UNIVERSIDAD EAFIT

PROYECTO 3

Presentado por:

Isabella Ocampo Sánchez

Curso:

Organización de computadores

Profesor:

Jose Luis Montoya Pareja

Universidad EAFIT

Ingeniería de Sistemas

Medellín, Colombia

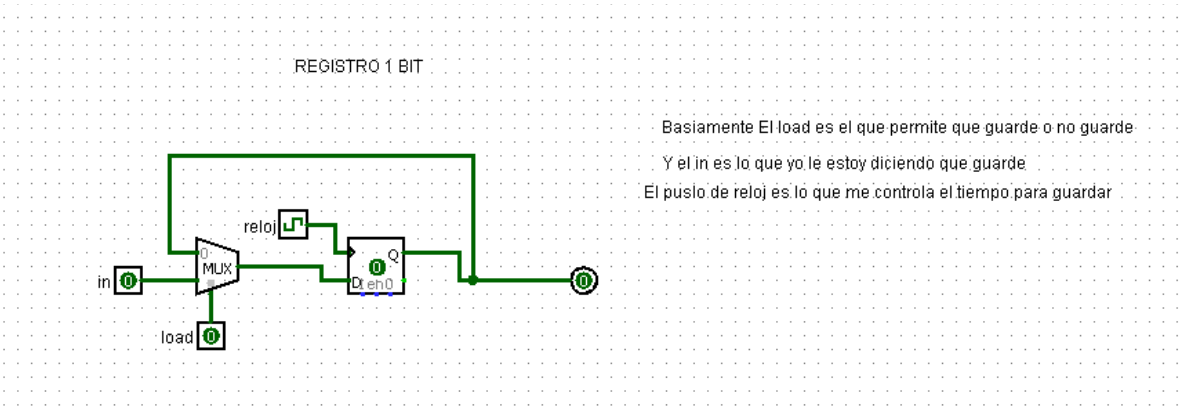
2026

UNIVERSIDAD EAFIT

Tabla de contenido

CHIP 1-BIT

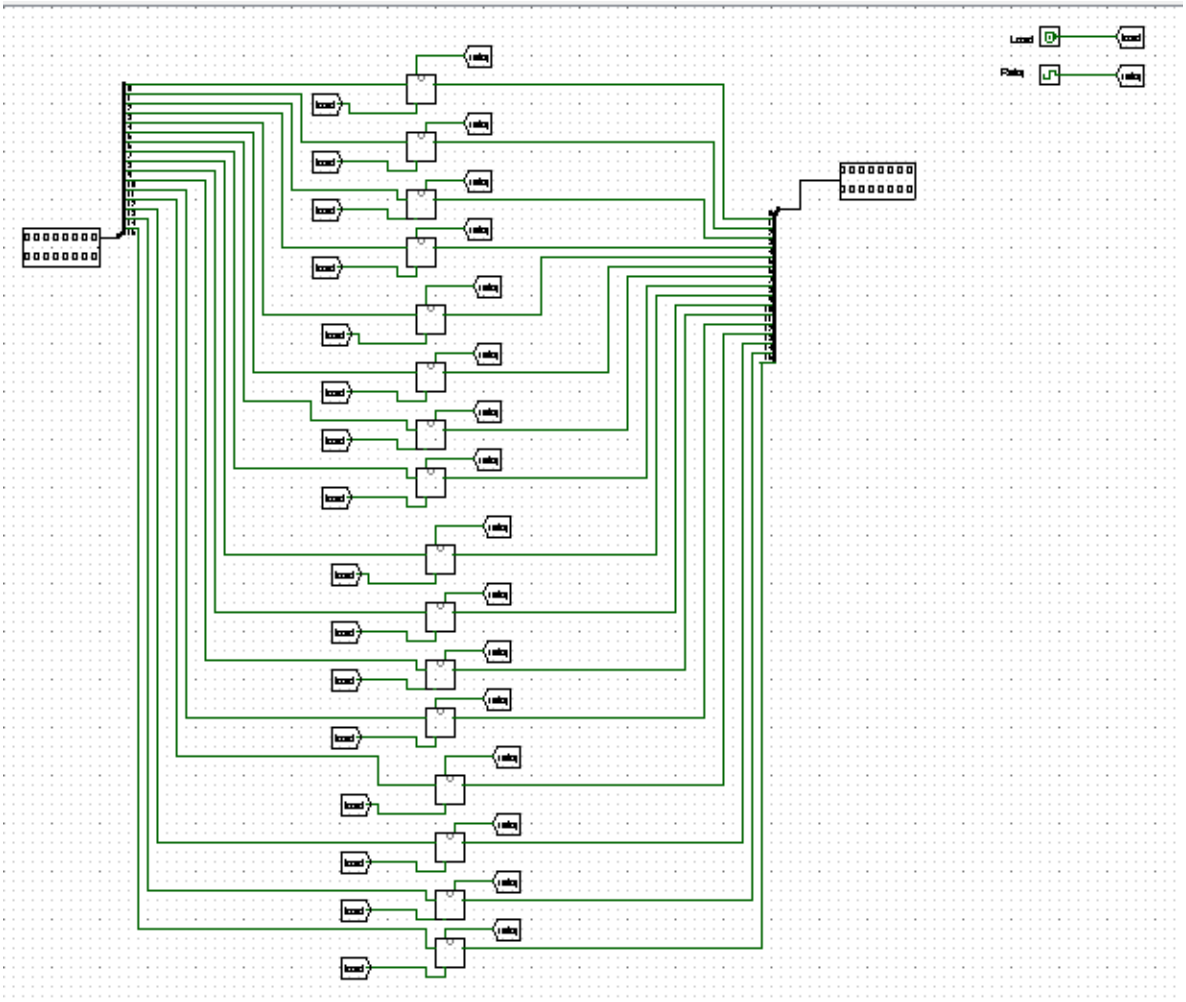
```
CHIP Bit {

   IN in, load;

   OUT out;

   PARTS:

   DFF(in= out2, out=flip, out=out );

   Mux(a= flip, b= in, sel= load, out=out2);

}
```



