    5   13    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN21    15   16   18    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP22    18   17   15    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP23    12    5   11    1 PMOS L=1.2U W=18.3U AD=87P AS=110P PD=46U PS=49U
MN24    11   17   12    0 NMOS L=1.2U W=7.5U AD=43P AS=44P PD=26U PS=28U
MN25    18   17    6    0 NMOS L=1.2U W=7.5U AD=35P AS=44P PD=24U PS=28U
MP26     6   16   18    1 PMOS L=1.2U W=18.3U AD=105P AS=110P PD=48U PS=49U
MP27    14   17    5    1 PMOS L=1.2U W=18.3U AD=131P AS=110P PD=50U PS=49U
MN28     5   17    0    0 NMOS L=1.2U W=7.5U AD=35P AS=90P PD=24U PS=39U
MN29    16   17    0    0 NMOS L=1.2U W=7.5U AD=44P AS=90P PD=28U PS=39U
MP30    16   17    1    1 PMOS L=1.2U W=18.3U AD=87P AS=134P PD=46U PS=52U

.TRAN 1N 80N
.OPT TEMP=125
.END
```

RAC


— v(6) entrada
V
5
4
3
2
1
0
0 10 20 30 40 50 60 70 80
nS time


— v(9) saida
V
5
4
3
2
1
0
-1
0 10 20 30 40 50 60 70 80
nS time


— v(17) carga
V
5
4
3
2
1
0
0 10 20 30 40 50 60 70 80
nS time


— v(21) libera
V
5
4
3
2
1
0
0 10 20 30 40 50 60 70 80
nS time

```
CIRCUITO: tpass.cel

* Modelos dos transistores
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P

.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P

* Transistores tipo NMOS: 2
* Transistores tipo PMOS: 0
* No    7 = out2
* No    8 = in2
* No    9 = in
* No   10 = out
* No   12 = com

.SUBCKT fonte 30 32 vdd
M1 31 30 vdd vdd PMOS L=1.2U W=7.5U
M2  0 30  31   0 NMOS L=1.2U W=3.0U
M3 32 31 vdd vdd PMOS L=1.2U W=15U
M4  0 31  32   0 NMOS L=1.2U W=6.0U
.ENDS fonte

.SUBCKT fonte2 30 32 vdd
M1 31 30 vdd vdd PMOS L=1.2U W=15U
M2  0 30  31   0 NMOS L=1.2U W=6U
M3 32 31 vdd vdd PMOS L=1.2U W=38U
M4  0 31  32   0 NMOS L=1.2U W=15U
.ENDS fonte2

.SUBCKT carga 40 vdd
* carga de 0.8pF *
M1 41 40 vdd vdd PMOS L=1.2U W=230U
M2 41 40  0   0  NMOS L=1.2U W=230U
C1 41 0 1PF
.ENDS carga

vdd 1 0 dc 4.5
vcm 2 0 pulse(0 4.5 12ns 1ns 1ns 7ns 25ns)
vpc 3 0 pulse(4.5 0  2ns 1ns 1ns 4ns 25ns)

X1 2 12 1 fonte
X2 3  5 1 fonte
X3  7 1 carga
X4 10 1 carga
R8 8 0 10
R9 9 0 10
R1 10 0 5MEG
R2  7 0 5MEG
R3 12 0 5MEG
MN1  9  12  10  0 NMOS L=1.2U W=12.0U
MN2  8  12   7  0 NMOS L=1.2U W=12.0U

MP1  1   5   7  1 PMOS L=1.2U W=50U
MP2  1   5  10  1 PMOS L=1.2U W=50U

.tran 1ns 55ns
.opt temp=125
.END
```

muxin.cir

```
* Modelos dos transistores
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P

.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P


* No    0 = gnd
* No    1 = vdd
* No    5 = b1
* No    6 = out2
* No    7 = b0
* No    8 = lig
* No    9 = out
* No   11 = mb
* No   12 = nmb
* No   13 = mlig
* No   14 = nmlig

.SUBCKT fonte 30 32 vdd
M1 31 30 vdd vdd PMOS L=1.2U W=7.5U
M2  0 30  31   0 NMOS L=1.2U W=3U
M3 32 31 vdd vdd PMOS L=1.2U W=15U
M4  0 31  32   0 NMOS L=1.2U W=6U
.ENDS fonte

.SUBCKT fonte2 30 32 vdd
M1 31 30 vdd vdd PMOS L=1.2U W=25U
M2  0 30  31   0 NMOS L=1.2U W=10U
M3 32 31 vdd vdd PMOS L=1.2U W=60U
M4  0 31  32   0 NMOS L=1.2U W=25U
.ENDS fonte2


.SUBCKT carga 40 vdd
* carga de 0.5pF *
M1 41 40 vdd vdd PMOS L=1.2U W=145U
M2 41 40  0   0  NMOS L=1.2U W=145U
C1 41 0 0.5PF
.ENDS carga

vdd 1 0 dc 4.5
vin 2 0 pulse(0 4.5 10ns 1ns 1ns 8ns 30ns)
vcn 3 0 pulse(0 4.5 1ns 1ns 1ns 26ns 30ns)
vcp 4 0 pulse(4.5 0 1ns 1ns 1ns 26ns 30ns)
X1 2  7 1 fonte2
X2 3 11 1 fonte2
X3 4 12 1 fonte2
X4 9 1 carga

R1  7 0 5MEG
R2  9 0 5MEG
R3 12 0 5MEG
R4 11 0 5MEG
R5 14 1 100
R6 13 0 100
R7  8 0 100
R8 10 0 100
* BIT1 *
```

```
MP1      8   14    9    1 PMOS L=1.2U W=30U
MN4      8   13    9    0 NMOS L=1.2U W=12U
MP6      7   12    9    1 PMOS L=1.2U W=30U
MN8      7   11    9    0 NMOS L=1.2U W=12U
MN10     9   10    0    0 NMOS L=1.2U W=15U
* BIT2 *
*MP2      8   14    6    1 PMOS L=1.2U W=30U
*MN3      8   13    6    0 NMOS L=1.2U W=12U
*MP5      5   12    6    1 PMOS L=1.2U W=30U
*MN7      5   11    6    0 NMOS L=1.2U W=12U
*MN9      6   10    0    0 NMOS L=1.2U W=15U
.tran 1ns 50ns
.opt temp=125

.END
```

latchd.cel

```
* modelo worst
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P

.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P

* Transistores tipo NMOS: 8
* Transistores tipo PMOS: 8
* No   0 = gnd
* No   1 = vdd2
* No   2 = in2
* No   3 = in
* No   4 = vdd
* No   5 = out2
* No   6 = out

.SUBCKT fonte 30 32 vdd
M1 31 30 vdd vdd PMOS L=1.2U W=6U
M2  0 30  31   0 NMOS L=1.2U W=3U
M3 32 31 vdd vdd PMOS L=1.2U W=6U
M4  0 31  32   0 NMOS L=1.2U W=3U
C10 32 0 0.1pf
.ENDS fonte

.SUBCKT carga 40 vdd
* Equivale a uma capacitancia de aprox. 1PF *
M1 41 40 vdd vdd PMOS L=1.2U W=290U
M2 41 40  0   0 NMOS L=1.2U W=290U
C1 41 0 1PF
.ENDS carga

vdd1 4 0 dc 4.5
vdd2 1 0 dc 4.5
vin1 30 0 pulse(0 4.5 5ns 1ns 1ns 12ns 24ns)
vin2 31 0 pulse(0 4.5 5ns 1ns 1ns 12ns 24ns)

X1 30 3 1 fonte
X2 31 2 1 fonte

Rout1 5 0 4MEG
Rout2 6 0 4MEG
X3 6 1 carga
* BIT 1 *
MP1     7   3   4   1 PMOS L=1.2U W=15.0U
MP2     6   7   4   1 PMOS L=1.2U W=37.0U
MP3    14   7   4   1 PMOS L=1.2U W=19.5U
MP4     7  14   4   1 PMOS L=1.2U W=10.0U
MN5     7   3   0   0 NMOS L=1.2U W=6.0U
MN6     6   7   0   0 NMOS L=1.2U W=15.0U
MN7    14   7   0   0 NMOS L=1.2U W=8.00U
MN8     7  14   0   0 NMOS L=1.2U W=4.20U
* BIT 2 *
MN9     8   2   0   0 NMOS L=1.2U W=6.0U
MN10    5   8   0   0 NMOS L=1.2U W=9.0U
MN11   13   8   0   0 NMOS L=1.2U W=6.0U
MN12    8  13   0   0 NMOS L=1.2U W=6.0U
MP13    8   2   1   1 PMOS L=1.2U W=12.0U
MP14    5   8   1   1 PMOS L=1.2U W=18.0U
MP15   13   8   1   1 PMOS L=1.2U W=12.0U
```

```
MP16    8  13   1   1 PMOS L=1.2U W=12.0U

.tran 1ns 48ns
.opt temp=125

.END
```

latche.cel

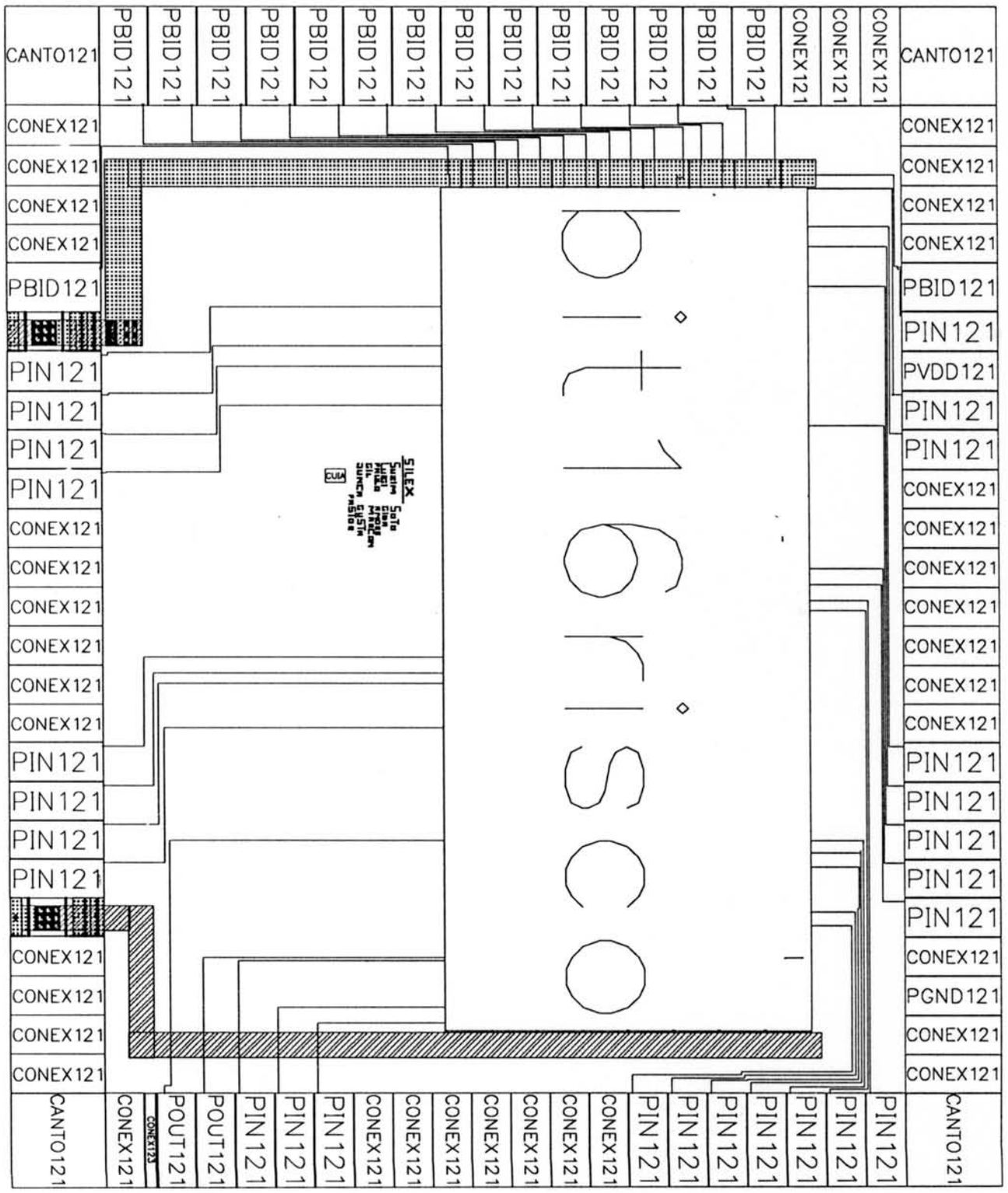
```
*modelo worst
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P

.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P

* Transistores tipo NMOS: 8
* Transistores tipo PMOS: 8
* No    0 = gnd
* No    1 = vdd2
* No    2 = in2
* No    4 = in
* No    6 = vdd
* No    7 = out2
* No    8 = out
MP1      6   14    8    1 PMOS L=1.2U W=3.0U AD=11P AS=20P PD=13U PS=19U
MP2      6    4   14    1 PMOS L=1.2U W=3.0U AD=18P AS=24P PD=18U PS=22U
MP3      6    5   14    1 PMOS L=1.2U W=3.0U AD=11P AS=20P PD=13U PS=19U
MP4      6   14    5    1 PMOS L=1.2U W=3.0U AD=11P AS=20P PD=13U PS=19U
MN5      8   14    0    0 NMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
MN6     14    4    0    0 NMOS L=1.2U W=3.0U AD=24P AS=18P PD=22U PS=18U
MN7     14    5    0    0 NMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
MN8      5   14    0    0 NMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
MN9      7   13    0    0 NMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
MN10    13    2    0    0 NMOS L=1.2U W=3.0U AD=24P AS=18P PD=22U PS=18U
MN11    13    3    0    0 NMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
MN12     3   13    0    0 NMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
MP13     7   13    1    1 PMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
MP14    13    2    1    1 PMOS L=1.2U W=3.0U AD=24P AS=18P PD=22U PS=18U
MP15    13    3    1    1 PMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
MP16     3   13    1    1 PMOS L=1.2U W=3.0U AD=20P AS=11P PD=19U PS=13U
.END
```

```
* Barramentos Independentes de Leitura e Escrita
*
.MODEL NMOS NMOS LEVEL=2 LD=0.06U TOX=275E-10 NSUB=2E16 VTO=0.81
+UO=510 UEXP=0.22 UCRIT=24.3K DELTA=0.4 XJ=0.4U VMAX=54K NEFF=4
+RSH=62 NFS=0 JS=2U CJ=150U CJSW=720P MJ=0.53 MJSW=0.53 PB=0.68V
+CGDO=270P CGSO=270P
*
.MODEL PMOS PMOS LEVEL=2 LD=0.03U TOX=275E-10 NSUB=5E16 VTO=-1.23
+UO=210 UEXP=0.33 UCRIT=51K DELTA=0.4 XJ=0.5U VMAX=47K NEFF=0.88
+RSH=85 NFS=0 JS=10U CJ=560U CJSW=672P MJ=0.46 MJSW=0.46 PB=0.78V
+CGDO=270P CGSO=270P
*

* FONTES PARA ESCRITA

M1 1 2 4 1 PMOS  L=1.2U  W=18U  AD=60p  AS=60p  PD=42.6u  PS=42.6u
M2 4 3 0 0 NMOS  L=1.2U  W=12U  AD=40p  AS=40p  PD=30.6u  PS=30.6u

* ESCREVE

M3 4 10 7 0 NMOS  L=1.2U  W=3U AD=10p  AS=10p  PD=12.6u  PS=12.6u

* LE
M40 8 0 81 0 NMOS  L=1.2U  W=4.5U AD=15p  AS=15p  PD=15.6u  PS=15.6u
M4 8 11 5 0 NMOS  L=1.2U  W=4.5U AD=15p  AS=15p  PD=15.6u  PS=15.6u

* PRE-CARGA PARA LEITURA

M5 1 6 5 1 PMOS  L=1.2U  W=18U  AD=60p  AS=60p  PD=42.6u  PS=42.6u
M6 5 0 0 0 NMOS  L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u

* "FLIP-FLOP"

M7  1 7 8 1 PMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u
M8  8 7 0 0 NMOS L=1.2U  W=16.5U  AD=55p  AS=55p  PD=39.6u  PS=39.6u

M9  1 8 7 1 PMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u
M10 7 8 0 0 NMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u

M11   1 111 112 1 PMOS L=1.2U  W=4.5U  AD=15P  AS=15P  PD=15.6U  PS=15.6U
M12 112 111   0 0 NMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u
M13   1 112 113 1 PMOS L=1.2U  W=4.5U  AD=15P  AS=15P  PD=15.6U  PS=15.6U
M14 113 112   0 0 NMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u
M15   1 113  11 1 PMOS L=1.2U  W=4.5U  AD=15P  AS=15P  PD=15.6U  PS=15.6U
M16  11 113   0 0 NMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u

M110   1 101 102 1 PMOS L=1.2U  W=4.5U  AD=15P  AS=15P  PD=15.6U  PS=15.6U
M120 102 101   0 0 NMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u
M130   1 102 103 1 PMOS L=1.2U  W=4.5U  AD=15P  AS=15P  PD=15.6U  PS=15.6U
M140 103 102   0 0 NMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u
M150   1 103  10 1 PMOS L=1.2U  W=4.5U  AD=15P  AS=15P  PD=15.6U  PS=15.6U
M160  10 103   0 0 NMOS L=1.2U  W=3U  AD=10p  AS=10p  PD=12.6u  PS=12.6u

*

Cbesc 4 0 460F
Cble  5 0 515F

Cselesc 10 0 80F
Csel_le 11 0 80F
R7 7 0 5MEG
*
VDD 1 0 DC 4.5

Vescl 2 0 pulse (4.5 0 5n 1n 1n 25n 120n)
Vesc0 3 0 pulse (0 4.5 45n 1n 1n 25n 120n)
```

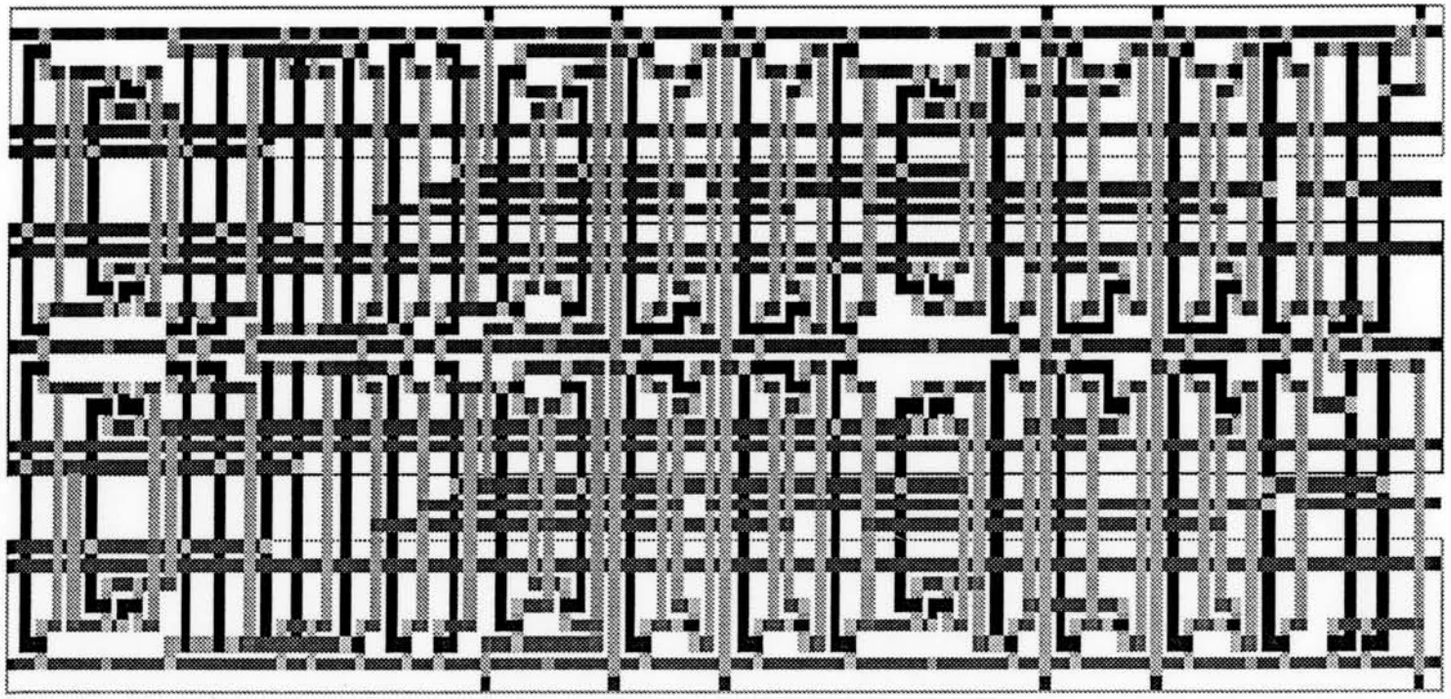
```
VSesc 101 0 pulse (4.5 0 11n 1n 1n 19n 50n)

