Orgasmall en Verilog

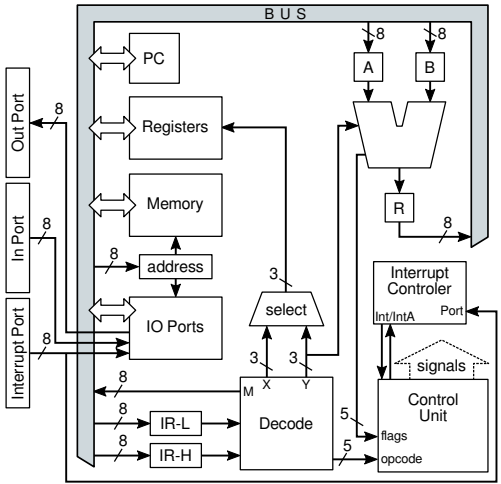
David Alejandro González Márquez

Programación de softcores en FPGAs
Programa de Profesoras/es Visitantes
Departamento de computación
Universidad de Buenos Aires

Clase disponible en: `https://github.com/fokerman/fpgaSoftcoreProgrammingCourse`

Arquitectura `OrgaSmall`

`OrgaSmall` es un procesador (*system on chip*) diseñado e implementado con fines didácticos. Su implementación en *Verilog* se denominada **`OrgaSmallSystem`**.



- Arquitectura *von Neumann*, memoria de datos e instrucciones compartida.
- 8 registros de propósito general, R0 a R7.
- 1 registro de propósito específico PC.
- Tamaño de palabra de 8 bits e instrucciones de 16 bits.
- Memoria de 256 palabras de 8 bits.
- Bus de 8 bits.
- Diseño microprogramado.
- Soporta 3 puertos, 2 de entrada y 1 de salida mapeados a memoria.
- Una interrupción asociada a cambios sobre uno de los puertos de entrada.

Arquitectura OrgaSmall

Codificación de instrucciones

Las instrucciones son de 16 bits en 4 posibles codificaciones.
Los primeros 5 bits indentifican el opcode de la instrucción, el resto de los bits indican sus parámetros.

Caso	Codificación	Parámetros
A	00000 XXXYYY-----	XXX = Registro X, YYY = Registro Y o inmediato
B	00000 XXX--------	XXX = Registro X
C	00000 ---MMMMMMMM	MMMMMMMM = Dirección de memoria o Inmediato
D	00000 XXXMMMMMMMM	XXX = Registro X, MMMMMMMM = Dir. de memoria o Imm.

- Los bits XXX codifican 8 registros posibles.
- Los bits YYY codifican 8 registros posibles o un valor inmediato de 3 bits (*shift*).
- Los bits MMMMMMMM codifican una dirección de memoria absoluta o un valor inmediato de 8 bits.
- Los valores indicados por – deben valer cero.

Existen 32 opcodes posibles, 29 codifican instrucciones y 3 son reservados.

Observación: En este diseño los opcode reservados dependen de la microarquitectura.

Arquitectura OrgaSmall

Conjunto de instrucciones

Rx o **Ry**: Índices de registros, número entre 0 y 7.

M: Dirección de memoria o valor inmediato, número de 8 bits.

t: Desplazamiento, número entre 0 y 7. Se codifica como YYY en 3 bits.

Operador **|**: Identifica el registro usado como tope de la pila (ej. |Rx|).

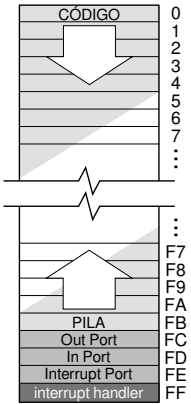
|R7| es el registro de pila obligatorio para atender interrupciones.

La instrucción de opcode 15 es libre o reservada para futuras extensiones.