UNIVERSIDAD EAFIT

CHIP 16BITS- REGISTRO

```
CHIP Register {

    IN in[16], load;

    OUT out[16];


    PARTS:

    Bit(in=in[0] , load= load, out=out[0] );

    Bit(in=in[1] , load= load, out=out[1] );

    Bit(in=in[2] , load= load, out=out[2] );

    Bit(in=in[3] , load= load, out=out[3] );

    Bit(in=in[4] , load= load, out=out[4] );

    Bit(in=in[5] , load= load, out=out[5] );

    Bit(in=in[6] , load= load, out=out[6] );

    Bit(in=in[7] , load= load, out=out[7] );

    Bit(in=in[8] , load= load, out=out[8] );

    Bit(in=in[9] , load= load, out=out[9] );

    Bit(in=in[10] , load= load, out=out[10] );

    Bit(in=in[11] , load= load, out=out[11] );

    Bit(in=in[12] , load= load, out=out[12] );

    Bit(in=in[13] , load= load, out=out[13] );

    Bit(in=in[14] , load= load, out=out[14] );

    Bit(in=in[15] , load= load, out=out[15] );

}
```



RAM 8

CHIP RAM8 {

IN in[16], load, address[3];

OUT out[16];

PARTS:

DMux8Way(in=load, sel=address, a=load0, b=load1, c=load2, d=load3, e=load4, f=load5, g=load6, h=load7);

Register(in=in, load=load0, out=out0);

Register(in=in, load=load1, out=out1);

Register(in=in, load=load2, out=out2);

Register(in=in, load=load3, out=out3);

Register(in=in, load=load4, out=out4);