Vpre 6 0 pulse (4.5 0 32n 1n 1n 6n 50n)

VSle 111 0 pulse (4.5 0 40n 1n 1n 18n 60n)
*

*
.TRAN .2N 100N
.OPT temp=125
.PRINT TRAN V(4) V(5) V(7) V(8)
.END
```

ANEXO A-4

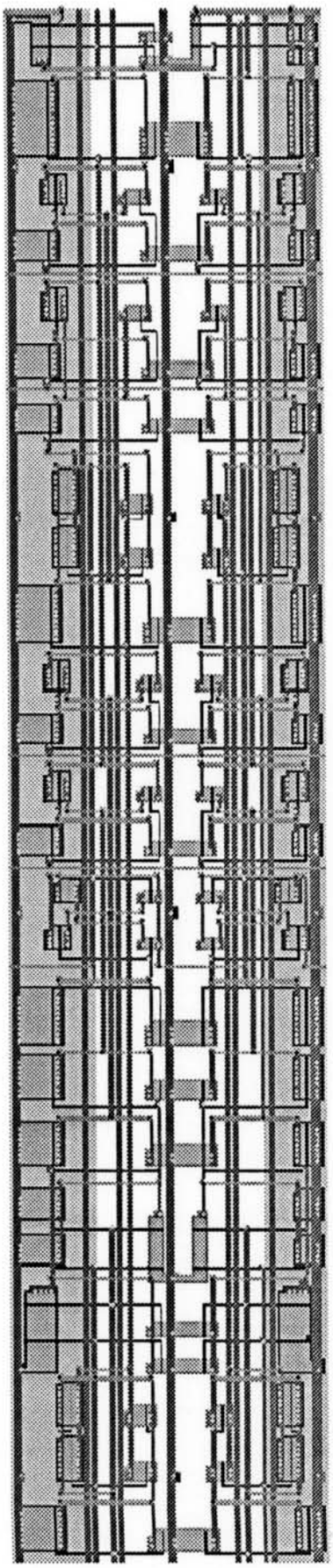
Layout de blocos e test-chip


CANTO121
PBID121
CONEX121
PIN121
POUT121
CONEX123
PVDD121
PGND121
bit16risc
SILEX
EUIA

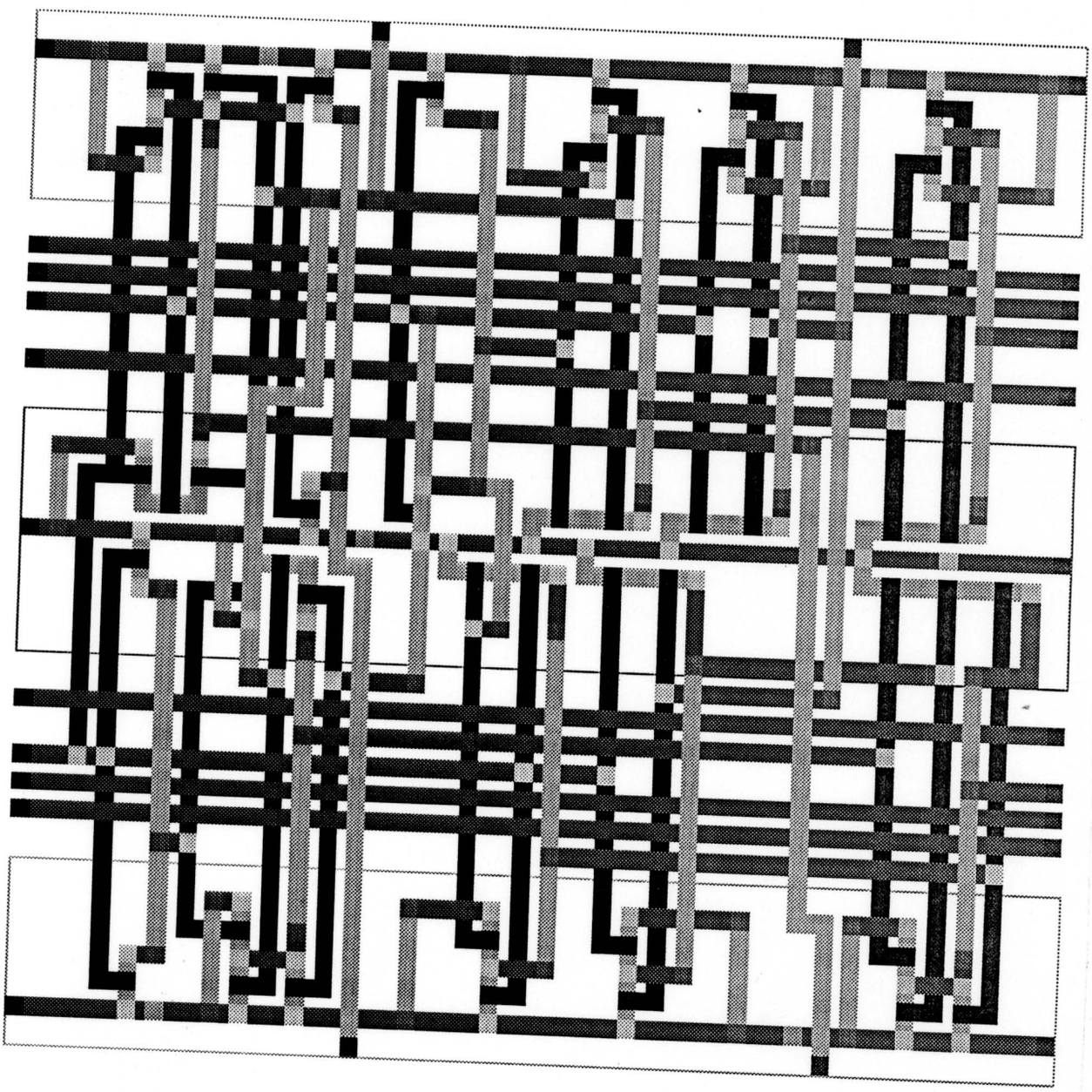
ULA - Simbolico



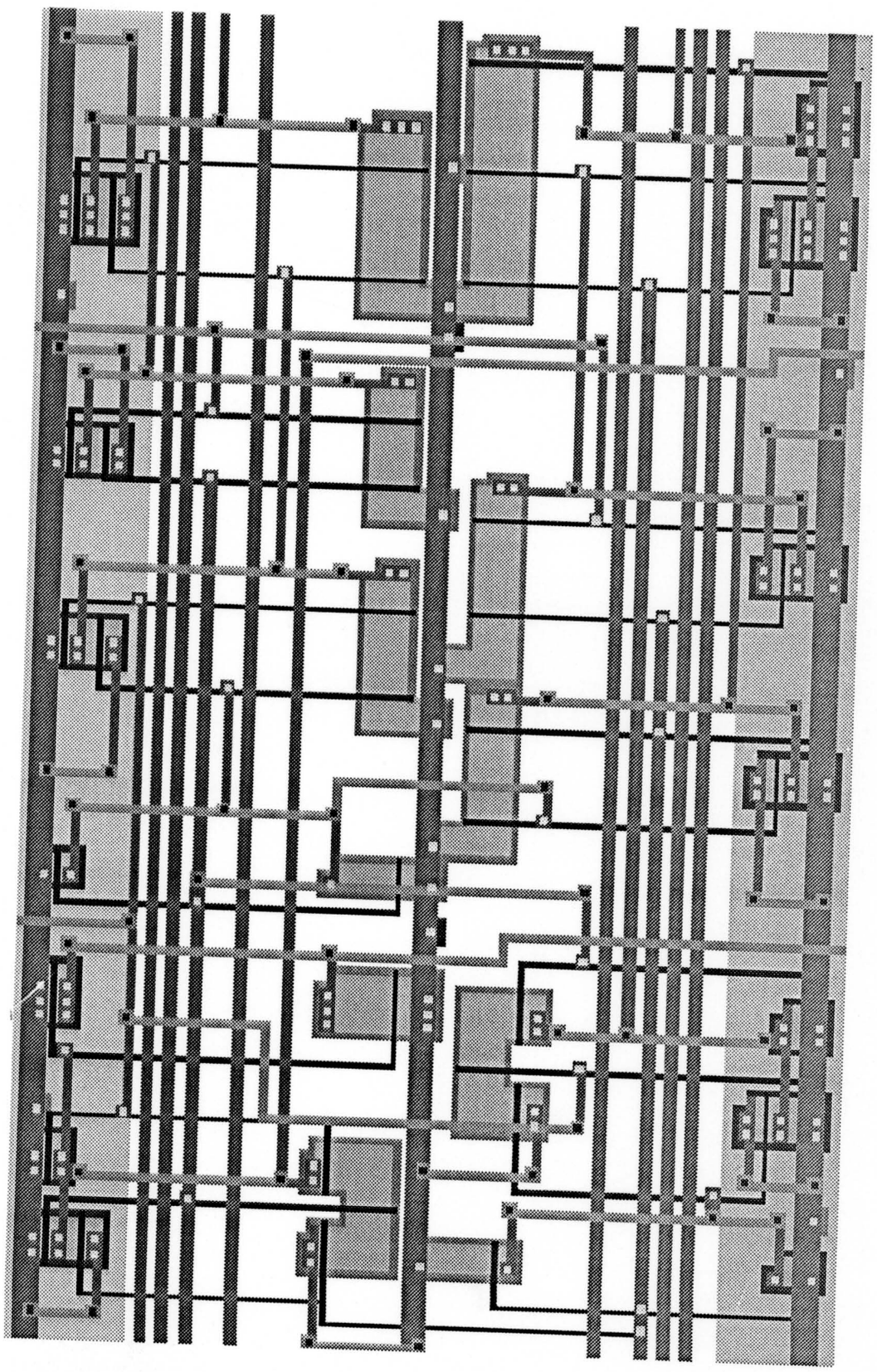
ULA - Layout



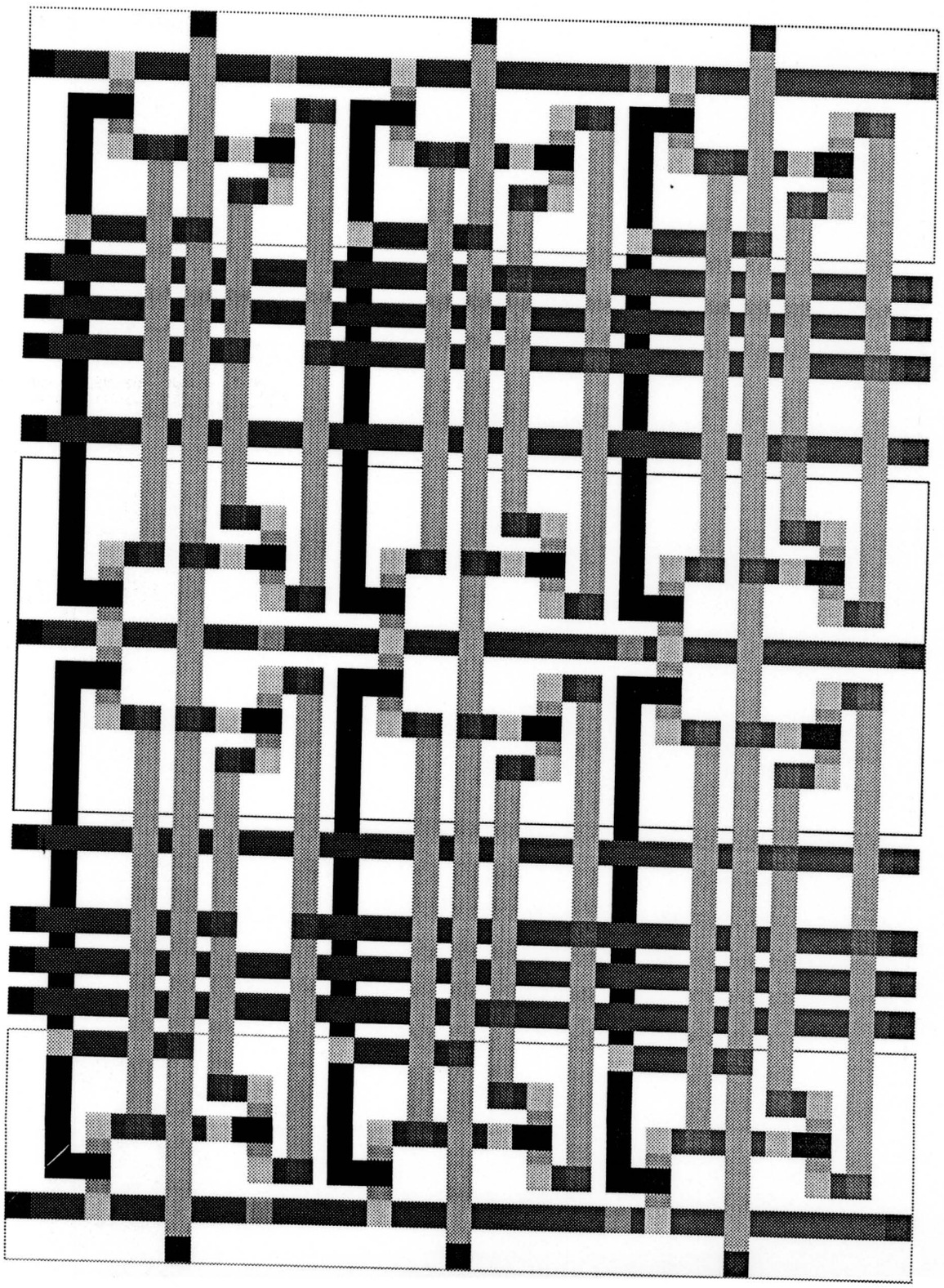
IPC - Simbolico



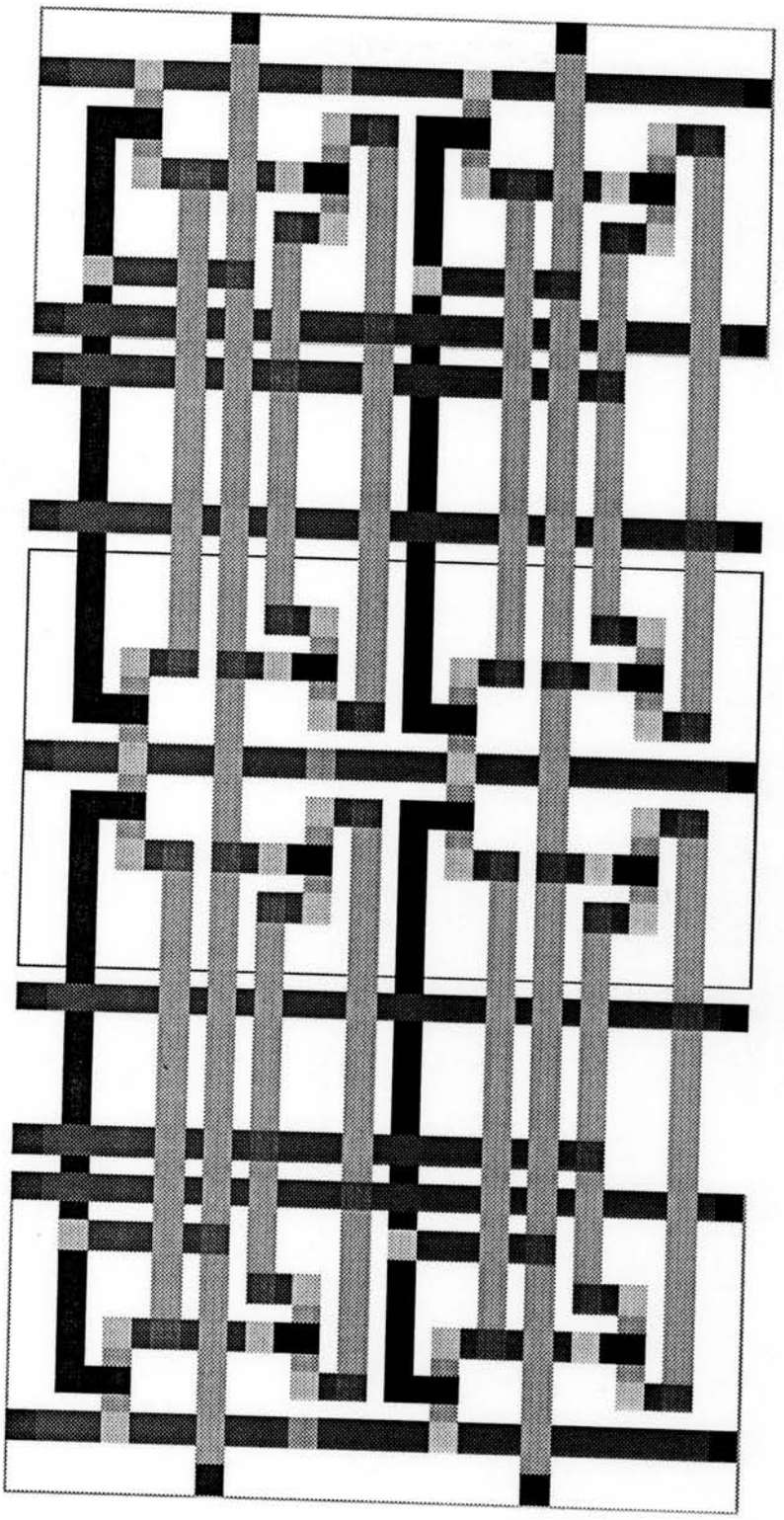
IPC - Layout



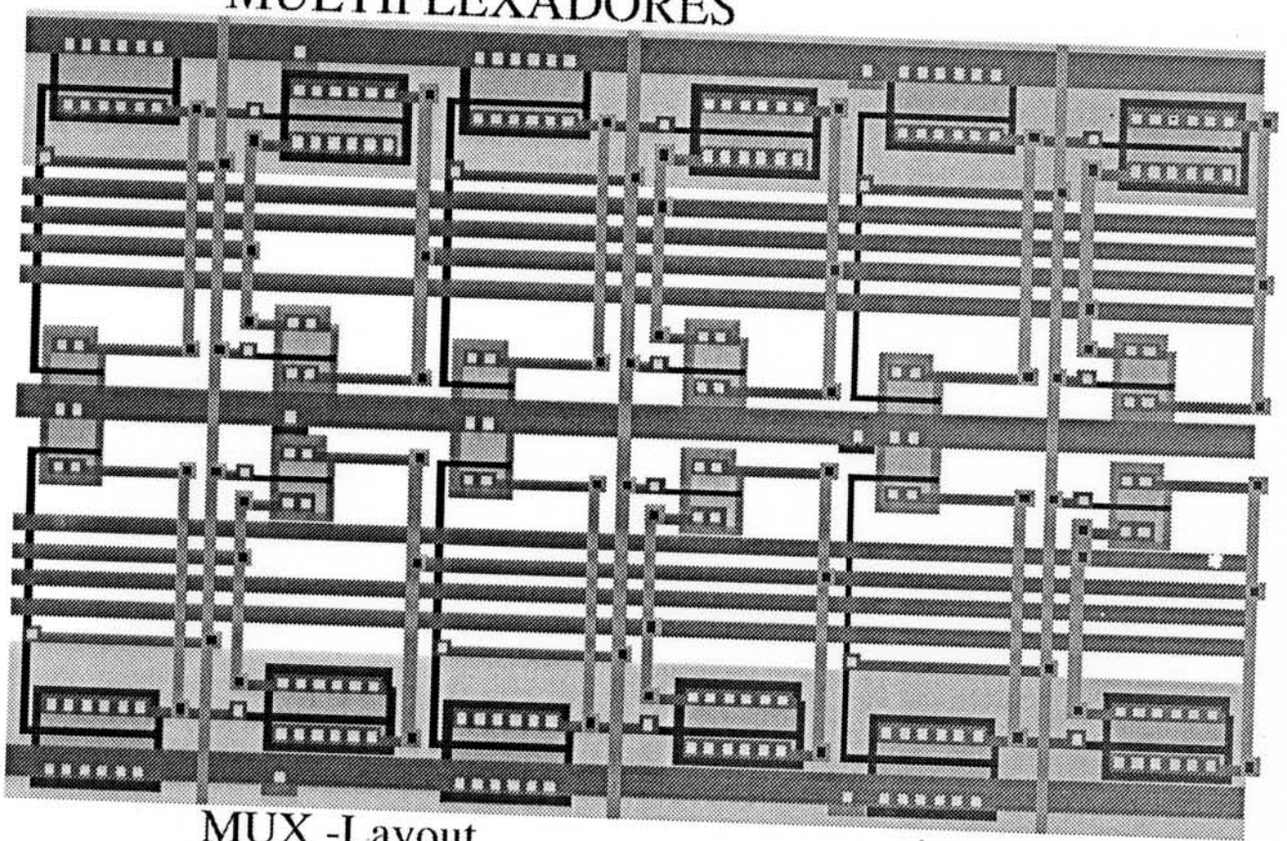
MUX - Simbolico



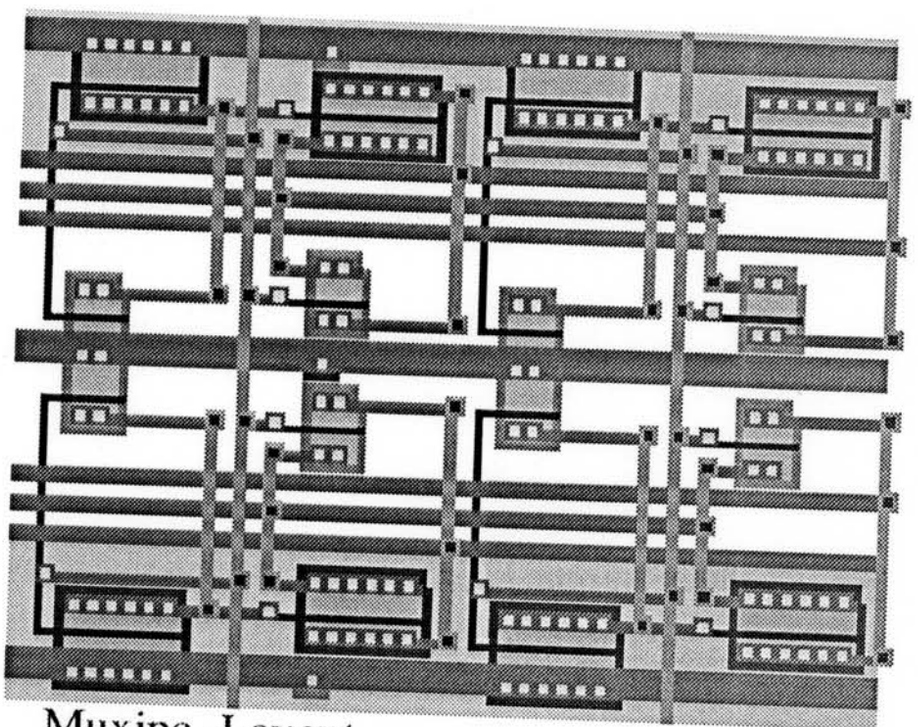
MUXIPC - Simbolico



MULTIPLEXADORES

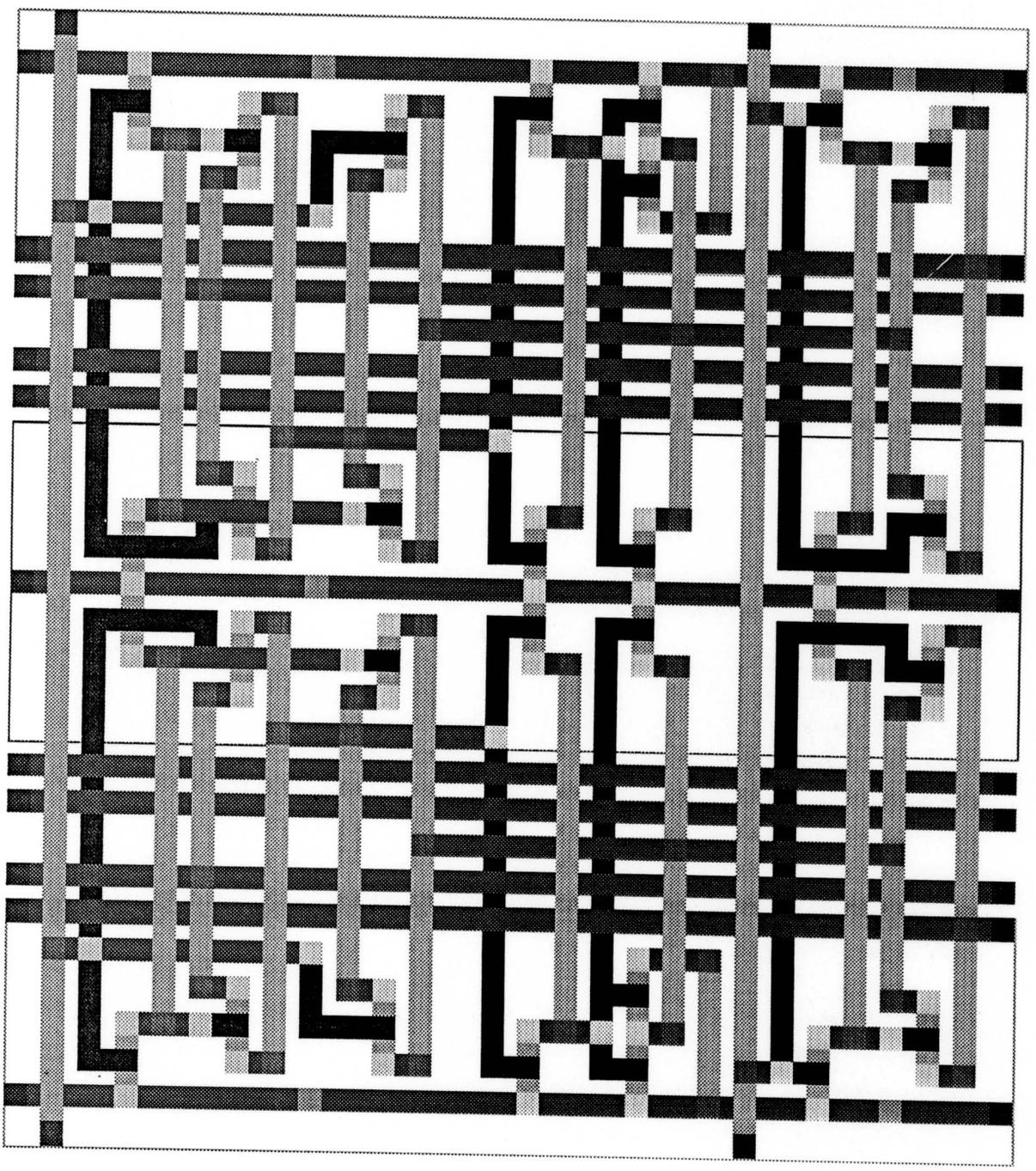


MUX -Layout

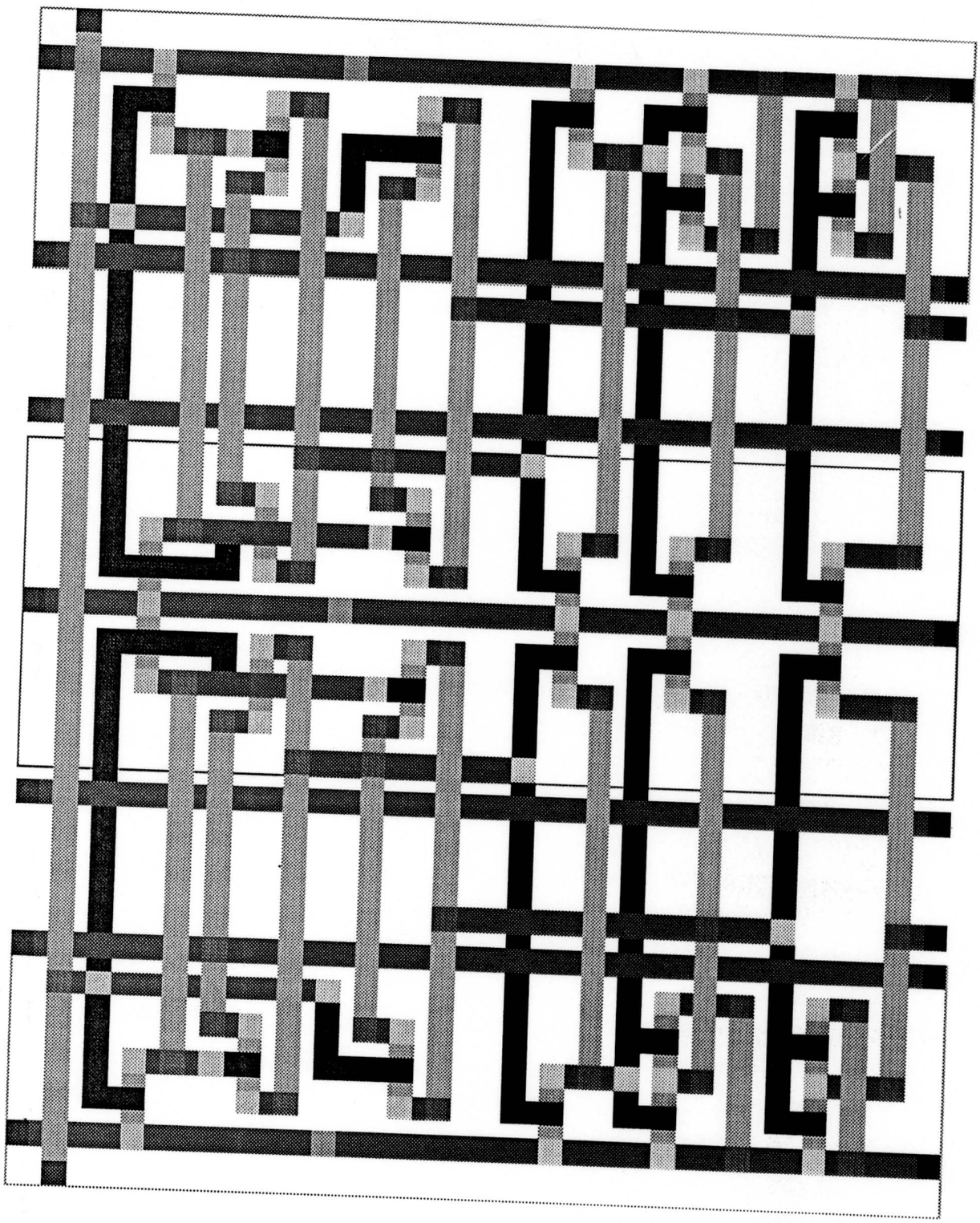


Muxipc -Layout

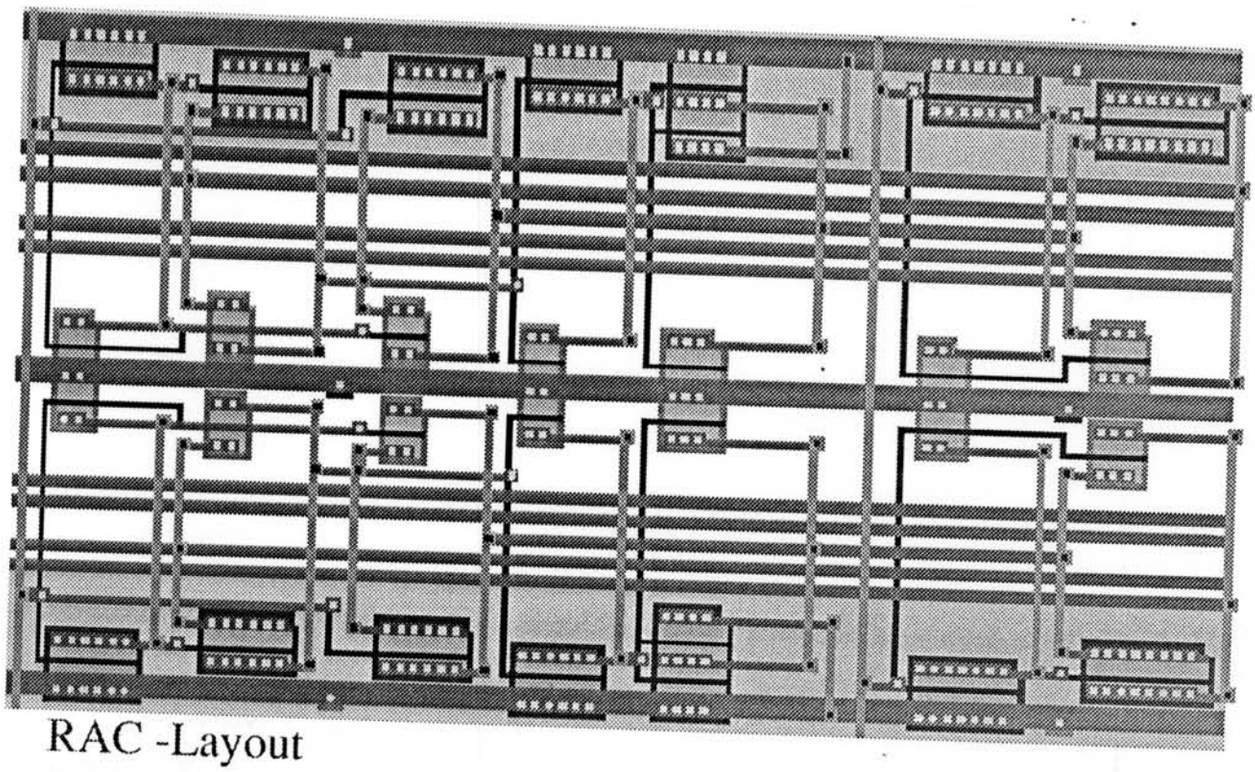
RAC -Simbolico



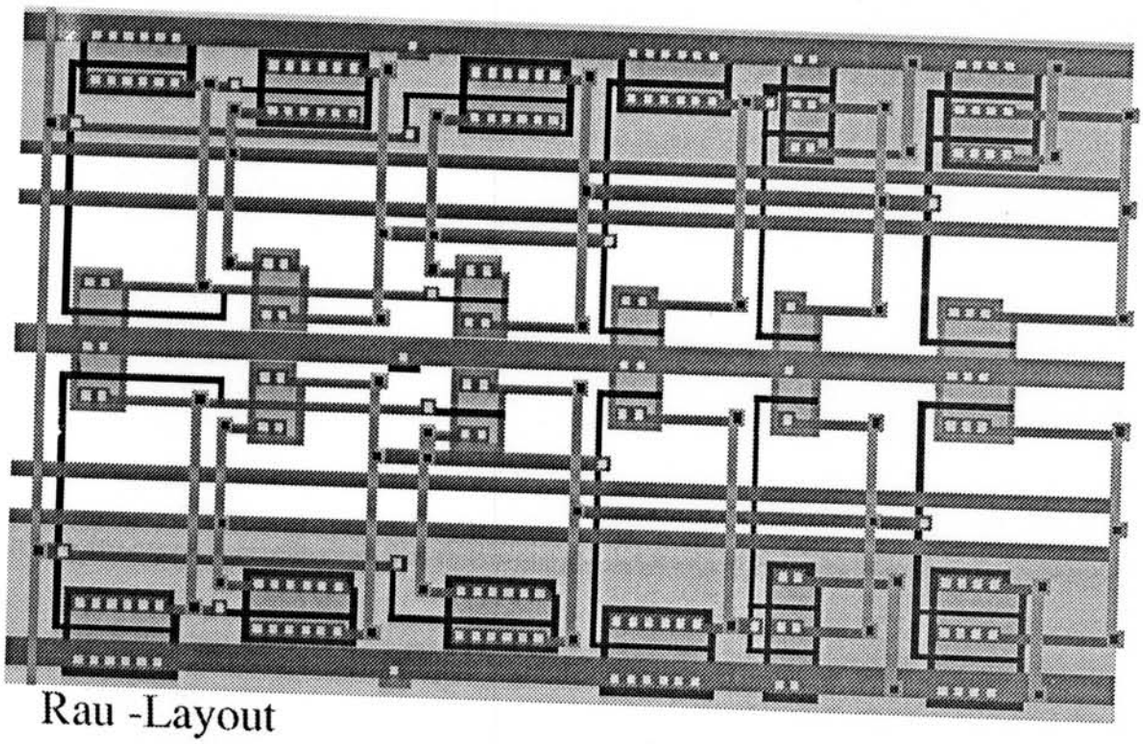
Rau - Simbolico



REGISTRADORES



RAC -Layout



Rau -Layout

