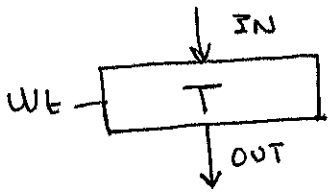
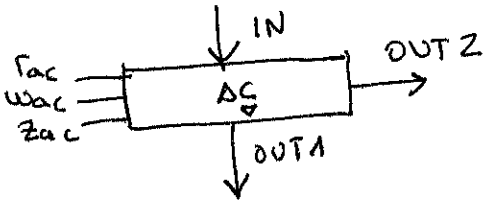
EdC. TI - Grupo 2 Prueba 1 Curso 2012/13

Problema 1

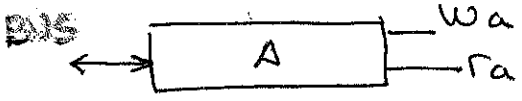
(a) descripción a nivel RT



Wt	T←	OUT=
0	T	[T]
1	IN	[T]

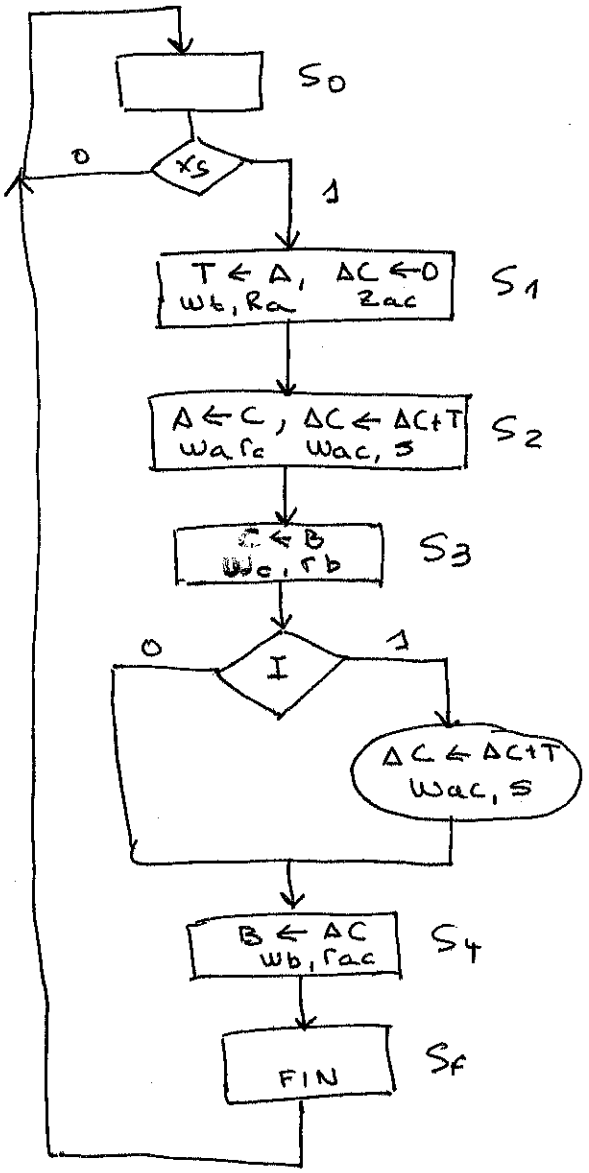


racwaczac	AC←	OUT1=	OUT2=
0 0 0	AC	HI	[AC]
0 0 1	0	HI	[AC]
0 1 0	IN	HI	[AC]
1 0 0	AC	[AC]	[AC]
resto	proh.	proh.	proh.



Wa ra	A ←	BUS =
0 0	A	HI
0 1	A	[A]
1 0	INOUT	entrada
1 1	proh	proh

(b) Carta ASM



(c) Descripción Verilog de los registros T y A

```
module registroT (input wt, input [7:0] IN,
                  output reg [7:0] OUT);

always @ (posedge Ck)

if (wt)
    OUT <= IN;

endmodule
```

```
module registroA (input Wa, Ra, inout [7:0] BUS);

reg [7:0] q;
always @ (posedge Ck)
if (Wa)
    q <= BUS
assign BUS = Ra? q : 'bZ ;
endmodule
```