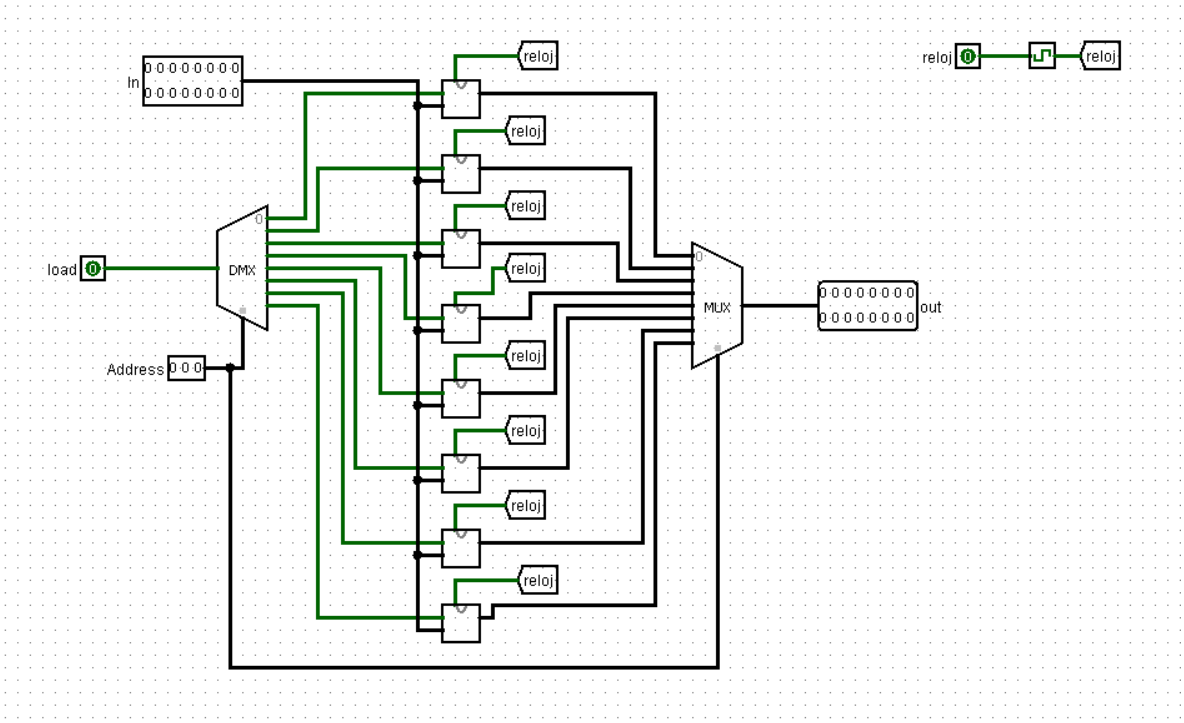
Register(in=in, load=load5, out=out5);

Register(in=in, load=load6, out=out6);

Register(in=in, load=load7, out=out7);

Mux8Way16(a=out0, b=out1, c=out2, d=out3, e=out4, f=out5, g=out6, h=out7, sel=address, out=out);