```
*                          DAS 9100                          * PAGE 1

TIMING DIAGRAM          MAG:    1                  CURSOR SEQ:   14
                                                   DELTA TIME:  OFF
SRCH =  XXXX XXXX XXXX XXXX
            T
POD CH  NAME   C
 2D 7                                                                  1
 2D 6                                                                  0
 2D 5                                                                  0
 2D 4                                                                  0

 2D 3                                                                  0
 2D 2                                                                  0
 2D 1                                                                  0
 2D 0                                                                  1

 2C 7                                                                  0
 2C 6                                                                  1
 2C 5                                                                  0
 2C 4                                                                  1

 2C 3                                                                  0
 2C 2                                                                  0
 2C 1                                                                  0
 2C 0                                                                  0
```

```
*                          DAS 9100                          * PAGE 1

TIMING DIAGRAM          MAG:    1                  CURSOR SEQ:    6
                                                   DELTA TIME:  OFF
SRCH =  XXXX XXXX XXXX XXXX
             CT
POD CH  NAME
 2D 7                                                                  0
 2D 6                                                                  0
 2D 5                                                                  0
 2D 4                                                                  0

 2D 3                                                                  0
 2D 2                                                                  0
 2D 1                                                                  0
 2D 0                                                                  0

 2C 7                                                                  0
 2C 6                                                                  0
 2C 5                                                                  0
 2C 4                                                                  0

 2C 3                                                                  0
 2C 2                                                                  0
 2C 1                                                                  1
 2C 0
```

ANEXO A-5

HDC do test-chip

```
/* pinos.cfg: */

PADS
RAC_C
RAC_D
RBC_C
RBC_D
RAS_C
RAS_D
RBS_C
RBS_D
RIPCI
PCOUT
M1_A
M1_B
M1_PC
M2_A
M2_B
RAU_C
RBU_C
CIN
ART
XOR
AND
OR
RULA
M3_A
M3_B
```