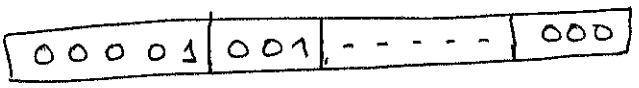
(d) Descripción Verilog de la carta

```
module Unidad_de_Control (input clk, reset, I, Xs, output reg Wt,
        Rac, Wac, Zac, Wa, Ra, Wb, Rb, Wc, Rc, FIN, S, R);

parameter  S0 = 3'b000, S1 = 3'b001, S2 = 3'b010,
           S3 = 3'b011, S4 = 3'b100, SF = 3'b101;

reg [2:0]  current_state, next_state;

always @ (posedge clk, posedge reset)
  begin  if (reset)  current_state <= S0;
         else        current_state <= next_state;
  end

always @ (current_state, Xs, I)
  begin
     Wt=0; Rac=0; Wac=0; Zac=0; Wa=0; Ra=0; Wb=0;
     Rb=0; Wc=0; Rc=0; FIN=0; S=0; R=0;
```

```
case (current_state)
   S0: if (xs)  next_state <= S1;
       else     next_state <= S0;
   S1: begin
       wt = 1; ra = 1; zac = 1;
       next_state <= S2;
       end
   S2: begin
       wa = 1; rc = 1; wac = 1; s = 1;
       next_state <= S3;
       end
   S3: begin
       wc = 1; rb = 1;
       if (I) begin wac = 1; s = 1; end
       next_state <= S4;
       end
   S4: begin
       wb = 1; rac = 1;
       next_state <= SF;
       end
   SF: begin
       fin = 1;
       next_state <= S0;
       end
endcase
end
endmodule
```

Problema 2

(a) LD R1, (R0)



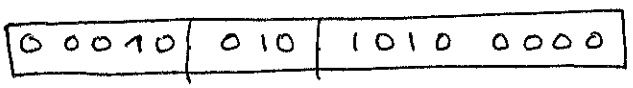
código máquina

Esta instrucción carga en R1 el contenido de la memoria direccionado por el registro R0.

Son sus microoperaciones:

1. $AC \leftarrow REG(IR_{2-0})$, $W_{AC}, OP_3, OP_2,$
2. $MAR \leftarrow AC$, R_{AC}, W_{MAR}
3. $MDR \leftarrow MEM(MAR)$, I/O*, W_{MDR}, R_{MEM}
4. $REG(IR_{10-8}) \leftarrow MDR$ I/O*, W_{REG}

STS 160, R2 160 = 0xA0



código máquina

Esta instrucción carga el contenido de R2 en la memoria, en la dirección 0xA0

microoperaciones:

1. $AC \leftarrow IR_{7-0}$ $W_{AC}, OP_3, OP_2,$ INM
2. $MAR \leftarrow AC$, $AC \leftarrow REG(IR_{10-8})$ W_{MAR}, R_{AC}, W_{AC} OP_2, OP_1
3. $MDR \leftarrow AC$ W_{MDR}, R_{AC}
4. $MEM(MAR) \leftarrow MDR$ W_{MEM}

CALL 0x10

00100	---	0001 0000

Esta instrucción llama a una subrutina que se encuentra escrita a partir de la dirección de memoria 0x10.

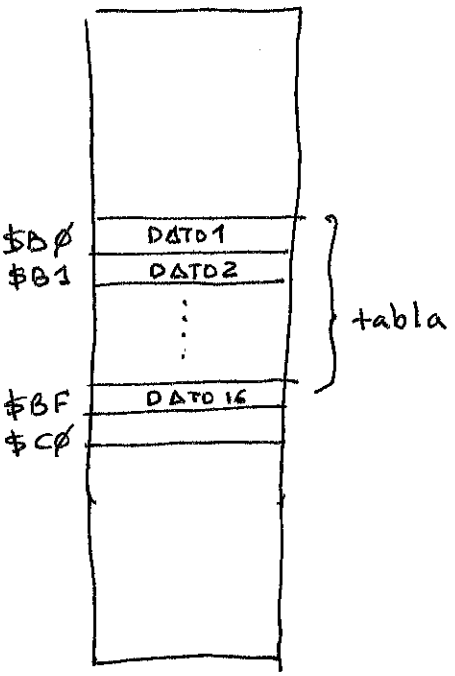
Microoperaciones

1. MDR ← PC , AC ← IR_{7-0} , OP_3, OP_2, INM, W_{MDR} R_{PC}, W_{AC}
2. MAR ← SP , SP ← SP−1, W_{MAR}, R_{SP}, D_{SP}
3. MEM (MAR) ← MDR , PC ← AC, W_{MEM}, W_{PC}, R_{AC}

(b) → teoría

(c) → teoría

(d) Programa:



Utilizaré:

- R0 como puntero para recorrer la tabla, (es decir, almacenaré en él las direcciones de los diferentes datos de forma sucesiva)
- R1 para guardar el DATO más grande encontrado hasta el momento
- R2 para almacenar en cada iteración el DATO actual

```
detectamayor:  ldi   R0, $B0     //inicializo puntero
               ld    R1, (R0)    // el 1er elto. es el mayor por ahora
bucle:         addi  RØ, 1       // apunto al siguiente
               cpi   R0, $C0     // veo si he llegado al final de la tabla
               Breq  fin
               lds   R2,(RØ)     // tomo un nuevo elto
               cp    R1, R2      // lo comparo con el mayor
               brlt  cambiamayor // si el nuevo es mayor hay que sustituir al anterior
               jmp   bucle
cambiamayor:   mov   R1, R2
               jmp   bucle
fin :          sts   $C0, R1     // almaceno resultado en la memoria y retorno.
               ret
```