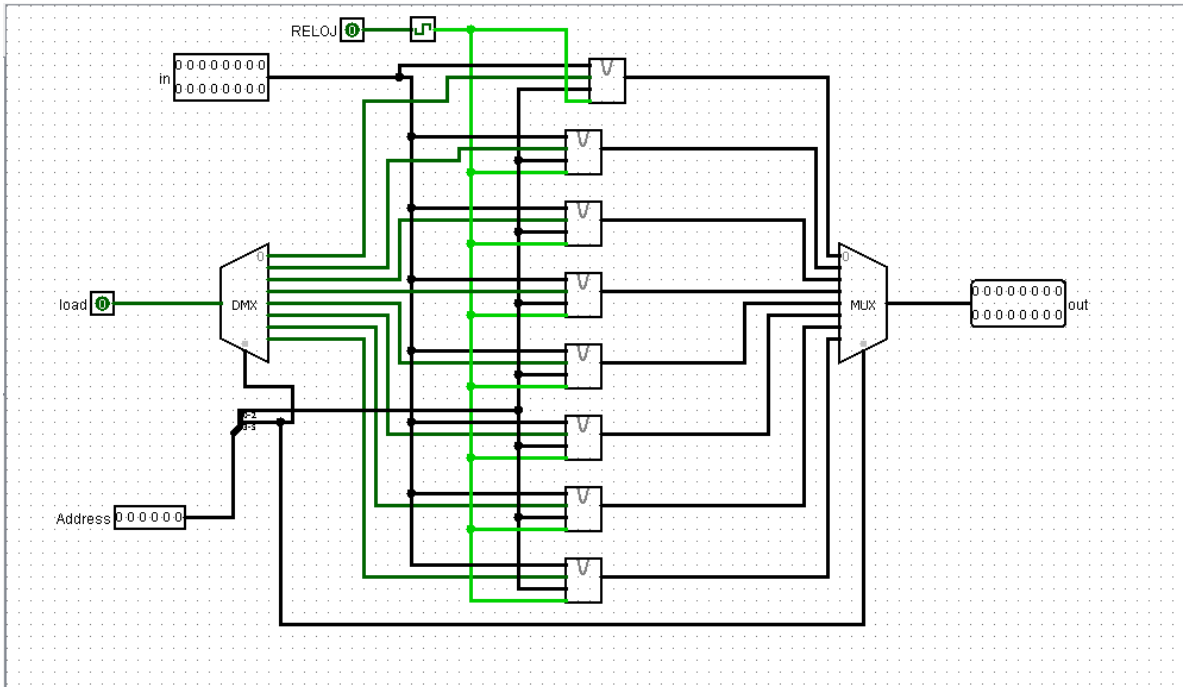
}



UNIVERSIDAD EAFIT

RAM 64

```
CHIP RAM64 {

   IN in[16], load, address[6];

   OUT out[16];


   PARTS:

   DMux8Way(in=load , sel=address[3..5] , a= load0, b=load1 , c=load2 , d=load3, e=load4 ,
f=load5 , g= load6, h=load7 );

   RAM8(in= in, load=load0 , address= address[0..2], out=out0 );

   RAM8(in= in , load= load1, address= address[0..2], out=out1 );

   RAM8(in= in, load= load2, address= address[0..2], out=out2 );

   RAM8(in= in, load=load3 , address= address[0..2], out=out3 );

   RAM8(in= in, load=load4 , address=address[0..2] , out=out4 );

   RAM8(in= in, load=load5 , address= address[0..2], out=out5 );

   RAM8(in= in, load=load6 , address= address[0..2], out=out6 );

   RAM8(in= in, load=load7 , address= address[0..2], out=out7 );

   Mux8Way16(a=out0  , b=out1  , c=out2  , d= out3 , e=out4  , f= out5 , g=out6  , h=out7  ,
sel= address[3..5], out= out);

}
```



RAM 512

CHIP RAM512 {

IN in[16], load, address[9];

OUT out[16];

PARTS:

DMux8Way(in=load , sel=address[6..8] , a= load0, b=load1 , c=load2 , d=load3, e=load4 , f=load5 , g= load6, h=load7);

RAM64(in= in, load=load0 , address= address[0..5], out=out0);

RAM64(in= in , load= load1, address= address[0..5], out=out1);

RAM64(in= in, load= load2, address= address[0..5], out=out2);

RAM64(in= in, load=load3 , address= address[0..5], out=out3);

RAM64(in= in, load=load4 , address=address[0..5] , out=out4);