```
/* ipcaa.dat: */

000 fff 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 fff 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 fff 000 000 000 000 000 000 000 fff fff 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 fff 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 fff 000 000 fff 000 000 000 000 000 000 000 000 000 000 000 000
fff 000 000 000 000 fff 000 000 000 000 000 000 000 000 fff 000 000 000 000 000 000 000 000 000 000 000
fff 000 000 000 000 000 fff 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

```
/* risco.ent: */

PADS
RAC_C
RAC_D
RBC_C
RBC_D
RAS_C
RAS_D
RBS_C
RBS_D
RIPCI
PCOUT
M1_A
M1_B
M1_PC
M2_A
M2_B
RAU_C
RBU_C
CIN
ART
XOR
AND
OR
RULA
M3_A
M3_B
```

```
/* risco.h: */

#ifndef RISCO_H

#define RISCO_H

#define SUM2(Ci,A0,A1,S0,S1,Co) { Co=Ci & A0 & A1;S0=(~Ci & A0)|(Ci & ~A0);\
                                  S1=(~Ci & A1)|(A1 & ~A0)|(Ci & ~A1 & A0);\
                                }

#define ULA1(A,B,CIN,ART,LXOR,LAND,LOR,S,COUT)                              \
                    {                                                       \
                       if (ART)  S=(~CIN ^ ~(A ^ B));                       \
                       if (LXOR) S=(A ^ B);                                 \
                       if (LAND) S=~(A | B);                                \
                       if (LOR)  S=~(A & B);                                \
                       if (~CIN)  COUT=~(A | B); else COUT=~(A & B);        \
                    }

long unsigned bintohex( char *);

long unsigned bintohex(S)
char S[];
{
 long unsigned x=0;
 int i;
 for(i=0; i<=BITs-1; i++) if(S[i])  x = x | (1<<i);
 return x;
}

#endif
```

```
/* ipc.hdc: */

/*-------------------------------------------------------------------------*/
/*            PROJETO RISCO      -      INCREMENTADOR  DO  PC             */
/*              Declaracao dos Nodos do Sistema Digital                    */
/*-------------------------------------------------------------------------*/

#define BITs 16

typedef struct
{
 char S[BITs],A[BITs],C[BITs+1],X[BITs];
} circuito;

#include "globais.h"
#include "risco.h"

/*-------------------------------------------------------------------------*/
/*    Declaracao dos nodos que serao alterados no decorrer da simulacao    */
/*                    NODO(char *, <long, unsigned, int, char>, int);      */
/*            NODO("Idetificacao", variavel_a_ser_alterada, radical);      */
/* radical: 16 para representacao hexa ou 10 para representacao em decimal */
/*-------------------------------------------------------------------------*/
void Altera_nodos()
{
 NODO ("Ci",  sa->C[0], 16);
}

/*-------------------------------------------------------------------------*/
/*                          Parametros de Simulacao                        */
/*-------------------------------------------------------------------------*/
void Parametros()
{
 n_fases=1;
 ATRASO=0;
}

/*-------------------------------------------------------------------------*/
/*                          Iniciacao das variaveis                        */
/*-------------------------------------------------------------------------*/
void Iniciacao()
{
 int k;
 for(k=0 ;k<=BITs; k++) sa->A[k]=false;
 sa->C[0]  = false;
}

/*-------------------------------------------------------------------------*/
/*                       Descricao do Sistema Digital                      */
/*-------------------------------------------------------------------------*/
void Descricao()
{
 int i;
 for (i=0 ;i<=BITs-2 ;i=i+2)
     {
      SUM2(sa->C[i],sa->A[i],sa->A[i+1],sn->S[i],sn->S[i+1],sn->C[i+2]);
     }
 for (i=0 ;i<=BITs-1 ;i++ )
     {
      if( f[1]) sn->X[i]=sa->S[i];
      if(~f[1]) sn->A[i]=sa->X[i];
     }
}
```

```
/*-------------------------------------------------------------------------*/
/*                    Exibicao grafica das variaveis                       */
/*                      JANELA(int, long, char *);                         */
/*                    VALOR(int, long, char *, int);                       */
/*     JANELA(numero_da_janela, valor_a_ser_exibido, "Idetificacao");      */
/*  VALOR(numero_do_valor, valor_a_ser_exibido, "Idetificacao", radical);  */
/*-------------------------------------------------------------------------*/

void Exibicao()
{
 int k;
 char tx[5];
 JANELA (0 ,  sa->C[0]    ,  "Ci"  );
 for (k=0 ;k<=BITs-1 ;k++ )
            {
             sprintf(tx,"s%d",k);
             JANELA (k+1 ,  sa->S[k]   ,  tx  );
            }
 JANELA (k+1 ,  sa->C[BITs]    ,  "Co"    );
 JANELA (k+2 ,  f[1]           ,  "fil"   );
 VALOR  ( 0  , bintohex(sa->S), "S" , 10 );
}
```

```
/* poi6.hdc: */

/* ************************************************************************** */
/*                                                                            */
/*              PROJETO RISCO      -     PARTE OPERATIVA                      */
/*                                                                            */
/* ************************************************************************** */

/*----------------------------------------------------------------------------*/
/*              Declaracao dos Nodos do Sistema Digital                       */
/*----------------------------------------------------------------------------*/
#define BITs 16

typedef struct
{
 /*--------------------- Barramentos do Sistema -------------------------*/
 char S[BITs]   , /* barramento externo ( conectado aos PADs )    */
      A[BITs]   , /* barramento A interno                         */
      B[BITs]   , /* barramento B interno                         */
      SPC[BITs] , /* barramento de saida do PC                    */
      EPC[BITs] , /* barramento de entrada do PC                  */
      SIPC[BITs], /* barramento de saida do incrementador do PC   */
      EIPC[BITs]; /* barramento de entrada do incrementador do PC */
 char EAU[BITs] , /* barramento de entrada A da ULA               */
      EBU[BITs] , /* barramento de entrada B da ULA               */
      SULA[BITs], /* barramento de saida da ULA                   */
      BU[BITs]  ; /* barramento de saida do registrador da ULA    */

 /*-------------------- Sinais de Controle do Sistema ---------------------*/
 char RAC_c,      /* carga no registrador    A <- S               */
      RAC_d,      /* descarga no registrador A <- S               */
      RBC_c,      /* carga no registrador    B <- S               */
      RBC_d,      /* descarga no registrador B <- S               */
      RAS_c,      /* carga no registrador    S <- A               */
      RAS_d,      /* descarga no registrador S <- A               */
      RBS_c,      /* carga no registrador    S <- B               */
      RBS_d,      /* descarga no registrador S <- B               */
      PADS;       /* controle dos pads bidirecionais              */
 char Ripci,      /* carga no registrador de entrada do IPC       */
      PCout,      /* carga no registrador PC                      */
      M1_a ,      /* coloca barramento A na entrada do PC         */
      M1_b ,      /* coloca barramento B na entrada do PC         */
      M1_pc,      /* coloca saida do IPC na entrada do PC         */
      M2_a ,      /* coloca saida do PC no barramento A           */
      M2_b ;      /* coloca saida do PC no barramento B           */
 char RAU_c,      /* carga no barramento de entrada A da ULA      */
      RBU_c,      /* carga no barramento de entrada B da ULA      */
      Cin  ,      /* sinal de carry de entrada da ULA             */
      ART  ,      /* habilita opera~ao ARIT da ULA                */
      XOR  ,      /* habilita opera~ao XOR  da ULA                */
      AND  ,      /* habilita opera~ao AND  da ULA                */
      OR   ,      /* habilita opera~ao  OR  da ULA                */
      OVF  ,      /* sinal de overflow da ULA ( Carry out )       */
      ZERO ,      /* sinal de zero da ULA                         */
      Rula ,      /* carga no registrador de saida da ULA         */
      M3_a ,      /* coloca resultado da ULA no barramento A      */
      M3_b ;      /* coloca resultado da ULA no barramento B      */
} circuito;

#include "globais.h"
#include "risco.h"

void COMUNIC(void);
void IPC(void);
void ULA(void);
```

```
/*-------------------------------------------------------------------------*/
/*    Declaracao dos nodos que serao alterados no decorrer da simulacao    */
/*-------------------------------------------------------------------------*/

void Altera_nodos()
{
 char tx[5];
 int k;
 NODO (" PADS "  ,sa->PADS    , 16);
 NODO (" RAC_c"  ,sa->RAC_c   , 16);
 NODO (" RAC_d"  ,sa->RAC_d   , 16);
 NODO (" RBC_c"  ,sa->RBC_c   , 16);
 NODO (" RBC_d"  ,sa->RBC_d   , 16);
 NODO (" RAS_c"  ,sa->RAS_c   , 16);
 NODO (" RAS_d"  ,sa->RAS_d   , 16);
 NODO (" RBS_c"  ,sa->RBS_c   , 16);
 NODO (" RBS_d"  ,sa->RBS_d   , 16);
 NODO (" Ripci"  ,sa->Ripci   , 16);
 NODO (" PCout"  ,sa->PCout   , 16);
 NODO ("  M1_a"  ,sa->M1_a    , 16);
 NODO ("  M1_b"  ,sa->M1_b    , 16);
 NODO ("  M1_c"  ,sa->M1_pc   , 16);
 NODO ("  M2_a"  ,sa->M2_a    , 16);
 NODO ("  M2_b"  ,sa->M2_b    , 16);
 NODO (" RAU_c"  ,sa->RAU_c   , 16);
 NODO (" RBU_c"  ,sa->RBU_c   , 16);
 NODO ("   Cin"  ,sa->Cin     , 16);
 NODO ("   ART"  ,sa->ART     , 16);
 NODO ("   XOR"  ,sa->XOR     , 16);
 NODO ("   AND"  ,sa->AND     , 16);
 NODO ("    OR"  ,sa->OR      , 16);
 NODO ("  Rula"  ,sa->Rula    , 16);
 NODO ("  M3_a"  ,sa->M3_a    , 16);
 NODO ("  M3_b"  ,sa->M3_b    , 16);
 for ( k=0 ; k<=BITs-1 ; k++ )
         {
          sprintf(tx, "S%d" , k);
          NODO ( tx ,  sa->S[k], 16);
         }
}

void Altera_nodos_batch()
{
 NODO_B (sa->PADS    , 16);
 NODO_B (sa->RAC_c   , 16);
 NODO_B (sa->RAC_d   , 16);
 NODO_B (sa->RBC_c   , 16);
 NODO_B (sa->RBC_d   , 16);
 NODO_B (sa->RAS_c   , 16);
 NODO_B (sa->RAS_d   , 16);
 NODO_B (sa->RBS_c   , 16);
 NODO_B (sa->RBS_d   , 16);
 NODO_B (sa->Ripci   , 16);
 NODO_B (sa->PCout   , 16);
 NODO_B (sa->M1_a    , 16);
 NODO_B (sa->M1_b    , 16);
 NODO_B (sa->M1_pc   , 16);
 NODO_B (sa->M2_a    , 16);
 NODO_B (sa->M2_b    , 16);
 NODO_B (sa->RAU_c   , 16);
 NODO_B (sa->RBU_c   , 16);
 NODO_B (sa->Cin     , 16);
 NODO_B (sa->ART     , 16);
 NODO_B (sa->XOR     , 16);
 NODO_B (sa->AND     , 16);
 NODO_B (sa->OR      , 16);
 NODO_B (sa->Rula    , 16);
 NODO_B (sa->M3_a    , 16);
 NODO_B (sa->M3_b    , 16);
}
```

```
/*-------------------------------------------------------------------------*/
/*                          Parametros de Simulacao                        */
/*-------------------------------------------------------------------------*/
void Parametros()
{
 n_fases=1;
 ATRASO=0;
}

/*-------------------------------------------------------------------------*/
/*                         Iniciacao das variaveis                         */
/*-------------------------------------------------------------------------*/
void Iniciacao()
{
 int k;
 sa->Cin = false;
 for (k=0 ;k<=BITs ;k++)
       {
        sa->S[k] = 0 ;
       }
}

/*-------------------------------------------------------------------------*/
/*                       Descricao do Sistema Digital                      */
/*-------------------------------------------------------------------------*/
void COMUNIC()
{
 static char TMP1[BITs],TMP2[BITs];
 int k,j;

 /* -------------- inicializa~ao dos barramentos do sistema --------------*/
 for (j=0 ;j<BITs ;j++)
       {
        sn->A[j]    = 0 ;
        sn->B[j]    = 0 ;
        sn->SIPC[j] = 0 ;
        sn->EPC[j]  = 0 ;
        sn->SULA[j] = 0 ;
       }

 /*------ comunicaçao entre barramento externo e barramentos internos -----*/
 if ((sa->RAC_c | sa->RBC_c)&~sa->PADS)
                for (k=0 ;k<=BITs-1 ;k++ )   TMP1[k]  = sa->S[k] ;
 if (sa->RAS_c) for (k=0 ;k<=BITs-1 ;k++ )   TMP2[k]  = sa->A[k] ;
 if (sa->RBS_c) for (k=0 ;k<=BITs-1 ;k++ )   TMP2[k]  = sa->B[k] ;
 if (sa->RAC_d) for (k=0 ;k<=BITs-1 ;k++ )  sn->A[k]  = TMP1[k]  ;
 if (sa->RBC_d) for (k=0 ;k<=BITs-1 ;k++ )  sn->B[k]  = TMP1[k]  ;
 if ((sa->RAS_d | sa->RBS_d)& sa->PADS)
                for (k=0 ;k<=BITs-1 ;k++ )   sn->S[k] = TMP2[k]  ;
}

void IPC()
{
 static char CP[BITs+1];
 int k,j;
 if (sa->Ripci) for (k=0 ;k<=BITs-1 ;k++ ) sn->EIPC[k] = sa->SPC[k];
 CP[0]=true;
 for (j=0 ;j<=BITs-2 ;j=j+2 )
           {
            SUM2(CP[j],sa->EIPC[j],sa->EIPC[j+1],sn->SIPC[j],
                 sn->SIPC[j+1],CP[j+2]);
           }
 if (sa->M1_a ) for (k=0 ;k<=BITs-1 ;k++ ) sn->EPC[k] = sa->A[k]   ;
 if (sa->M1_b ) for (k=0 ;k<=BITs-1 ;k++ ) sn->EPC[k] = sa->B[k]   ;
 if (sa->M1_pc) for (k=0 ;k<=BITs-1 ;k++ ) sn->EPC[k] = sa->SIPC[k];
 if (sa->PCout) for (k=0 ;k<=BITs-1 ;k++ ) sn->SPC[k] = sa->EPC[k] ;
 if (sa->M2_a ) for (k=0 ;k<=BITs-1 ;k++ ) sn->A[k]   = sa->SPC[k] ;
 if (sa->M2_b ) for (k=0 ;k<=BITs-1 ;k++ ) sn->B[k]   = sa->SPC[k] ;
}
```

```
void ULA()
{
 static char CU[BITs+1],Z[BITs+1];
 int k,j;
 if (sa->RAU_c) for (k=0 ;k<=BITs-1 ;k++ ) sn->EAU[k] =~sa->A[k]   ;
 if (sa->RBU_c) for (k=0 ;k<=BITs-1 ;k++ ) sn->EBU[k] =~sa->B[k]   ;
 CU[0] = sa->Cin;
 for (j=0 ;j<BITs ;j++ )
          {
           ULA1(sa->EAU[j],sa->EBU[j],CU[j],sa->ART,sa->XOR,
                sa->AND,sa->OR,sn->SULA[j],CU[j+1]);
          }
 Z[0] = false;
 for (j=1 ;j<BITs ;j=j+2 )
          {
           Z[j]   = NOR2 (sa->SULA[j-1], Z[j-1]);
           Z[j+1] = NAND2(~sa->SULA[j] ,  Z[j] );
          }
 sn->ZERO = Z[BITs] ;
 sn->OVF  = CU[BITs];
 if (sa->Rula ) for (k=0 ;k<=BITs-1 ;k++ ) sn->BU[k]  = sa->SULA[k];
 if (sa->M3_a ) for (k=0 ;k<=BITs-1 ;k++ ) sn->A[k]   = sa->BU[k]  ;
 if (sa->M3_b ) for (k=0 ;k<=BITs-1 ;k++ ) sn->B[k]   = sa->BU[k]  ;
}

void Descricao()
{
 COMUNIC();
 IPC();
 ULA();
}

/*-------------------------------------------------------------------------*/
/*                    Exibicao grafica das variaveis                       */
/*-------------------------------------------------------------------------*/
void Exibicao()
{
   int k;
   char tx[5];
   for(k=0 ;k<=BITs-1 ;k++)
        {
         sprintf( tx, "S%d" , k);
         JANELA (k ,  sa->S[k]   ,    tx  );
        }
   JANELA (BITs   ,  sa->OVF    ,  "Of"  );
   JANELA (BITs+1 ,  sa->ZERO   ,  "Zr"  );
   VALOR  ( 0 , bintohex(sa->A)     , "A   ", 10);
   VALOR  ( 1 , bintohex(sa->B)     , "B   ", 10);
   VALOR  ( 2 , bintohex(sa->S)     , "S   ", 10);
}
```

```
/* risco.hdc: */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                PROJETO RISCO      -      PARTE OPERATIVA               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/*-------------------------------------------------------------------------*/
/*              Declaracao dos Nodos do Sistema Digital                    */
/*-------------------------------------------------------------------------*/
#define BITs 16

typedef struct
{
 /*---------------------- Barramentos do Sistema -------------------------*/
 char S[BITs],A[BITs],B[BITs],SPC[BITs],EPC[BITs],SIPC[BITs],EIPC[BITs];
 char EAU[BITs],EBU[BITs],SULA[BITs],BU[BITs];

 /*-------------------- Sinais de Controle do Sistema --------------------*/
 /*  Barramento de saida    X   Barramentos internos   */
 char RAC_c,RAC_d,RBC_c,RBC_d,RAS_c,RAS_d,RBS_c,RBS_d,PADS;
 /*  Increm. do PC          X   Barramentos internos   */
 char Ripci,PCout,M1_a,M1_b,M1_pc,M2_a,M2_b;
 /*  Unid. Log. e Arit.     X   Barramentos internos   */
 char RAU_c,RBU_c,Cin,ART,XOR,AND,OR,OVF,ZERO,Rula,M3_a,M3_b;
} circuito;

#include "globais.h"
#include "risco.h"

/*-------------------------------------------------------------------------*/
/*    Declaracao dos nodos que serao alterados no decorrer da simulacao    */
/*              NODO(char *, <long, unsigned, int, char>, int);            */
/*         NODO("Idetificacao", variavel_a_ser_alterada, radical);         */
/* radical: 16 para representacao hexa ou 10 para representacao em decimal */
/*-------------------------------------------------------------------------*/
void Altera_nodos()
{
 char tx[5];
 int k;
 NODO (" PADS "  ,sa->PADS    , 16);
 NODO (" RAC_c"  ,sa->RAC_c   , 16);
 NODO (" RAC_d"  ,sa->RAC_d   , 16);
 NODO (" RBC_c"  ,sa->RBC_c   , 16);
 NODO (" RBC_d"  ,sa->RBC_d   , 16);
 NODO (" RAS_c"  ,sa->RAS_c   , 16);
 NODO (" RAS_d"  ,sa->RAS_d   , 16);
 NODO (" RBS_c"  ,sa->RBS_c   , 16);
 NODO (" RBS_d"  ,sa->RBS_d   , 16);
 NODO (" Ripci"  ,sa->Ripci   , 16);
 NODO (" PCout"  ,sa->PCout   , 16);
 NODO ("  M1_a"  ,sa->M1_a    , 16);
 NODO ("  M1_b"  ,sa->M1_b    , 16);
 NODO ("  M1_c"  ,sa->M1_pc   , 16);
 NODO ("  M2_a"  ,sa->M2_a    , 16);
 NODO ("  M2_b"  ,sa->M2_b    , 16);
 NODO (" RAU_c"  ,sa->RAU_c   , 16);
 NODO (" RBU_c"  ,sa->RBU_c   , 16);
 NODO ("   Cin"  ,sa->Cin     , 16);
 NODO ("   ART"  ,sa->ART     , 16);
 NODO ("   XOR"  ,sa->XOR     , 16);
 NODO ("   AND"  ,sa->AND     , 16);
 NODO ("    OR"  ,sa->OR      , 16);
 NODO ("  Rula"  ,sa->Rula    , 16);
 NODO ("  M3_a"  ,sa->M3_a    , 16);
 NODO ("  M3_b"  ,sa->M3_b    , 16);
 for ( k=0 ; k<=BITs-1 ; k++ )
         {
          sprintf(tx, "S%d" , k);
          NODO ( tx ,  sa->S[k], 16);
         }
}
```

```
Altera_nodos_batch()
{
 NODO_B (sa->PADS    , 16);
 NODO_B (sa->RAC_c   , 16);
 NODO_B (sa->RAC_d   , 16);
 NODO_B (sa->RBC_c   , 16);
 NODO_B (sa->RBC_d   , 16);
 NODO_B (sa->RAS_c   , 16);
 NODO_B (sa->RAS_d   , 16);
 NODO_B (sa->RBS_c   , 16);
 NODO_B (sa->RBS_d   , 16);
 NODO_B (sa->Ripci   , 16);
 NODO_B (sa->PCout   , 16);
 NODO_B (sa->M1_a    , 16);
 NODO_B (sa->M1_b    , 16);
 NODO_B (sa->M1_pc   , 16);
 NODO_B (sa->M2_a    , 16);
 NODO_B (sa->M2_b    , 16);
 NODO_B (sa->RAU_c   , 16);
 NODO_B (sa->RBU_c   , 16);
 NODO_B (sa->Cin     , 16);
 NODO_B (sa->ART     , 16);
 NODO_B (sa->XOR     , 16);
 NODO_B (sa->AND     , 16);
 NODO_B (sa->OR      , 16);
 NODO_B (sa->Rula    , 16);
 NODO_B (sa->M3_a    , 16);
 NODO_B (sa->M3_b    , 16);
}

/*-------------------------------------------------------------------------*/
/*                          Parametros de Simulacao                        */
/*-------------------------------------------------------------------------*/
void Parametros()
{
 n_fases=1;
 ATRASO=0;
}

/*-------------------------------------------------------------------------*/
/*                         Iniciacao das variaveis                         */
/*-------------------------------------------------------------------------*/
void Iniciacao()
{
 int k;
 Arquivo_batch("SIMUL.DAT");
 sa->Cin = false;
 for (k=0 ;k<=BITs ;k++)
        {
         sa->S[k] = 0 ;
        }
}

/*-------------------------------------------------------------------------*/
/*                       Descricao do Sistema Digital                      */
/*-------------------------------------------------------------------------*/
void Descricao()
{
 static char TMP1[BITs],TMP2[BITs],CP[BITs+1],CU[BITs+1],Z[BITs+1];
 int k,j;

 /* -------------- inicializa~ao dos barramentos do sistema --------------*/
 for (j=0 ;j<BITs ;j++)
                {
                 sn->A[j]    = 0 ;
                 sn->B[j]    = 0 ;
                 sn->SIPC[j] = 0 ;
                 sn->EPC[j]  = 0 ;
                 sn->SULA[j] = 0 ;
                }
```

```
/*------ comunicacao entre barramento externo e barramentos internos -----*/
if ((sa->RAC_c | sa->RBC_c)&~sa->PADS)
                for (k=0 ;k<=BITs-1 ;k++ ) TMP1[k]  = sa->S[k]   ;
if (sa->RAS_c)
                for (k=0 ;k<=BITs-1 ;k++ )   TMP2[k]  = sa->A[k] ;
if (sa->RBS_c)
                for (k=0 ;k<=BITs-1 ;k++ )   TMP2[k]  = sa->B[k] ;
if (sa->RAC_d)
                for (k=0 ;k<=BITs-1 ;k++ )  sn->A[k]  = TMP1[k]  ;
if (sa->RBC_d)
                for (k=0 ;k<=BITs-1 ;k++ )  sn->B[k]  = TMP1[k]  ;
if ((sa->RAS_d | sa->RBS_d)& sa->PADS)
                for (k=0 ;k<=BITs-1 ;k++ ) sn->S[k] = TMP2[k]    ;

/*-------- operacao de incremento do contador de programas  ( PC ) -------*/
if (sa->Ripci) for (k=0 ;k<=BITs-1 ;k++ ) sn->EIPC[k] = sa->SPC[k];
CP[0]=true;
for (j=0 ;j<=BITs-2 ;j=j+2 )
          {
           SUM2(CP[j],sa->EIPC[j],sa->EIPC[j+1],sn->SIPC[j],
                sn->SIPC[j+1],CP[j+2]);
          }
if (sa->M1_a )
                for (k=0 ;k<=BITs-1 ;k++ ) sn->EPC[k] = sa->A[k]   ;
if (sa->M1_b )
                for (k=0 ;k<=BITs-1 ;k++ ) sn->EPC[k] = sa->B[k]   ;
if (sa->M1_pc)
                for (k=0 ;k<=BITs-1 ;k++ ) sn->EPC[k] = sa->SIPC[k];
if (sa->PCout)
                for (k=0 ;k<=BITs-1 ;k++ ) sn->SPC[k] = sa->EPC[k] ;
if (sa->M2_a )
                for (k=0 ;k<=BITs-1 ;k++ ) sn->A[k]   = sa->SPC[k] ;
if (sa->M2_b )
                for (k=0 ;k<=BITs-1 ;k++ ) sn->B[k]   = sa->SPC[k] ;

/*--------------- operacao da unidade logica e aritmetica ---------------*/
if (sa->RAU_c)
                for (k=0 ;k<=BITs-1 ;k++ ) sn->EAU[k] =~sa->A[k]   ;
if (sa->RBU_c)
                for (k=0 ;k<=BITs-1 ;k++ ) sn->EBU[k] =~sa->B[k]   ;
CU[0] = sa->Cin;
for (j=0 ;j<BITs ;j++ )
          {
           ULA1(sa->EAU[j],sa->EBU[j],CU[j],sa->ART,sa->XOR,
                sa->AND,sa->OR,sn->SULA[j],CU[j+1]);
          }
Z[0] = false;
for (j=1 ;j<BITs ;j=j+2 )
          {
           Z[j]   = NOR2 (sa->SULA[j-1], Z[j-1]);
           Z[j+1] = NAND2(~sa->SULA[j] ,  Z[j] );
          }
sn->ZERO = Z[BITs] ;
sn->OVF  = CU[BITs];
if (sa->Rula )
                for (k=0 ;k<=BITs-1 ;k++ ) sn->BU[k]  = sa->SULA[k];
if (sa->M3_a )
                for (k=0 ;k<=BITs-1 ;k++ ) sn->A[k]   = sa->BU[k]  ;
if (sa->M3_b )
                for (k=0 ;k<=BITs-1 ;k++ ) sn->B[k]   = sa->BU[k]  ;
}
```

```
/*-------------------------------------------------------------------------*/
/*                    Exibicao grafica das variaveis                       */
/*                      JANELA(int, long, char *);                         */
/*                     VALOR(int, long, char *, int);                      */
/*     JANELA(numero_da_janela, valor_a_ser_exibido, "Idetificacao");      */
/*  VALOR(numero_do_valor, valor_a_ser_exibido, "Idetificacao", radical);  */
/*-------------------------------------------------------------------------*/
void Exibicao()
{
   int k;
   char tx[5];
   for(k=0 ;k<=BITs-1 ;k++)
        {
         sprintf( tx, "S%d" , k);
         JANELA (k ,  sa->S[k]   ,    tx  );
        }
   JANELA (BITs   ,  sa->OVF    ,  "Of"  );
   JANELA (BITs+1 ,  sa->ZERO   ,  "Zr"  );
   VALOR  ( 0 , bintohex(sa->A) , "A", 10);
   VALOR  ( 1 , bintohex(sa->B) , "B", 10);
   VALOR  ( 2 , bintohex(sa->S) , "S", 10);
}
```

```
/* ula.hdc: */

/*-------------------------------------------------------------------------*/
/*          PROJETO RISCO     -     UNIDADE LOGICA E ARITMETICA            */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*               Declaracao dos Nodos do Sistema Digital                   */
/*-------------------------------------------------------------------------*/
#define BITs 4

typedef struct
{
 char S[BITs],C[BITs+1],UART,UXOR,UAND,UOR;
 char A[BITs],B[BITs],ZERO,OVERFLOW;
} circuito;

#include "globais.h"
#include "risco.h"

/*-------------------------------------------------------------------------*/
/*    Declaracao dos nodos que serao alterados no decorrer da simulacao    */
/*               NODO(char *, <long, unsigned, int, char>, int);           */
/*            NODO("Idetificacao", variavel_a_ser_alterada, radical        */
/* radical: 16 para representacao hexa ou 10 para representacao em decimal */
/*-------------------------------------------------------------------------*/
void Altera_nodos()
{
 char tx[5];
 int k;
 NODO ("Ci" ,  sa->C[0], 16);
 NODO ("ART",  sa->UART, 16);
 NODO ("XOR",  sa->UXOR, 16);
 NODO ("AND",  sa->UAND, 16);
 NODO ("OR ",  sa->UOR , 16);
 for(k=0 ;k<=BITs-1 ;k++ )
          {
           sprintf(tx, "a%d" , k);
           NODO ( tx ,  sa->A[k], 16);
          }
 for(k=0 ;k<=BITs-1 ;k++ )
          {
           sprintf(tx, "b%d" , k);
           NODO ( tx ,  sa->B[k], 16);
          }
}

/*-------------------------------------------------------------------------*/
/*                          Parametros de Simulacao                        */
/*-------------------------------------------------------------------------*/
void Parametros()
{
 n_fases=1;
 ATRASO=0;
}

/*-------------------------------------------------------------------------*/
/*                         Iniciacao das variaveis                         */
/*-------------------------------------------------------------------------*/
void Iniciacao()
{
 sa->C[0] = false;
}
```

```
/*-------------------------------------------------------------------------*/
/*                        Descricao do Sistema Digital                     */
/*-------------------------------------------------------------------------*/
void Descricao()
{
 char Z[BITs+1];
 int i;
 for (i=0 ;i<=BITs-1 ;i++)
        {
         ULA1(sa->A[i],sa->B[i],sa->C[i],sa->UART,sa->UXOR,sa->UAND
             ,sa->UOR,sn->S[i],sn->C[i+1]);
        }
 Z[0]=0;
 for (i=1 ;i<BITs ;i=i+2)
        {
         Z[i]   = NOR2 (sa->S[i-1],Z[i-1]);
         Z[i+1] = NAND2(~sa->S[i]  ,Z[i] );
        }
 sn->ZERO=Z[BITs];
}

/*-------------------------------------------------------------------------*/
/*                    Exibicao grafica das variaveis                       */
/*                      JANELA(int, long, char *);                         */
/*                     VALOR(int, long, char *, int);                      */
/*     JANELA(numero_da_janela, valor_a_ser_exibido, "Idetificacao");      */
/*  VALOR(numero_do_valor, valor_a_ser_exibido, "Idetificacao", radical);  */
/*-------------------------------------------------------------------------*/
void Exibicao()
{
 int k;
 char tx[5];
 JANELA (   0    ,  sa->C[0]   ,  "Ci"  );
 for(k=0 ;k<=BITs-1 ;k++)
        {
         sprintf( tx, "s%d" , k);
         JANELA (k+1 ,  sa->S[k]   ,  tx );
        }
 JANELA (BITs+1 ,  sa->C[BITs],  "Of"  );
 JANELA (BITs+2 ,  sa->ZERO   ,  "Zr"  );
 VALOR  ( 2 , bintohex(sa->S) , "S", 10);
 VALOR  ( 0 , bintohex(sa->A) , "A", 10);
 VALOR  ( 1 , bintohex(sa->B) , "B", 10);
}
```

```
/* risco.inp: */

RAC_C
/
RAC_D
PCOUT
M1_A
RAU_C
/
RBC_C
RIPCIN
/
RBC_D
PCOUT
M1_PC
RBU_C
/
RAS_C
RIPCIN
ART
CIN
M2_A
/
PADS
RAS_D
PCOUT
M1_PC
ART
RULA
/
RAS_C
RBS_C
RIPCIN
M2_B
ART
M3_A
/
PADS
RBS_D
PCOUT
M1_PC
ART
RULA
/
PADS
RAS_D
RBS_C
M3_B
XOR
/
RAS_C
M2_A
/
PADS
RAS_D
/
PADS
RBS_D
RIPCIN
XOR
RULA
/
RAS_C
PCOUT
M1_PC
AND
M3_A
/
PADS
RAS_D
RBS_C
M2_B
AND
```

```
RULA
/
PADS
RBS_D
/
RIPCIN
RAS_C
M3_A
OR
/
PADS
RAS_D
PCOUT
M1_PC
OR
RULA
/
RBS_C
RAS_C
M2_A
M3_B
RIPCIN
/
PADS
RAS_D
PCOUT
M1_PC
/
PADS
RBS_D
RAS_C
M2_A
/
PADS
RAS_D
/
```

```
/* risco.out: */

00
ff
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
ff
00
00
00
00
00
00
00
ff
ff
00
00
00
00
ff
00
00
00
00
00
00
```

```
/* risco_1.pas: */

program config;
uses CRT;
type
    str12=string[12];
    str06=string[ 6];
var
    pinos:array [1..50] of str06;
    datas:array [1..50] of str06;
    nomei,nomeo,nomep:str12;
    n:integer;

function num_data(nome:str12):integer;
var
   arq:text;
   i:integer;
   d:str06;
begin {num_data}
   assign(arq,nome);
   reset(arq);
   i:=0;
   repeat
      readln(arq,d);
      i:=i+1;
   until eof(arq);
   num_data:=i;
   close(arq);
end;  {num_data}

procedure read_conf(name:str12);
var
   arq:text;
   i:integer;
   data:str06;
begin {read_conf}
   assign(arq,name);
   reset(arq);
   i:=1;
   repeat
      readln(arq,data);
      pinos[i]:=data;
      i:=i+1;
   until eof(arq);
   if (i-1)<>n then halt;
   close(arq);
end;  {read_conf}
```

```
procedure read_data(name1,name2:str12);
var
   arq1,arq2:text;
   i,j:integer;
   data:str06;
begin {read_data}
   i:=1;
   assign(arq1,name1);
   assign(arq2,name2);
   reset(arq1);
   rewrite(arq2);
   for j:=1 to n do datas[j]:='00';
   repeat
      readln(arq1,data);
      for j:=1 to n do
         begin
            if data=pinos[j] then datas[j]:='ff';
         end;
      if data='/' then
         begin
            for j:=1 to n do
               begin
                  writeln(arq2,datas[j]);
                  i:=1;
               end;
            for j:=1 to n do datas[j]:='00';
         end;
   until eof(arq1);
   close(arq1);
   close(arq2);
end;  {read_data}

begin
clrscr;
gotoxy(15,10);
write(' arquivo de entrada =');
readln(nomei);
gotoxy(15,12);
write(' arquivo  de  saida =');
readln(nomeo);
gotoxy(15,14);
write(' arquivo  de  pinos =');
readln(nomep);
n:=num_data(nomep);
clrscr;
gotoxy(25,12);
write('   configurando ...');
read_conf(nomep);
read_data(nomei,nomeo);
delay(2000);
clrscr;
gotoxy(25,12);
write('       gravando ...');
delay(2000);
end.
```

ANEXO A-6

Especificação da linguagem de montagem

ESPECIFICAÇÃO DO MONTADOR PARA O RISCO

CONSIDERAÇÕES PRELIMINARES
v.: 1.2

```
<linha de instrução> ::=
      <rótulo> <campo de instrução> <comentário> <return>
```

```
<rótulo> ::=
      <vazio> ¦
      <esp ou vaz> <nome do rótulo> :

<nome do rótulo> ::= <identificador>
```

```
<comentário> ::=
      <vazio> ¦
      <esp ou vaz> ; <texto do comentário>

<texto do comentário> ::=
      <vazio> ¦
      <texto do comentário> <caracter>
```

```
/*
Possibilidades para o campo de instrução:

¦                          \     /                          ¦
¦  mnemônico                \   /      Ra, Rb,   Rc        ¦
¦  mnemônico.aps             \ /       Ra, Rb,   num       ¦
¦  mnemônico.cond            / \       Ra, &RO,  num       ¦
¦  mnemônico.cond.aps       /   \      Ra, &RD,  num       ¦
¦                         /     \     Ra, &RDH, num       ¦
¦                        /       \                         ¦
*/

<campo de instrução> ::=
vazio> ¦
      <esp ou vaz> <instrução>

<instrução> ::=
      <mnem aritlog e mem>                    <aps> <campo> ¦
      <mnem mem e altera fb>                  <aps> <campo restrito> ¦
      <mnem jmp e sr>          <condição> <aps> <campo>

<mnem aritlog e mem> ::=
      ADD ¦ ADDC ¦
      add ¦ addc ¦
      SUB ¦ SUBC ¦ SUBR ¦ SUBRC ¦
      sub ¦ subc ¦ subr ¦ subrc ¦
      AND ¦ OR ¦ XOR ¦
      and ¦ or ¦ xor ¦
      RLL ¦ RLLC ¦ RLA ¦ RLAC ¦
      rll ¦ rllc ¦ rla ¦ rlac ¦
      RRL ¦ RRLC ¦ RRA ¦ RRAC ¦
      rrl ¦ rrlc ¦ rra ¦ rrac ¦
      SLL ¦ SLLC ¦ SLA ¦ SLAC ¦
      sll ¦ sllc ¦ sla ¦ slac ¦
      SRL ¦ SRLC ¦ SRA ¦ SRAC ¦
      srl ¦ srlc ¦ sra ¦ srac ¦
      LD ¦ ST ¦
      ld ¦ st
```

```
<mnem mem e altera fb> ::=
      LDPRI ¦ LDPOI ¦ LDPOD ¦
      ldpri ¦ ldpoi ¦ ldpod ¦
      STPRI ¦ STPOI ¦ STPOD ¦
      stpri ¦ stpoi ¦ stpod

<mnem jmp e sr> ::=
      JMP ¦ SR ¦
      jmp ¦ sr

<condição> ::=
      <vazio> ¦
      .TR ¦ .NS ¦ .CS ¦ .OS ¦ .ZS ¦ .GE ¦ .GT ¦ .EQ ¦
      .tr ¦ .ns ¦ .cs ¦ .os ¦ .zs ¦ .ge ¦ .gt ¦ .eq ¦
      .FL ¦ .NN ¦ .NC ¦ .NO ¦ .NZ ¦ .LT ¦ .LE ¦ .NE ¦
      .fl ¦ .nn ¦ .nc ¦ .no ¦ .nz ¦ .lt ¦ .le ¦ .ne

<aps> ::=
      <vazio> ¦
      .APS ¦
      .aps

----------

<campo> ::=
      <campo restrito> ¦
      <espaço> <reg> <separa> <reg> <separa> <num> <esp ou vaz> ¦
      <espaço> <reg> <separa> <fb>  <separa> <num> <esp ou vaz>

<campo restrito> ::=
      <espaço> <reg> <separa> <reg> <separa> <reg> <esp ou vaz>

<separa> ::=
      <espaço> ¦
      <esp ou vaz> , <esp ou vaz>

<fb> ::=
      &r0  ¦ &R0 ¦
      &rd  ¦ &RD ¦
      &rdh ¦ &RDH

<reg> ::= <identificador de registrador> <número decimal>

<identificador de registrador> ::= r ¦ R

<num> ::=
      <número com base> ¦
      <sinal> <número com base>

<sinal> ::= + ¦ -

<número com base> ::=
      <número hexadecimal> <base hexadecimal> ¦
      <número decimal> <base decimal> ¦
      <número octal> <base octal> ¦
      <número binário> <base binária>

<base hexadecimal> ::= h ¦ H

<base decimal> ::= d ¦ D ¦ <vazio>

<base octal> ::= o ¦ O

<base binária> ::= b ¦ B

----------

<caracter> ::=
      <letra ou algarismo> ¦
      <branco> ¦
      ! ¦ " ¦ # ¦ $ ¦ % ¦ & ¦ ' ¦ ( ¦ ) ¦ * ¦ + ¦ , ¦
      - ¦ . ¦ / ¦ : ¦ ; ¦ < ¦ = ¦ > ¦ ? ¦ @ ¦ [ ¦ \ ¦
      ] ¦ ^ ¦ ` ¦ { ¦ | ¦ } ¦ ~
```

```
<identificador> ::=
      <letra> ¦
      <identificador> <letra ou algarismo>

<letra ou algarismo> ::= <letra> ¦ <algarismo decimal>

<letra> ::=
      A ¦ B ¦ C ¦ D ¦ E ¦ F ¦ G ¦ H ¦ I ¦ J ¦ K ¦ L ¦ M ¦
      N ¦ O ¦ P ¦ Q ¦ R ¦ S ¦ T ¦ U ¦ V ¦ W ¦ X ¦ Y ¦ Z ¦ _ ¦
      a ¦ b ¦ c ¦ d ¦ e ¦ f ¦ g ¦ h ¦ i ¦ j ¦ k ¦ l ¦ m ¦
      n ¦ o ¦ p ¦ q ¦ r ¦ s ¦ t ¦ u ¦ v ¦ w ¦ x ¦ y ¦ z

----------

<número hexadecimal> ::=
      <algarismo hexadecimal> ¦
      <número hexadecimal> <algarismo hexadecimal>

<número decimal> ::=
      <algarismo decimal> ¦
      <número decimal> <algarismo decimal>

<número octal> ::=
      <algarismo octal> ¦
      <número octal> <algarismo octal>

<número binário> ::=
      <algarismo binário> ¦
      <número binário> <algarismo binário>

<algarismo hexadecimal> ::=
      <algarismo decimal> ¦
      A ¦ B ¦ C ¦ D ¦ E ¦ F ¦ a ¦ b ¦ c ¦ d ¦ e ¦ f

<algarismo decimal> ::= <algarismo octal> ¦ 8 ¦ 9

<algarismo octal> ::= <algarismo binário> ¦ 2 ¦ 3 ¦ 4 ¦ 5 ¦ 6 ¦ 7

<algarismo binário> ::= 0 ¦ 1

----------

<esp ou vaz> ::= <espaço> ¦ <vazio>

<espaço> ::= <branco> ¦ <espaço> <branco>

<branco> ::= ascii 32d   /* <--- ??? */

----------

<return> ::= ascii 13d  /* <--- ??? */
```

ANEXO A-7

Códigos de máquina

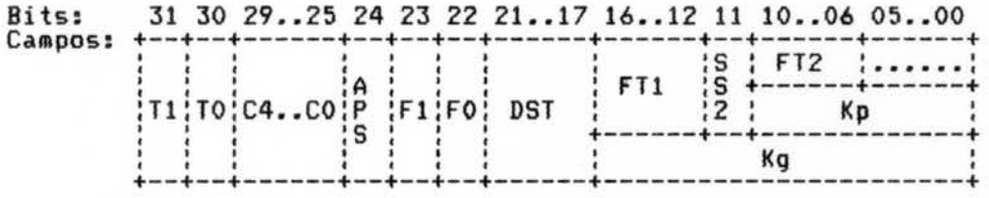
ESPECIFICAÇÃO DOS CÓDIGOS DE MÁQUINA PARA O RISCO

Versão 1.0

1. Possibilidades para o campo de instrução:

Mnemônico	Operandos
mnemônico	Rdst, Rft1, Rft2
mnemônico.aps	Rdst, Rft1, num
mnemônico.cond	Rdst, &R0, num
mnemônico.cond.aps	Rdst, &RD, num
	Rdst, &RDH, num

2. Campos da palavra de instrução de máquina:



3. Bits 31 a 25: Os campos T1, T0, C4, C3, C2, C1 e C0 são determinados pelo mnemônico da instrução de montagem.

3.1 Instruções aritmético-lógicas

Mnem. \ Bits / Campo	31 30 T1 T0	29 28 27 26 25 C4 C3 C2 C1 C0
AND	0 0	0 1 1 1 1
OR	0 0	0 1 1 1 0
XOR	0 0	0 1 1 0 1
ADD	0 0	0 0 0 0 0
ADDC	0 0	0 0 0 1 0
SUB	0 0	0 0 1 0 1
SUBC	0 0	0 0 1 1 1
SUBR	0 0	0 1 0 0 1
SUBRC	0 0	0 1 0 1 1
RLL	0 0	1 0 0 0 0
RLLC	0 0	1 0 0 0 1
RLA	0 0	1 0 0 1 0
RLAC	0 0	1 0 0 1 1
RRL	0 0	1 0 1 0 0
RRLC	0 0	1 0 1 0 1
RRA	0 0	1 0 1 1 0
RRAC	0 0	1 0 1 1 1
SLL	0 0	1 1 0 0 0
SLLC	0 0	1 1 0 0 1
SLA	0 0	1 1 0 1 0
SLAC	0 0	1 1 0 1 1
SRL	0 0	1 1 1 0 0
SRLC	0 0	1 1 1 0 1
SRA	0 0	1 1 1 1 0
SRAC	0 0	1 1 1 1 1

3.2 Instruções de acesso à memória

Mnem. \ Bits / Campo	31 T1	30 T0	29 C4	28 C3	27 C2	26 C1	25 C0
LD	1	0	0	x	0	x	x
LDPRI	1	0	0	x	1	1	1
LDPOI	1	0	0	x	1	0	1
LDPOD	1	0	0	x	1	0	0
ST	1	0	1	x	0	x	x
STPRI	1	0	1	x	1	1	1
STPOI	1	0	1	x	1	0	1
STPOD	1	0	1	x	1	0	0

3.3 Instruções de salto e de sub-rotina

Mnem. \ Bits / Campo	31 T1	30 T0	29 C4	28 C3	27 C2	26 C1	25 C0
JMP	0	1	-	-	-	-	-
SR	1	1	-	-	-	-	-
(sem .cond)	-	-	1	1	1	1	1
.TR	-	-	1	1	1	1	1
.NS	-	-	1	0	0	0	1
.CS	-	-	1	0	0	1	0
.OS	-	-	1	0	1	0	0
.ZS	-	-	1	1	0	0	0
.GE	-	-	1	0	0	1	1
.GT	-	-	1	0	1	1	0
.EQ	-	-	1	1	1	0	0
.FL	-	-	0	0	0	0	0
.NN	-	-	0	0	0	0	1
.NC	-	-	0	0	0	1	0
.NO	-	-	0	0	1	0	0
.NZ	-	-	0	1	0	0	0
.LT	-	-	0	0	0	1	1
.LE	-	-	0	0	1	1	0
.NE	-	-	0	1	1	0	0

==========

4. Bit 24: Se existe a extensão .aps no mnemônico, então o campo APS é setado (=1).

==========

5. Especificação do formato e operandos da palavra de instrução de máquina, através da instrução de montagem.

Instr. de montagem	Formato	F1	F0	SS2	Operação
Mnem Rdst, Rft1, Rft2	DST/FT1/FT2	0	0	0	DST <-- FT1 op FT2
Mnem Rdst, Rft1, num	DST/FT1/Kp	0	0	1	DST <-- FT1 op Kpe
Mnem Rdst, &R0, num	DST/Kg	0	1	x	DST <-- R0 op Kgl
Mnem Rdst, &RDH, num	DST/Kg	1	0	x	DST <-- DST op Kgh
Mnem Rdst, &RD, num	DST/Kg	1	1	x	DST <-- DST op Kgl

Mnem = mnemônico, mnemônico.aps, mnemônico.cond, mnemônico.cond.aps,

==========

```
=====================================================================
6. Bits de 23 a 0: Pelos operandos da instrução de montagem são
determinados os campos F1, F0, SS2, DST, FT1, FT2, Kp e Kg da palavra de
instrução de máquina.

Operandos: Rdst, Rft1, Rft2
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 0  0 <--- Rdst ---> <--- Rft1 --->  0 <- Rft2 -> x x x x x x

Operandos: Rdst, Rft1, num
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 0  0 <--- Rdst ---> <--- Rft1 --->  1 <------- num -------->

Operandos: Rdst, &R0, num
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 0  1 <--- Rdst ---> <---------------- num ------------------>

Operandos: Rdst, &RDH, num
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 1  0 <--- Rdst ---> <---------------- num ------------------>

Operandos: Rdst, &RD, num
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 1  1 <--- Rdst ---> <---------------- num ------------------>

=====================================================================
```

ANEXO A-8

Manual do Montador

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA

MONTADOR RISCO

por
Daniela Cunha Lima

Porto Alegre, setembro de 1993.