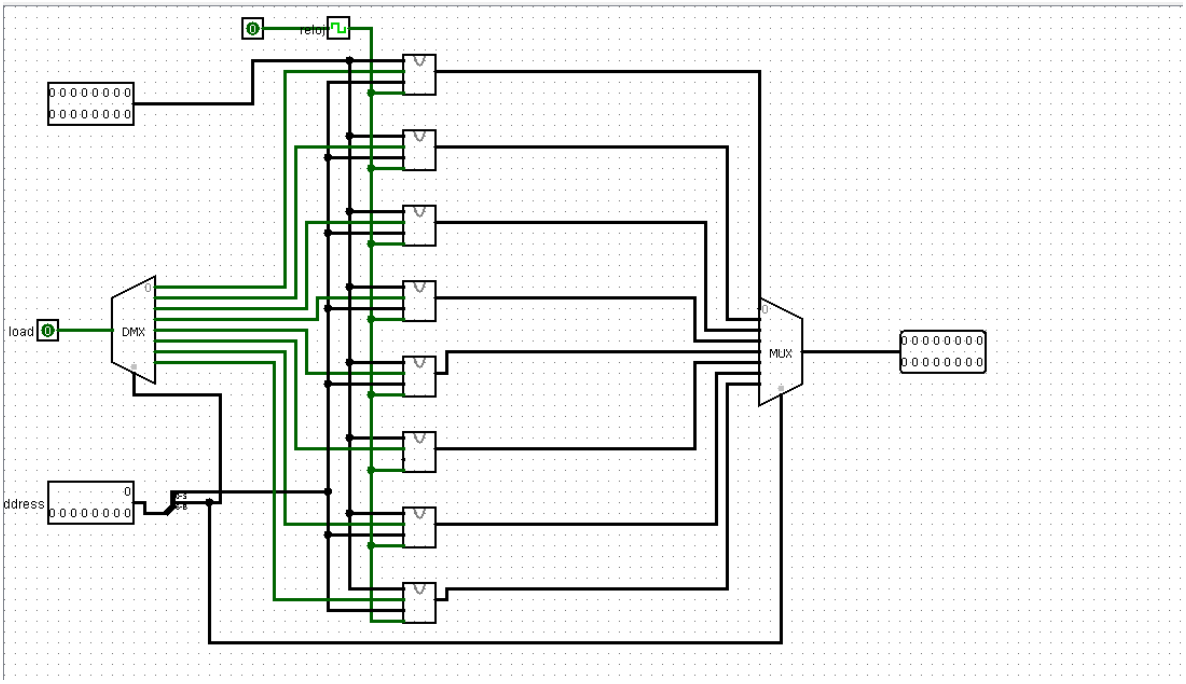
RAM64(in= in, load=load5 , address= address[0..5], out=out5);

RAM64(in= in, load=load6 , address= address[0..5], out=out6);

RAM64(in= in, load=load7 , address= address[0..5], out=out7);

Mux8Way16(a=out0 , b=out1 , c=out2 , d= out3 , e=out4 , f= out5 , g=out6 , h=out7 , sel= address[6..8], out= out);

}



ALU

CHIP ALU {

IN

x[16], y[16], // 16-bit inputs

zx, // zero the x input?

nx, // negate the x input?

zy, // zero the y input?

ny, // negate the y input?

f, // compute (out = x + y) or (out = x & y)?

no; // negate the out output?

OUT

out[16], // 16-bit output

```
    zr,     // if (out == 0) equals 1, else 0
    ng;     // if (out < 0)  equals 1, else 0

PARTS:
// --- PROCESAMIENTO DE X ---
Mux16(a=x, b=false, sel=zx, out=x1);
Not16(in=x1, out=notX1);
Mux16(a=x1, b=notX1, sel=nx, out=xFinal);

// --- PROCESAMIENTO DE Y ---
Mux16(a=y, b=false, sel=zy, out=y1);
Not16(in=y1, out=notY1);
Mux16(a=y1, b=notY1, sel=ny, out=yFinal);

// --- OPERACIÓN (f) ---
Add16(a=xFinal, b=yFinal, out=sumaXY);
And16(a=xFinal, b=yFinal, out=andXY);
Mux16(a=andXY, b=sumaXY, sel=f, out=resultadoPreNo);

// --- SALIDA FINAL Y NEGACIÓN (no) ---
Not16(in=resultadoPreNo, out=notRes);

// El bus-slicing (out[15]=ng, etc.) es la clave para las banderas
Mux16(a=resultadoPreNo, b=notRes, sel=no, out=out, out[15]=ng, out[0..7]=parteBaja,
out[8..15]=parteAlta);
```

```
    // --- CÁLCULO DE ZR Y ng---
    Or8Way(in=parteBaja, out=orBajo);
    Or8Way(in=parteAlta, out=orAlto);
    Or(a=orBajo, b=orAlto, out=noEsCero);
    Not(in=noEsCero, out=zr);
}
```

PC

```
CHIP PC {
    IN in[16], reset, load, inc;
    OUT out[16];

    PARTS:
    Inc16(in=currentOut , out=currentOut1 );
    Mux16(a=currentOut , b=currentOut1 , sel=inc , out=outmux1 );
    Mux16(a=outmux1 , b=in , sel=load , out=outmux2 );
    Mux16(a=outmux2 , b=false , sel=reset , out=outmux3 );
    Register(in=outmux3 , load=true , out=out, out=currentOut);
```