MONTADOR PARA O PROCESSADOR RISCO

1. CARACTERÍSTICAS DO PROCESSADOR

O Risco é um microprocessador CMOS de 32 bits com arquitetura tipo **RISC** (Reduced Instruction Set Computer). Instruções, dados e endereços são palavras de 32 bits. O espaço máximo de endereçamento é constituído de 4 Giga palavras (16 Gbytes) e não existe endereçamento a byte ou a meia-palavra (2 bytes).

O processador possui 32 registradores de 32 bits, alguns deles com significado especial:

R00 - Contém sempre o valor 0.
R01 - Contém a palavra de estado do processador (PSW).
R31 - Contém o valor atual do contador de programa (PC).

Todas as instruções possuem o seguinte formato:

1	1	5	1	1	1	5	17
T1	T0	C4..C0	APS	F1	F0	DST	FNT

T1-T0 — Definem o tipo de instrução:
00 - Aritmético-lógica
10 - Acesso à memória
01 - Salto
11 - Sub-rotina

C4..C0 — Definem os opcodes numéricos das instruções aritmético-lógicas e de acesso à memória, ou a condição de teste nas instruções de salto e de sub-rotina.

APS — Flag para atualizar ou não a palavra de estado após a operação, se APS=1, então os bits N (negativo), O (overflow), Z (zero) e C (carry) da PSW serão atualizados de acordo com o resultado da operação.

F1-F0 — Definem qual o formato dos operandos fontes (FNT).

DST — Contém o número do registrador destino.

FNT — Contém a especificação dos operandos fontes, conforme os valores de F1 e F0.

O Risco utiliza sempre três endereços para os operandos. Usualmente as instruções possuem um operando destino e dois operandos fontes. Existem também instruções com um destino, um fonte é uma constante de 11 bits; ou com um destino, um fonte igual ao destino e uma constante de 17 bits. Estes três formatos possíveis de operandos são mostrados abaixo:

31	30	29..25	24	23	22	21..17	16..12	11	10..06	05..00
T1	T0	C4..C0	APS	F1	F0	DST	FT1	SS2	FT2	.....
T1	T0	C4..C0	APS	F1	F0	DST	FT1	SS2	Kp	
T1	T0	C4..C0	APS	F1	F0	DST	Kg			

Os bits F1 e F0 indicam o tipo do formato dos operandos que a palavra de instrução possui, como mostrado a seguir:

F1	F0	SS2	Operandos fonte
0	0	0	FT1 e FT2
0	0	1	FT1 e Kp
0	1	x	R00 e Kgl
1	0	x	DST e Kgh
1	1	x	DST e Kgl

Onde:

FT1 Contém o número do registrador do primeiro operando fonte.

FT2 Contém o número do registrador do segundo operando fonte.

SS2 Bit utilizado para definir o formato dos bits de 0 a 10.

Kp Constante de 11 bits, estende sinal.

Kgl Constante de 17 bits, estende sinal.

Kgh Constante de 17 bits, ocupa a parte mais significativa e estende o sinal para os bits menos significativos.

Havendo alguma constante na palavra de instrução, ela é carregada em um registrador interno temporário para ser utilizada como o segundo operando fonte. As diferentes constantes assumem determinados formatos dentro desse registrador.

A constante Kp tem o bit mais significativo estendido, permitindo valores entre -1024 e 1023. A constante Kg possui dois formatos diferentes. Ela pode ter o bit mais significativo estendido, Kgl, permitindo valores entre -65536 e 65535. No segundo modo, a constante é carregada nos 16 bits mais significativos do registrador temporário e seu 17° bit é estendido nos 16 bits menos significativos do registrador, Kgh.

2. ESPECIFICAÇÃO DO MONTADOR

2.1 O MONTADOR

O montador para o microprocessador Risco é um típico montador de duas passagens, que possui um processador de macros embutido. Os mnemômicos e códigos de máquina das instruções do processador estão em um arquivo (instr.tab); quando o montador é executado esses dados são lidos e colocados na memória para serem utilizados na geração do código de máquina.

Existe uma linguagem símbolica própria do montador para que os programas sejam escritos mais facilmente. A especificação dessa linguagem é feita no próximo capítulo.

O código gerado é escrito em dois arquivos. Um arquivo contém apenas o código binário e o outro contém a posição relativa na memória, o código hexadecimal e uma cópia do programa fonte.

2.2 O PROCESSADOR DE MACROS

O montador possui um processador de macros que está embutido na sua primeira passagem. Se existir alguma macro, sua definição é guardada na memória para que posteriormente, na segunda passagem, seja feita sua expansão.

O processador de macros, de uma passagem, permite apenas que macros sejam chamadas após terem sidos definidas. Não são permitidas definições internas a outras macros, mas podem ser feitas chamadas internas desde que a macro chamada já tenha sido definida.

A definição de uma macro é feita utilizando as diretivas MACRO e MACROFIM. O corpo de uma macro pode ser constituído por quaisquer tipos de comando, diretivas ou instruções.

A chamada de uma macro é feita através do nome dado durante sua definição, incluindo os parâmetros desejados, caso existirem. A sintaxe para definição e chamada de macros será apresentada mais adiante.

2.3 COMO UTILIZAR O MONTADOR

O montador Risco é utilizado através do comando de linha: risco [-n] <arquivo[.ext]>. A opção **-n** refere-se à geração automática de NOP após cada instrução de JMP. A geração de NOP é opcional. O nome do arquivo a ser montado deve ter a extensão .EXT, sendo esta opcional para execução do programa. Todas as saídas do montador, códigos e mensagens de erros são geradas em arquivos.

2.4 ARQUIVOS GERADOS

São gerados 3 arquivos com as saídas do montador. O <arquivo.bin>, que contém o código binário do programa; o <arquivo.doc>, que contém informação de alocação de memória e o código em hexadecimal; o <arquivo.err>, que contém informação de erros que possam ocorrer durante a montagem do programa. Erros de sintaxe são detectados pelo analisador léxico (LEX) e enviados para o arquivo de erros, abortando o programa. Quando houver algum erro o usuário será avisado para que o arquivo de erros seja verificado; quando o número de erros for elevado o programa será abortado e uma mensagem será enviada para a tela.

3. DESCRIÇÃO DA LINGUAGEM SIMBÓLICA

A linguagem simbólica possui alguns elementos básicos, como rótulos, registradores, constantes e comentários. A forma como cada um é representado na linguagem está descrita abaixo.

Tudo que vier depois de um ';' será considerado pelo montador como sendo um comentário.

<comentário> : ';' <texto do comentário>

Os rótulos podem aparecer no início da linha, sozinhos, ou obrigatoriamente antes de algumas diretivas, ou ainda , opcionalmente antes de instruções.

<rótulo> : <identificador> ':'

Representam-se registradores através da letra 'R' ou 'r' mais um número, ou com um identificador. Este identificador deve ter sido previamente definido utilizando a diretiva DEFINE, caso contrário ocorrerá um erro.

<reg> :
'r' <número decimal> |
'R' <número decimal> |
<identificador>

Constantes de diferentes bases são permitidas, como hexadecimal, decimal, octal e binária.

<num> :
<número com base> |
<sinal> <número com base>

<sinal> : '-'

<número com base> :
<número hexadecimal> <base hexadecimal> |
<número decimal> <base decimal> |
<número octal> <base octal> |
<número binario> <base binária>

<base hexadecimal> : 'h' | 'H'

<base decimal> : 'd' | 'D' | <vazio>

<base octal> : 'o' | 'O'

<base binária> : 'b' | 'B'

3.1 DIRETIVAS

A linguagem simbólica do montador dispõe de nove diretivas que podem ser utilizadas em qualquer parte e várias vezes dentro do programa. As suas funções são as seguintes: reservar área de memória, definir identificadores como registradores, inicializar constantes, alterar o contador de programa e definir macros.

O formato de cada uma varia, algumas possuem rótulo e outras não, o tipo e o número de parâmetros também são diferentes. A seguir, é especificado o formato e a função de cada diretiva.

1- Define registrador

DEFINE <identificador> Rxx

Define que o identificador representa o registrador Rxx.

2- Reserva área não inicializada

<rótulo> ALOCA <num>

Aloca <num> palavras para serem utilizadas como variáveis. O <rótulo> representa o endereço da primeira palavra na área alocada. A área não é inicializada e <num> deve ser uma constante de 32 bits sem sinal.

3- Reserva área e inicializa com um único valor

<rótulo> INICIA <num> ',' <num_ident>

Aloca <num> palavras para serem utilizadas como variáveis. O <rótulo> representa o endereço da primeira palavra na área alocada. A área é inicializada com o valor de <num_ident>, que deve ser uma constante de 32 bits ou um identificador que esteja associado a uma constante.

4- Reserva área e inicializa com valores diferentes

<rótulo> TABELA <num_ident> ',' <num_ident> ',' ... ',' <num_ident>

Aloca palavras para serem utilizadas como variáveis. O número de palavras depende do número de parâmetros especificados. O <rótulo> representa o endereço da primeira palavra na área alocada. A área é inicializada com os diferentes valores de <num_ident>, na ordem em que eles aparecem, podendo ser uma constante de 32 bits ou um identificador que esteja associado a uma constante.

5- Altera contador de programa

ORIGEM <num>

O contador de programa (PC) recebe o valor de <num>. As instruções que seguem serão posicionadas a partir do novo endereço. O <num> deve ser uma constante de 32 bits sem sinal. Esta diretiva pode ser utilizada várias vezes durante um mesmo módulo, mas o novo valor para o contador de posição deve ser maior ou igual ao valor atual.

6- Define sinônimo

<rótulo> EQU <num>

Associa a <rótulo> o valor de <num>. O valor de <num> deve ser uma constante de 32 bits.

7- Inicia definição de uma macro

<rótulo> MACRO <parâmetros>

Inicia a definição da macro <rótulo>. O corpo da macro é formado pelas linhas seguintes até a diretiva MACROFIM. Os parâmetros são opcionais, mas, quando existir mais de um, são separados por vírgulas. Esses parâmetros de definição são apenas identificadores.

8- Termina definição de uma macro

MACROFIM

Indica o término do corpo da macro que estava sendo definida. Deve ser precedido no programa por uma diretiva MACRO.

9- Chamada de uma macro

<identificador> <parâmetros>

Na realidade, não existe nenhuma diretiva especial para isso, apenas utiliza-se um identificador que representa o nome da macro, seguido de seus parâmetros de expansão caso a macro possua parâmetros. Se possuir mais de um parâmetro, eles devem ser separados por vírgulas. Os parâmetros de expansão podem ser identificadores, registradores, ou ainda, constantes.

Essa sintaxe para chamada de macros é a mesma para chamadas internas, dentro da definição de uma outra macro.

3.2 INSTRUÇÕES

O conjunto de instruções do Risco é formado por quatro tipos: aritmético-lógica, salto, acesso à memória (carga/armazenamento) e sub-rotina. São implementadas 25 instruções aritmético-lógica, 8 de acesso à memória, 1 de salto e 1 de sub-rotina com 16 condições cada uma, totalizando 35 instruções.

O formato básico das instruções na linguagem simbólica do montador é o seguinte:

<linha de instrução> : <rótulo> <campo da instrução> |
<campo da instrução>

<campo da instrução> : <instrução> | <vazio>

Existem outros formatos que são específicos para determinados tipos de instruções:

<instrução> : <mnemônicos aritlog e mem> <aps> <operando> |
<mnemônicos mem> <aps> <operando restrito> |
<mnemônicos jmp e sr> <condição> <aps> <operando>

Instruções aritmético-lógicas e algumas de acesso à memoria (LD|ST), que aceitam o mesmo tipo de operandos:

<mnemônicos aritlog e mem> :
ADD | ADDC |
add | addc |
SUB | SUBC | SUBR | SUBRC
sub | subc | subr | subrc |
AND | OR | XOR |
and | or | xor |
RLL | RLLC | RLA | RLAC |
rll | rllc | rla | rlac |
RRL | RRLC | RRA | RRAC |
rrl | rrlc | rra | rrac |
SLL | SLLC | SLA | SLAC |
sll | sllc | sla | slac |
SRL | SRLC | SRA | SRAC |
srl | srlc | sra | srac |
LD | ST |
ld | st

Existe também o mnemômico NOP, que não possui operandos e tem função equivalente a de uma instrução ADD R0, R0, R0.

Instruções de acesso à memória.

<mnemônico mem> :

LDPRI | LDPOI | LDPOD |
ldpri | ldpoi | ldpod |
STPRI | STPOI | STPOD |
stpri | stpoi | stpod

Instruções de salto e de sub-rotina.

<mnemônico jmp e sr> :

JMP | SR |
jmp | sr

Instruções de salto e sub-rotina podem ser condicionais ou incondicionais. No caso do salto condicional, existem as seguintes condições aceitas pela linguagem:

<mnemônico de condição> :

.TR | .NS | .CS | .OS | .ZS | .GE | .GT | .EQ |
.tr | .ns | .cs | .os | .zs | .ge | .gt | .eq |
.FL | .NN | .NC | .NO | .NZ | .LT | .LE | .NE |
.fl | .nn | .nc | .no | .nz | .lt | .le | .ne |
<vazio>

Toda instrução de máquina possui o bit APS. Por default, todas as instruções são geradas pelo montador com o bit APS desligado. Para que esse bit seja ligado, é necessário adicionar o sufixo ".APS" ou ".aps" a cada instrução desejada.

<mnemônico de aps> :

.APS |
.aps |
<vazio>

Existem três combinações básicas de operandos, todas possuem um operando destino e dois fontes. A maioria das instruções de acesso à memória aceitam apenas a combinação <operando restrito>, exceto (LD|ST).

<operando> :

<operando restrito> |
<reg> ',' <reg> ',' <exp> |
<reg> ',' <fb> ',' <exp>

<operando restrito> :

<reg> ',' <reg> ',' <reg>

O campo **fb** serve para indicar o tipo do primeiro operando fonte. Este operando pode ser o registrador 00 (&R0) ou o próprio operando destino (&RD ou &RDH). No último caso, o campo **fb** também informa qual o tipo da constante no segundo operando fonte. Pode ser uma constante de 17 bits (&RD), estendendo o sinal, indicando que é uma constante **Kgl**, ou uma constante de 17 bits (&RDH), ocupando a parte mais significativa, que é uma **Kgh**.

<fb> :

&r0 | &R0 |
&rd | &RD |
&rdh | &RDH

O terceiro operando de uma instrução pode ser um registrador, uma constante, um identificador ou, ainda, uma expressão. São permitidas expressões de soma, subtração, multiplicação e divisão; os operandos não podem ser registradores ou identificadores de registradores.

<exp> :

<oper> <operador> <exp> |
<oper>

<oper> :

<identificador> |
<num>

<operador>: '+' | '-' | '*' | '÷'

4. ESPECIFICAÇÃO DAS INSTRUÇÕES

4.1 INSTRUÇÕES ARITMÉTICO-LÓGICAS

O Risco implementa apenas 25 instruções aritmético-lógicas das 32 possíveis definidas no campo C4..C0.

Tabela de instruções aritmético-lógicas.

MNEMÔNICO	CÓDIGO (C4..C0)	OPERAÇÃO	COMENTÁRIO
ADD	0 0 0 0 0	dst ← ft1 + ft2	adição
ADDC	0 0 0 1 0	dst ← ft1 + ft2 + C	adição c/ carry
SUB	0 0 1 0 1	dst ← ft1 - ft2	subtração
SUBC	0 0 1 1 1	dst ← ft1 - ft2 - C	subtração c/ carry
SUBR	0 1 0 0 1	dst ← ft2 - ft1	subtração reversa
SUBRC	0 1 0 1 1	dst ← ft2 - ft1 - C	subtração reversa c/ carry
AND	0 1 1 1 1	dst ← ft1 and ft2	E lógico
OR	0 1 1 1 0	dst ← ft1 or ft2	OU lógico
XOR	0 1 1 0 1	dst ← ft1 xor ft2	OU exclusivo lógico
RLL	1 0 0 0 0	dst ← ft1 rot ft2 bits	rot. lógica esq.
RLLC	1 0 0 0 1	dst ← ft1 /C rot ft2 bits	rot. lógica esq. c/ carry
RLA	1 0 0 1 0	dst ← ft1 rot ft2 bits	rot. aritmética esq.
RLAC	1 0 0 1 1	dst ← ft1 /C rot ft2 bits	rot. aritmética esq. c/ carry
RRL	1 0 1 0 0	dst ← ft1 rot ft2 bits	rot. lógica dir.
RRLC	1 0 1 0 1	dst ← ft1 /C rot ft2 bits	rot. lógica dir. c/ carry
RRA	1 0 1 1 0	dst ← ft1 rot ft2 bits	rot. aritmética dir.
RRAC	1 0 1 1 1	dst ← ft1 /C rot ft2 bits	rot. aritmética dir. c/ carry
SLL	1 1 0 0 0	dst ← ft1 desl ft2 bits	desl. lógico esq.
SLLC	1 1 0 0 1	dst ← ft1 /C desl fte2 bits	desl. lógico esq. c/ carry
SLA	1 1 0 1 0	dst ← ft1 desl ft2 bits	desl. aritmético esq.
SLAC	1 1 0 1 1	dst ← ft1 /C desl ft2 bits	desl. aritmético esq. c/ carry
SRL	1 1 1 0 0	dst ← ft1 desl ft2 bits	desl. lógico dir.
SRLC	1 1 1 0 1	dst ← ft1 /C desl ft2 bits	desl. lógico dir. c/ carry
SRA	1 1 1 1 0	dst ← ft1 desl ft2 bits	desl. aritmético dir.
SRAC	1 1 1 1 1	dst ← ft1 /C desl ft2 bits	desl. aritmético dir. c/ carry

dst, ft1, ft2: operandos da palavra de instrução;
rot.: rotação; desl.: deslocamento;
dir.: direita; esq.: esquerda.

4.2 INSTRUÇÕES DE ACESSO À MEMÓRIA

São implementadas as operações de carga e armazenamento de registrador na memória, com a opção de fazer pré-incremento, pós-incremento e pós-decremento do segundo operando fonte, quando esse é um registrador. O endereço **E** é obtido através da soma dos dois operandos fontes.

Tabela de instruções de acesso à memória.

MNEMÔMICO	CÓDIGO (C4..C0)	OPERAÇÃO	COMENTÁRIOS
LD	0 X 0 X X	dst ← M[ft1 + ft2]	carga
LDPRI	0 X 1 0 1	ft2 ← ft2 + 1; dst ← M[ft1 + ft2]	carga c/ pré-inc.
LDPOI	0 X 1 0 0	dst ← M[ft1 + ft2]; ft2 ← ft2 + 1	carga c/ pós-inc.
LDPOD	0 X 0 X X	dst ← M[ft1 + ft2]; ft2 ← ft2 - 1	carga c/ pós-dec.
ST	0 X 1 1 1	M[ft1 + ft2] ← dst	armazenamento
STPRI	0 X 1 1 1	ft2 ← ft2 + 1; M[ft1 + ft2] ← dst	armaz. c/ pré-inc.
STPOI	0 X 1 0 1	M[ft1 +ft2] ← dst; ft2 ← ft2 + 1	armaz. c/ pós-inc.
STPOD	0 X 1 0 0	M[ft1 + ft2] ← dst; ft2 ← ft2 - 1	armaz. c/ pós-dec.

dst, ft1.ft2: operandos da palavra de instrução;
M[ ]: conteúdo da memória; ft1 + ft2: endereço de memória E;
inc.: incremento; dec.: decremento.

4.3 INSTRUÇÕES DE SALTO

As instruções de salto são implementadas como uma soma condicional. Para se obter o salto deve ser especificado **dst** = PC (= R31) na palavra de instrução. COND é o resultado do teste (especificado no campo C4..C0) sobre a PSW (=R01) e o endereço **E** é obtido da mesma maneira como nas instruções de acesso à memória. São utilizadas 16 condições de testes visando diminuir operações intermediárias para a obtenção do mesmo teste.

Quando não existir condição de salto, o código assumido será o da tabela abaixo.

Tabela de instrução de salto.

MNEMÔNICO	CÓDIGO (C4..C0)	OPERAÇÃO	COMENTÁRIO
JMP	1 1 1 1 1	se COND = 1 então dts ← ft1 + ft2	salto
COND: resultado do teste; dst, ft1, ft2: operandos da palavra de instrução; ft1 + ft2 = E: endereço de salto E.			

4.4 INSTRUÇÕES DE SUB-ROTINA

O registrador **dst** pode ser qualquer um dos 32 registradores escolhido pelo montador para ser o SP. A condição COND e o endereço **E** são obtidos como nas instuções de salto. O retorno da sub-rotina é realizado como uma instrução de carga de registrador, PC, da memória com pós-incremento (LDPOI).

LDPOI: (PC ← M[R00 + SP];
SP ← SP + 1;)

Quando não houver condição para a sub-rotina, o código assumido será o da tabela abaixo.

Tabela de instrução de sub-rotina.

MNEMÔNICO	CÓDIGO (C4..C0)	OPERAÇÃO	COMENTÁRIO
SR	1 1 1 1 1	se COND = 1 então dst ← dst - 1; M[dst] ← PC; PC ← ft1 + ft2	sub-rotina
COND: resultado do teste; dst, ft1, ft2: operandos da palavra de instrução; ft1 +ft2 = E: endereço de salto E; M[ ]: conteúdo da memória; dst = Rxx: endereço de pilha; PC = R31: contador de programa.			

4.5 CONDIÇÕES DE SALTO E DE SUB–ROTINA

Existem 16 opções de condições para salto ou sub-rotina. Quando for especificada a condição, serão assumidos os seguintes códigos da tabela abaixo.

Tabela para condições de salto e sub-rotina.

MNEMÔNICO	CÓDIGO (C4..C0)	COMENTÁRIO
.TR	1 1 1 1 1	soma sempre (true)
.NS	1 0 0 0 1	soma se flag Ng ligada (negative is set)
.CS	1 0 0 1 0	soma se flag Cy ligada (carry is set)
.OS	1 0 1 0 0	soma se flag Ov ligada (overflow is set)
.ZS	1 1 0 0 0	soma se flag Zr ligada (zero is set)
.GE	1 0 0 1 1	soma se maior ou igual
.GT	1 0 1 1 0	soma se maior
.EQ	1 1 1 0 0	soma se igual
.FL	0 0 0 0 0	soma nunca (false)
.NN	0 0 0 0 1	soma se flag Ng desligada (negative is not set)
.NC	0 0 0 1 0	soma se flag Cy desligada (carry is not set)
.N0	0 0 1 0 0	soma se flag Ov desligada (overflow is not set)
.NZ	0 1 0 0 0	soma se flag Zr desligada (zero is not set)
.LT	0 0 0 1 1	soma se menor
.LE	0 0 1 1 0	soma se menor ou igual
.NE	0 1 1 0 0	soma se diferente

5. EXEMPLOS

5.1 EXEMPLO 1

5.1.1 PROGRAMA FONTE

```
;
; Inicio do programa

DEFINE SP R28
DEFINE pc R31

ORIGEM 0

start:
        ld      r1,r0,r0
increm:
        add     r1,r1,1
        sub.apsr0,r1,100
        jmp.lt  r31,r0,increm
        add     r0,r0,r0        ; equivale a um nop
espera:
        jmp     r31,r0,espera
        add     r0,r0,r0        ; equivale a um nop
;
; Fim do Programa
```

5.1.2 SAÍDA DO PROGRAMA MONTADO

```
                        1       ;
                        2       ; Inicio do programa
                        3
                        4       DEFINE SP R28
                        5       DEFINE pc R31
                        6
                        7       ORIGEM 0
                        8
                        9       start:
00000000    80020000    10              ld      r1,r0,r0
                        11      increm:
00000001    00021801    12              add     r1,r1,1
00000002    0B001864    13              sub.apsr0,r1,100
00000003    463E0801    14              jmp.lt  r31,r0,increm
00000004    00000000    15              add     r0,r0,r0        ; equivale a um nop
                        16      espera:
00000005    7E3E0805    17              jmp     r31,r0,espera
00000006    00000000    18              add     r0,r0,r0        ; equivale a um nop
                        19      ;
                        20      ; Fim do Programa
```

5.2 EXEMPLO 2

5.2.1 PROGAMA FONTE

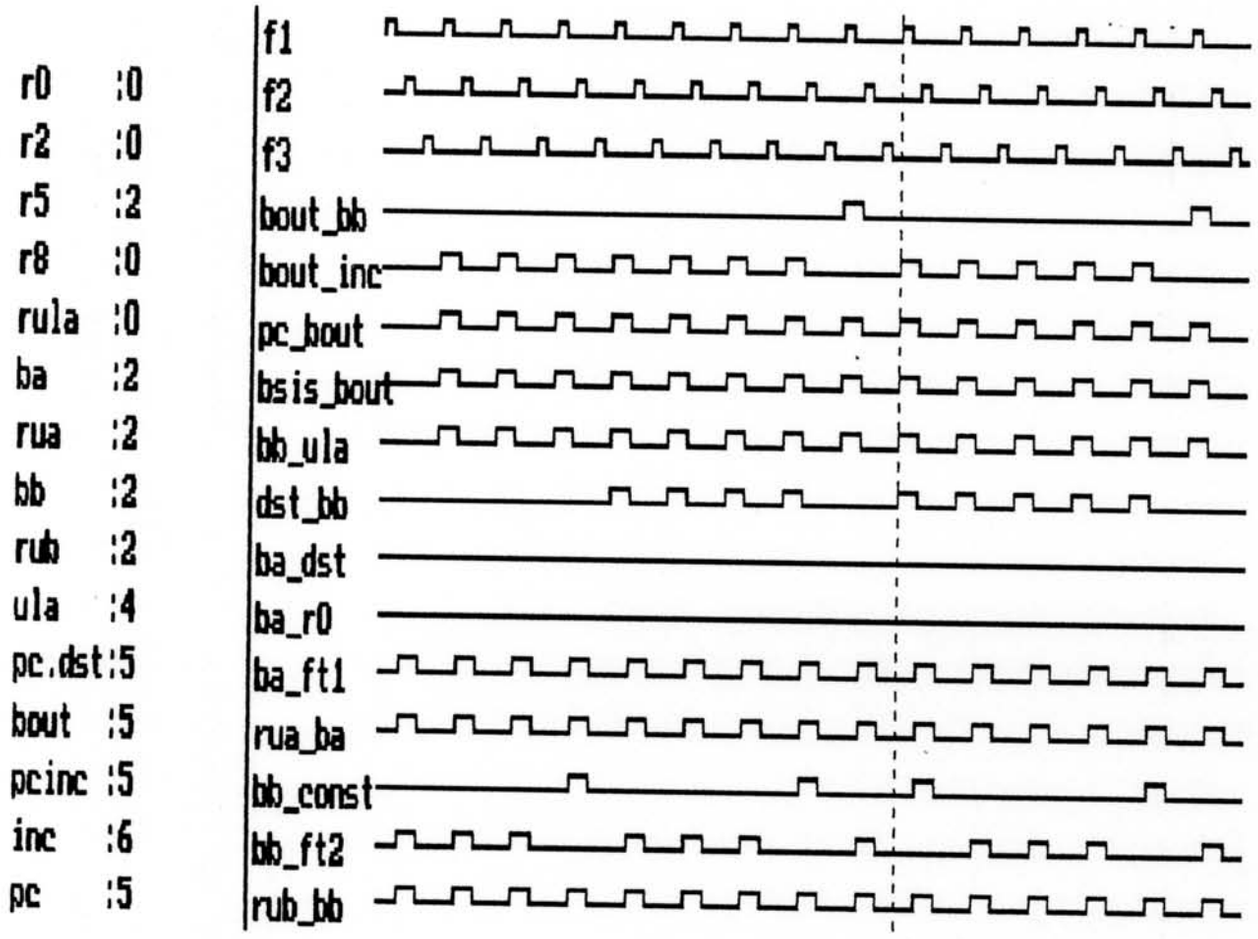
```
;
; Inicio do programa

DEFINE SP R28

CHAR:       EQU 10

ORIGEM 0

tab:    INICIA 4, CHAR

        add     r10,r0,tab      ; carrega endereco memoria valor 1
        add     r11,r0,tab+1    ; carrega endereco memoria valor 2
        add     r12,r0,tab+2    ; carrega endereco memoria valor 3
        add     r13,r0,tab+3    ; carrega endereco memoria valor 4

        ld      r20,r10,r0      ; carrega valor da mem para registrador
        ld      r21,r11,r0      ; carrega valor da mem para registrador
        ld      r22,r12,r0      ; carrega valor da mem para registrador
        ld      r23,r13,r0      ; carrega valor da mem para registrador

        add     r24,r20,r21     ;\
        add     r25,r22,r23     ; > soma valores
        add     r26,r24,r25     ;/

        add     r27,r0,4        ;\
        sra     r26,r26,r27     ;/ calcula media dos valores

        add     r14,r0,tabRAM           ; carrega endereco mem resultado

        st      r26,r14,r0      ; carrega resultado na memoria

aqui:   jmp     r31,r0,aqui     ; tranca processador
        nop

tabRAM: ALOCA   1               ;fim do programa

;
; Fim do Programa
```

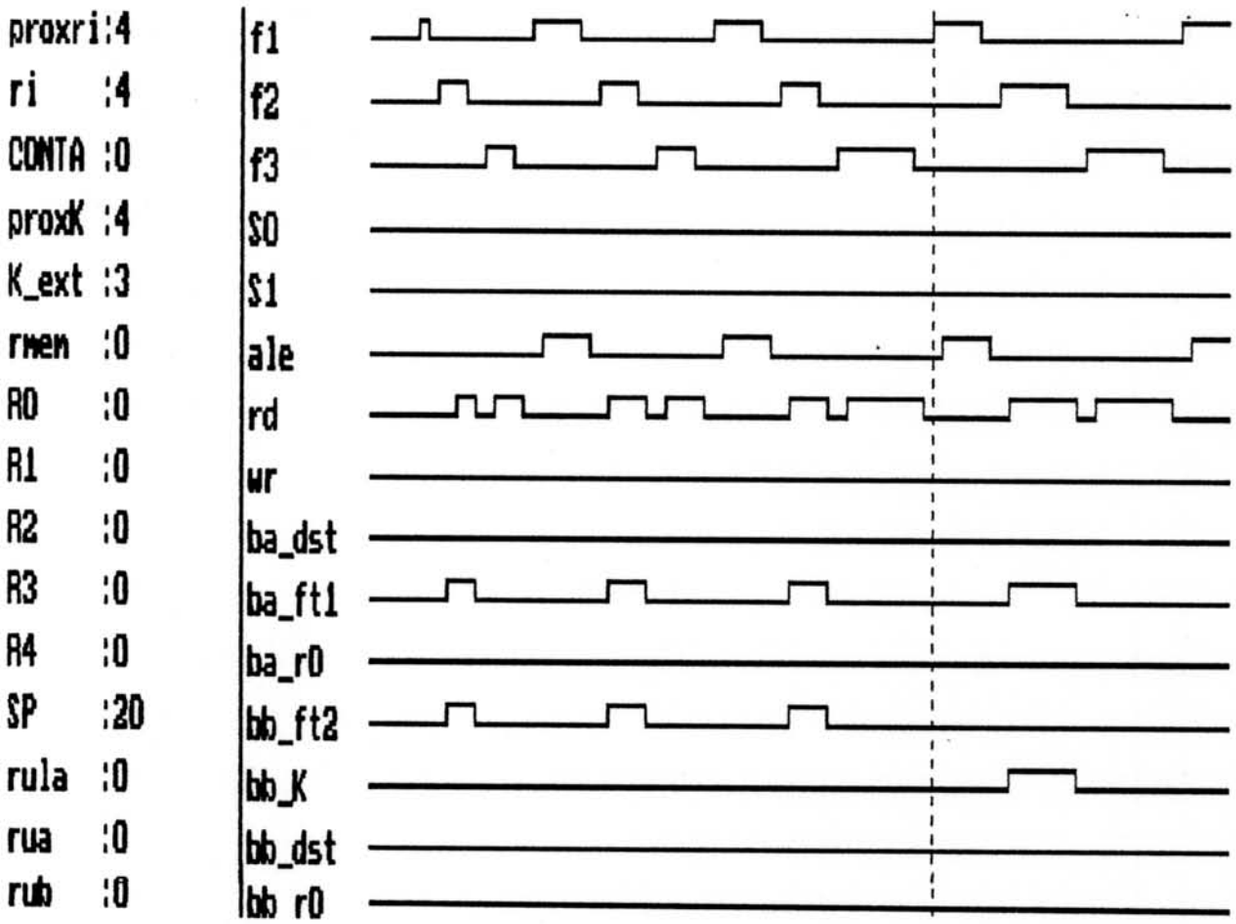
5.2.2 SAÍDA DO PROGRAMA MONTADO

```
                        1   ;
                        2   ; Inicio do programa
                        3
                        4   DEFINE SP R28
                        5
                        6   CHAR:       EQU 10
                        7
                        8   ORIGEM 0
                        9
                        10  tab:    INICIA 4, CHAR
                        11
00000004    00140800    12          add     r10,r0,tab      ; carrega endereco memoria valor 1
00000005    00160801    13          add     r11,r0,tab+1    ; carrega endereco memoria valor 2
00000006    00180802    14          add     r12,r0,tab+2    ; carrega endereco memoria valor 3
00000007    001A0803    15          add     r13,r0,tab+3    ; carrega endereco memoria valor 4
                        16
00000008    8028A000    17          ld      r20,r10,r0      ; carrega valor da mem para
registrador
00000009    802AB000    18          ld      r21,r11,r0      ; carrega valor da mem para
registrador
0000000A    802CC000    19          ld      r22,r12,r0      ; carrega valor da mem para
registrador
0000000B    802ED000    20          ld      r23,r13,r0      ; carrega valor da mem para
registrador
                        21
0000000C    00314540    22          add     r24,r20,r21     ;\
0000000D    003365C0    23          add     r25,r22,r23     ; > soma valores
0000000E    00358640    24          add     r26,r24,r25     ;/
                        25
0000000F    00360804    26          add     r27,r0,4        ;\
00000010    3C35A6C0    27          sra     r26,r26,r27     ;/ calcula media dos valores
                        28
00000011    001C0815    29          add     r14,r0,tabRAM       ; carrega endereco mem
resultado
                        30
00000012    8E34E000    31          st      r26,r14,r0      ; carrega resultado na memoria
                        32
00000013    7E3E0813    33  aqui:    jmp     r31,r0,aqui     ; tranca processador
00000014    00000000    34          nop
                        35
                        36  tabRAM: ALOCA   1               ;fim do programa
                        37
                        38  ;
                        39  ; Fim do Programa
                        40
                        41
```

ANEXO A-9

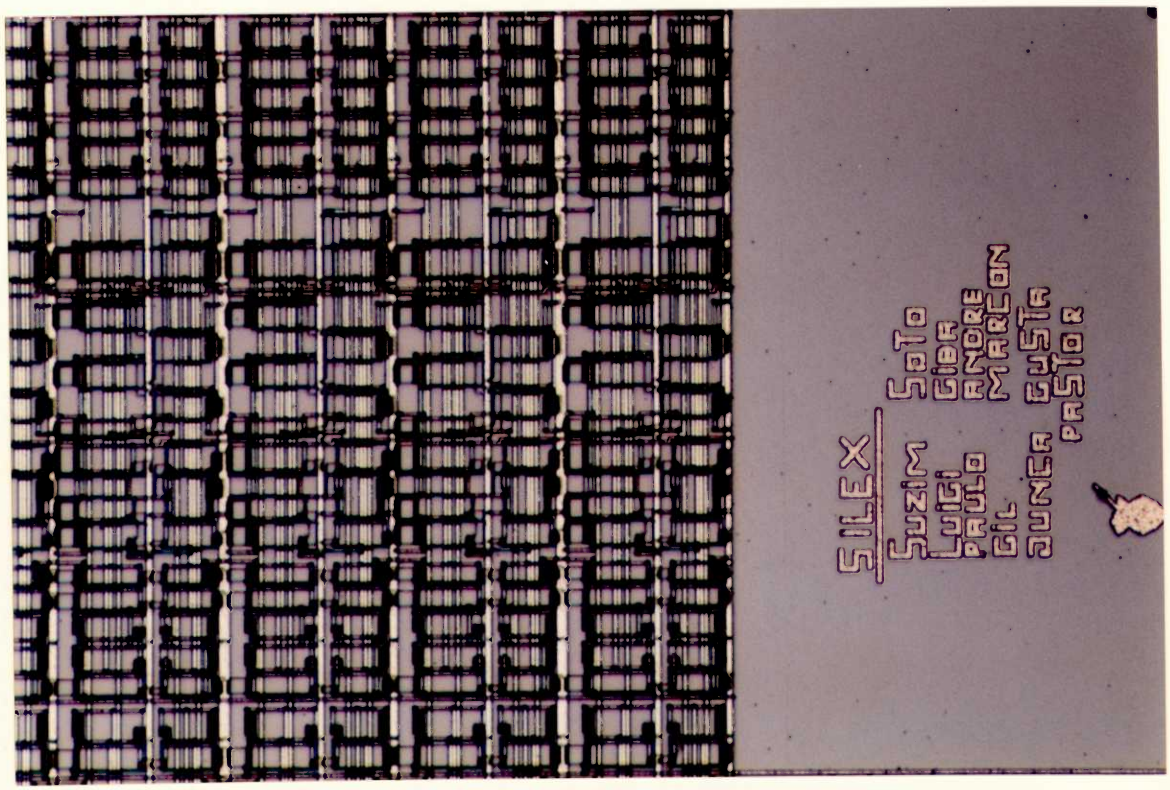
Resultado de simulações HDC

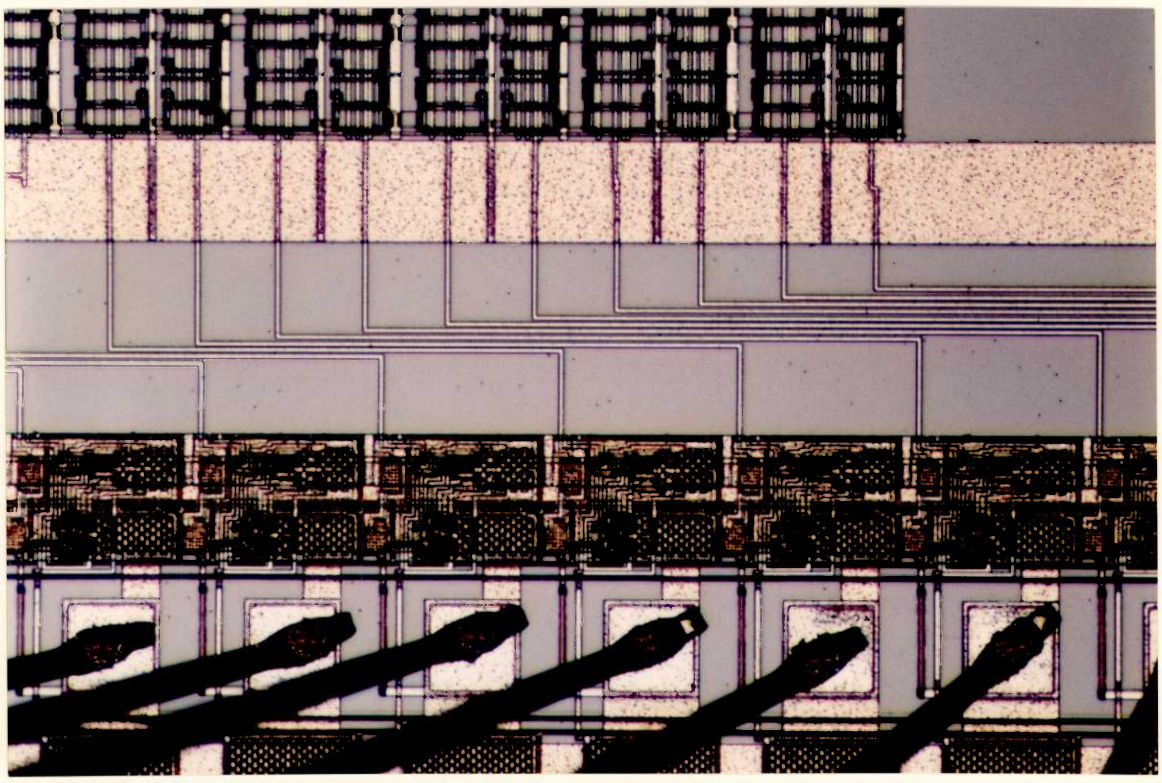

r0 :0
r2 :0
r5 :2
r8 :0
rula :0
ba :2
rua :2
bb :2
rub :2
ula :4
pc.dst :5
bout :5
pcinc :5
inc :6
pc :5
f1
f2
f3
bout_bb
bout_inc
pc_bout
bsis_bout
bb_ula
dst_bb
ba_dst
ba_r0
ba_ft1
rua_ba
bb_const
bb_ft2
rub_bb

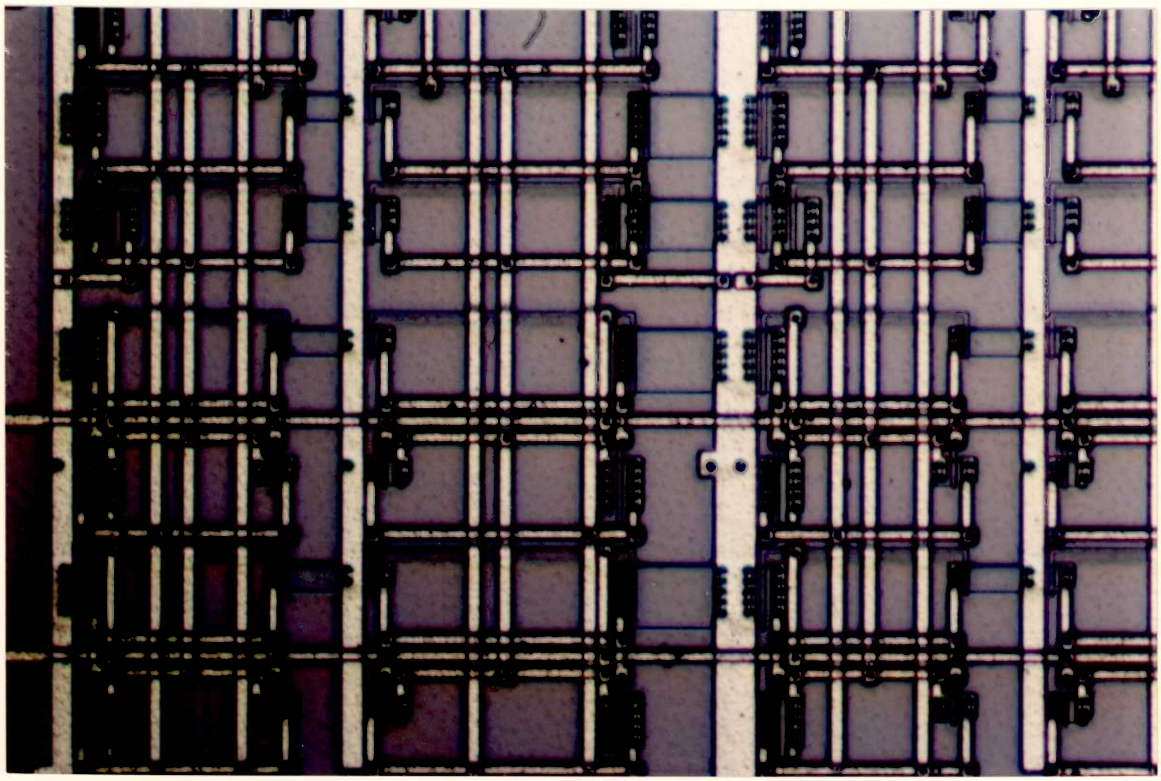

proxri :4
ri :4
CONTA :0
proxK :4
K_ext :3
rmem :0
R0 :0
R1 :0
R2 :0
R3 :0
R4 :0
SP :20
rula :0
rua :0
rub :0
f1
f2
f3
S0
S1
ale
rd
wr
ba_dst
ba_ft1
ba_r0
bb_ft2
bb_K
bb_dst
bb r0

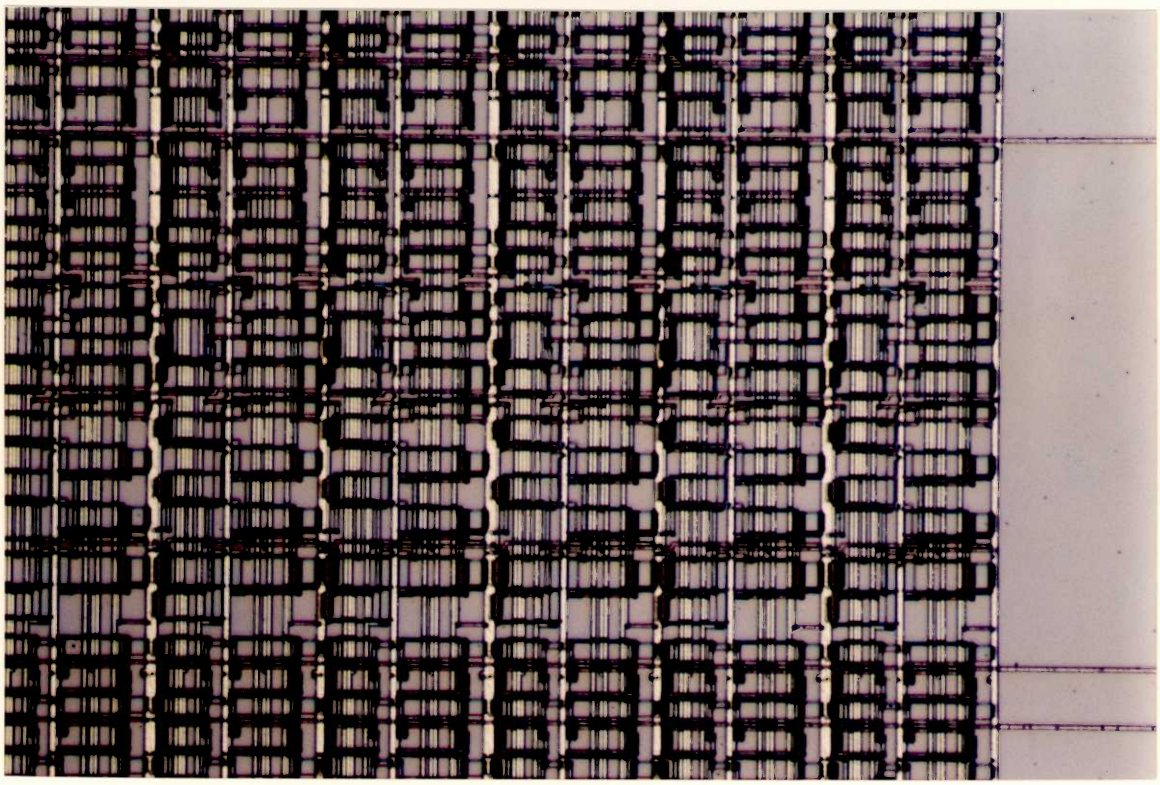
ANEXO A-10

Fotos do test-chip


SILEX
Suzim Soto
Luigi Gioa
Paulo Andre Marcon
Gil Junca Justa
Pastor







BIBLIOGRAFIA

/AGR 88/ AGRAWAL, A. et al. Design considerations for a bipolar implementation of SPARC. In: IEEE COMPCON 88, 1988, New York. **Proceedings...** New York: IEEE, 1988. p.6-9.

/AHO 77/ AHO, A. V.; ULLMAN, J. D. **Principles of compiler design**, Reading: Addison-Wesley, 1977.

/ALE 75/ ALEXANDER, G.; WORTMAN, D. Static and dynamic characteristics of XPL programs. **IEEE Computer**, Los Angeles, v.18, n.11, p.41-46, Nov. 1975.

/ANC 86/ ANCEAU, François. **The architecture of microprocessors.** Wokingham: Addison-Wesley, 1986. 251p. il. (Microelectronics Systems Design Series).

/CAR 89/ CARRO, Luigi. **Gerador parametrizável de partes operativas CMOS.** Porto Alegre: Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Rio Grande do Sul, 1989. 221p. il. (Dissertação de Mestrado).

/CAR 92/ CARRO, Luigi et al. GGMOD: A generator of module generators. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 7., jul. 8-10, 1992, São Paulo, Universidade de São Paulo. **Anais...** São Paulo: Sociedade Brasileira de Microeletrônica, 1992. p.365-374. il.

/CHA 82/ CHAITIN, G. J. Register allocation & spilling via graph coloring. In: SIGPLAN'82 SYMPOSIUM ON COMPILER CONSTRUCTION, June 23-25, 1982, Boston. **Proceedings...** New York: ACM SIGPLAN, v.17, n.6, p.98-105, june 1982.

/COL 85/ COLWELL, Robert P. et al. Computers, complexity, and controversy. **IEEE Computer**, Los Angeles, v.18, n.9, p.8-19, Sept. 1985.

/COL 85a/ COLWELL, Robert P. et al. More controversy about "Computers, complexity, and controversy". **IEEE Computer**, Los Angeles, v.18, n.12, p.93, Dec. 1985.

/CYP 90/ CYPRESS Semicondutor. **SPARC RISC user's guide.** California: Cypress, Feb. 1990. 416p.

/DOB 92/ DOBBERPUHL, Daniel W. et al. A 200-MHz 64-b dual-issue CMOS microprocessor. **IEEE Journal of Solid-State Circuits**, New York, v.27, n.11, p.1555-1567, Nov. 1992.

/FER 85/ FERNANDEZ, Eduardo B. Microprocessor architecture: the 32-bit generation. **VLSI Systems Design**, Manhasset, v.6, n.10, p.34-43, Oct. 1985.

/FIT 82/ FITZPATRICK, John K. A RISCy approach to VLSI. **Computer Architecture News**, Califórnia, v.10, n.1, p.28-32, Mar. 1982.

/FOT 84/ FOTI, D. E. et al. Reduced instruction-set multi-microcomputer system. In: PROCEEDINGS OF THE NCC, June, 1984, Las Vegas. **Proceedings...** Las Vegas: NCC, 1984. p.69-76.

/FOX 86/ FOX, E. R. et al. Reduced instruction set architecture for a GaAs microprocessor system. **IEEE Computer**, Los Angeles, v.19, n.10, p.71-81, Oct. 1986.

/GLA 85/ GLASSER, Lance A.; DOBBERPUHL, Daniel W. **The design and analysis of VLSI circuits.** Reading: Addison-Wesley, 1985. 473p. il.

/GRO 81/ GROSS, T. R.; HENNESSY, J. Optimizing delayed branchs. **ACM SIGMICRO Newsletter**, v.13, n.4, Dec. 1981.

/GRO 85/ GROSS, T. R. Floating-point arithmetic on a reduced instruction set processor. In: SYMPOSIUM ON COMPUTER ARITHMETIC, 7., June 1985. **Proceedings...** p.86-92.

/HEN 82/ HENNESSY, John et al. The MIPS machine. In: COMPUTER CONGRESS, San Francisco, Feb. 22-25, 1982, San Francisco. **Digest of Papers...** New York: IEEE, 1982. p.2-7.

/HEN 84/ HENNESSY, John L. VLSI processor architecture. **IEEE Transactions on Computers**, New York, v.33, n.12, p.1221-1246, Dec. 1984.

/HUN 89/ HUNTER, Colin; BANNING, John. DOS at RISC. **Byte**, New Jersey, p.361-368, Nov. 1989.

/KAR 89/ KARR, Paul G. Sparc gets a boost from ECL. **Computer Design**, p.72, Nov. 13, 1989.

/KAT 85/ KATEVENIS, Manolis G. H. **Reduced instruction set computer architectures for VLSI**. Cambridge: MIT Press, 1985. 215p. il. (ACM Doctoral Dissertation Awards, 1984). (Tese Ph.D.).

/KER 78/ KERNIGHAN, Brian W.; RITCHIE, Dennis M. **The C programming language.** Englewood Cliffs: Prentice-Hall, 1978. 208p.

/MAC 92/ MARCON, César A. M.; SUZIM, Altamiro A. Descrição e simulação de sistemas em linguagem de alto nível. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 7., jul. 8-10, 1992, São Paulo, Universidade de São Paulo. **Anais...** São Paulo: Sociedade Brasileira de Microeletrônica, 1992. p.341-351. il.

/MAC 92a/ MARCON, César A. M. **Planejamento estrutural e simulação de partes de controle de circuitos integrados.** Porto Alegre: Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Rio Grande do Sul, 1992. 308p. il. (Dissertação de Mestrado).

/MAN 79/ MANO, M. Morris. **Digital logic and computer design.** Englewood Cliffs: Prentice-Hall, 1979. 612p. il.

/MAN 82/ MANO, M. Morris. **Computer system architecture.** 2. ed. Englewood Cliffs: Prentice-Hall, 1982. 531p. il.

/MAR 89/ MARCHIORO, Gilberto F.; CARRO, Luigi. Editor simbólico para circuitos integrados. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 4., jul. 12-14, 1989, Porto Alegre, Universidade Federal do Rio Grande do Sul. **Anais...** Porto Alegre: Sociedade Brasileira de Microeletrônica, 1989. 2v. V.2 p.837-848.

/MAR 92/ MARCHIORO, Gilberto F. et al. Sylex - Sistema para a integração de ferramentas de projeto de circuitos integrados. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 7., jul. 8-10, 1992, São Paulo, Universidade de São Paulo. **Anais...** São Paulo: Sociedade Brasileira de Microeletrônica, 1992. p.375-384. il.

/MEA 80/ MEAD, Carver; CONWAY, Lynn. **Introduction to VLSI systems.** Reading: Addison-Wesley, 1980. 396p. il.

/MIL 88/ MILUTINOVIĆ, Veljko M. (Ed.). **Computer architecture - concepts and systems.** New York: Elsevier Science, 1988. 566p. il.

/MIL 89/ MILUTINOVIĆ, Veljko M. (Ed.). **High-level language computer architecture.** New York: Computer Science Press, 1989. 474p. il. (Advances in VLSI Series, 1).

/MYE 82/ MYERS, Glenford J. **Advances in computer architectures.** 2. ed. New York: John Wiley & Sons, 1982.

/MYE 86/ MYERS, Glenford J. et al. Microprocessor technology trends. **Proceedings of the IEEE**, New York, v.74, n.12, p.1605-1622, Dec. 1986.

/NAM 88/ NAMJOO, M. et al. CMOS custom implementation of the SPARC architecture. In: IEEE COMPCON 88, 1988, New York. **Proceedings...** New York: IEEE, 1988. p.18-20.

/NAM 88a/ NAMJOO, M. et al. CMOS gate array implementation of the SPARC architecture. In: IEEE COMPCON 88, 1988, New York. **Proceedings...** New York: IEEE, 1988. p.10-13.

/OBR 82/ OBREBSKA, Monika. **Etude comparative de différentes méthodes de conception des parties contrôle des microprocesseurs.** Grenoble: Institut National Polytechnique de Grenoble, juin 1982. 547p. il. (Tese Doct. Ing. Génie Inf.).

/PAT 80/ PATTERSON, David A.; DITZEL, David R. The case for the reduced instruction set computer. **Computer Architecture News**, New York, v.8, n.6, p.25-33, Oct. 1980.

/PAT 82/ PATTERSON, David A.; PIEPHO, Richard S. Assessing RISCs in HLL support. **IEEE Micro**, Califórnia, v.2, n.4, p.9-19, Nov. 1982.

/PAT 85/ PATTERSON, David A. Reduced instruction set computers. **Communications of the ACM**, New York, v.28, n.1, p.8-21, Jan. 1985.

/PAT 85a/ PATTERSON, David A.; HENNESSY, John L. Response to "Computers, complexity, and controversy". **IEEE Computer**, Los Angeles, v.18, n.11, p.142-143, Nov. 1985.

/RAD 83/ RADIN, George. The 801 Minicomputer. **IBM Journal of Research and Development**, v.27, n.3, p.237-246, May 1983.

/REI 90/ REIHARDT, A.; SMITH, B. Sizzling RISC systems from IBM. **Byte**, New Jersey, p.124-128, April 1990.

/SHE 82/ SHERBURNE, R. W. et al. Datapath design for RISC. In: CONFERENCE ON ADVANCED RESEARCH IN VLSI, jan. 1982, Massachussets. **Proceedings...** MIT: 1982, p.53-62.

/SHO 88/ SHOJI, Masakazu. **CMOS digital circuit technology.** Englewood Cliffs: Prentice-Hall, 1988. 434p. il.

/STA 88/ STALLINGS, William. Reduced instruction set computer architecture. **Proceedings of the IEEE**, New York, v.76, n.1, p.43-55, Jan. 1988.

/SUN 87/ SUN Microsystems. **The SPARC architecture manual.** 7. ver. Mountain View: SUN Microsystems, 1987. 199p. il.

/SUZ 81/ SUZIM, Altamiro A. **Etude des parties opératives à éléments modulaires pour processeurs monolithiques.** Grenoble: Institut National Polytechnique de Grenoble, nov. 1981. 153p. il. (Tese Doct. Ing. Génie Inf.).

/SUZ 85/ SUZIM, Altamiro A. Partes operativas CMOS. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 2., jul. 20-27, 1985, Porto Alegre. **Anais...** Porto Alegre: Sociedade Brasileira de Computação, 1985. 2v. V.2 p.197-206. il.

/SUZ 89/ SUZIM, Altamiro A. et al. Microprocessador de 16 bits com arquitetura RISC. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 4., jul. 12-14, 1989, Porto Alegre, Universidade Federal do Rio Grande do Sul. **Anais...** Porto Alegre: Sociedade Brasileira de Microeletrônica, 1989. 2v. V.1 p.425-426. il.

/SUZ 90/ SUZIM, Altamiro A. et al. Risco - um microprocessador RISC CMOS de 32 bits. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 5., out. 24-26, 1990, Ouro Preto. **Anais...** Ouro Preto: Sociedade Brasileira de Computação, 1990. 334p., p.143-155. il.

/TAB 87/ TABAK, Daniel. **Reduced instruction set computer - RISC - architecture.** Hertfordshire: Research Studies Press, 1987. 161p. il. (Industrial Control, Computers and Communications Series, 1).

/TAB 88/ TABAK, Daniel. RISC Systems. **Microprocessors and microsystems**, England, v.12, n.4, p.179-185, May 1988.

/TOD 86/ TODESCO, Antônio R. W. **Concepção de um circuito integrado do tipo processador com conjunto de instruções reduzido.** Porto Alegre: Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Rio Grande do Sul, 1986. 212p. il. (Dissertação de Mestrado).

/WES 85/ WESTE, Neil H. E.; ESHRAGHIAN, K. **Principles of CMOS VLSI design - a systems perspective.** Reading: Addison-Wesley, 1985. 531p. il.

/ZYS 83/ ZYSMAN, Eytan; SUZIM, Altamiro A. Estruturas microprogramadas para VLSI. In: SIMPÓSIO BRASILEIRO DE MICROELETRÔNICA, 3., jul. 25-27, 1983, São Paulo, Universidade de São Paulo. **Anais...** São Paulo: Universidade de São Paulo, 1983. 2v. V.1 p.59-96. il.

Informática
UFRGS

Risco - Microprocessador RISC CMOS de 32 bits.

Dissertação apresentada aos Senhores:

Prof. Dr. Altamiro Amadeu Suzim

Prof. Dr. Dante Augusto Couto Barone

Prof. Dr. Sergio Bampi

Prof. Dr. Wilhelmus Van Noije (USP)

Vista e permitida a impressão.
Porto Alegre, 20/10/93.

Prof. Dr. Altamiro Amadeu Suzim,
Orientador.

Prof. Dr. Ricardo A. da L. Reis,
Coordenador do Curso de Pós-Graduação
em Ciência da Computação.