Instrucción	Acción	Codificación
Reservada	Codifica el *Fetch* de instrucciones	00000----------
ADD Rx, Ry	Rx ← Rx + Ry	00001XXXYYY-----
ADC Rx, Ry	Rx ← Rx + Ry + Flag_C	00010XXXYYY-----
SUB Rx, Ry	Rx ← Rx - Ry	00011XXXYYY-----
AND Rx, Ry	Rx ← Rx and Ry	00100XXXYYY-----
OR Rx, Ry	Rx ← Rx or Ry	00101XXXYYY-----
XOR Rx, Ry	Rx ← Rx xor Ry	00110XXXYYY-----
CMP Rx, Ry	Modifica flags de Rx - Ry	00111XXXYYY-----
MOV Rx, Ry	Rx ← Ry	01000XXXYYY-----
PUSH |Rx|, Ry	Mem[Rx] ← Ry ; Rx ← Rx-1	01001XXXYYY-----
POP |Rx|, Ry	Rx ← Rx+1 ; Ry ← Mem[Rx]	01010XXXYYY-----
CALL |Rx|, Ry	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← Ry	01011XXXYYY-----
CALL |Rx|, M	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← M	01100XXXMMMMMMMM
RET |Rx|	Rx ← Rx+1 ; PC ← Mem[Rx]	01101XXX--------
RETI |Rx|	Rx ← Rx+1 ; PC ← Mem[Rx] ; Rx ← Rx+1 ; Flags ← Mem[Rx]	01110XXX--------
Libre		01111----------
STR [M], Rx	Mem[M] ← Rx	10000XXXMMMMMMMM
LOAD Rx, [M]	Rx ← Mem[M]	10001XXXMMMMMMMM
STR [Rx], Ry	Mem[Rx] ← Ry	10010XXXYYY-----
LOAD Rx, [Ry]	Rx ← Mem[Ry]	10011XXXYYY-----
JMP M	PC ← M	10100---MMMMMMMM
JC M	Si flag_C=1 entonces PC ← M	10101---MMMMMMMM
JZ M	Si flag_Z=1 entonces PC ← M	10110---MMMMMMMM
JN M	Si flag_N=1 entonces PC ← M	10111---MMMMMMMM
JO M	Si flag_O=1 entonces PC ← M	11000---MMMMMMMM
SHRA Rx, t	Rx ← Rx >>> t	11001XXXYYY-----
SHR Rx, t	Rx ← Rx >> t	11010XXXYYY-----
SHL Rx, t	Rx ← Rx << t	11011XXXYYY-----
READF Rx	Rx ← Flags	11100XXX--------
LOADF Rx	Flags ← Rx	11101XXX--------
SET Rx, M	Rx ← M	11110XXXMMMMMMMM
Reservada INT	Mem[R7] ← Flags ; R7 ← R7-1 ; Mem[R7] ← PC ; R7 ← R7-1	11111----------

Arquitectura OrgaSmall

Pila y Palabra de estado



La pila está implementada **en memoria**,
crece en el sentido de las **direcciones más bajas**.
El registro utilizado como tope de la pila,
apunta a la **primer dirección libre** en la pila.

Las intrucciones PUSH, POP, CALL, RET y RETI son las únicas que operan con la pila, **además de las interrupciones**.

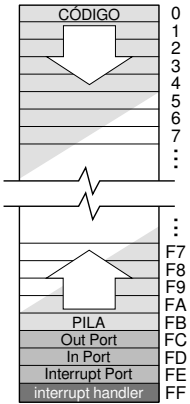
La palabra de estado es almacenada en la ALU, permitiendo ser modificada mediante dos operaciones específicas.

El orden de los bits de la palabra de estado es el siguiente:
000I ONZC (desde más significativo a menos significativo).

Donde I es el *flag* de habilitación de interrupciones,
O (*overflow*), N (*negative*), Z (*zero*) y C (carry).

Arquitectura OrgaSmall

Entrada-Salida e Interrupciones



El sistema cuenta con 3 puertos de 8 bits:

- `0xFC` : **`OutPort`** : Puerto de salida.
- `0xFD` : **`InPort`** : Puerto de entrada.
- `0xFE` : **`InterruptPort`** : Puerto de entrada sensible a interrupciones.

Con el *flag* de habilitación de interrupciones activado, el puerto `InterruptPort` genera una interrupción por cada **cambio de estado en alguno de sus bits**.

Detectada la interrupción, el sistema carga en la pila los *flags*, luego el *PC* y por último se salta a la rutina de atención de interrupciones, tomando la dirección en `0xFF`.

Para todo este proceso, se utiliza como *stack pointer* el registro R7 que en el caso inicial se debe cargar en `0xFB`.

Procesador OrgaSmall: Componentes

Consiste en 8 componentes interconectados.

- Registers (Banco de Registros)
- PC (Contador de Programa)
- ALU (Unidad Aritmético Lógica)
- Memory (Memoria)
- IOports (Puertos de Entrada/Salida)
- Decode (Decodificador de Instrucciones)
- ControlUnit (Unidad de Control)
- InterruptControl (Controlador de Interrupciones)

Cada uno de estos componentes es controlado desde la unidad de control por medio de las señales:

00	RB_enIn	08	ALU_enA	16	JC_microOp	24	DE_enOutImm
01	RB_enOut	09	ALU_enB	17	JZ_microOp	25	DE_loadL
02	RB_selectIndexIn	10	ALU_enOut	18	JN_microOp	26	DE_loadH
03	RB_selectIndexOut	11	ALU_opW	19	JO_microOp	27	INT_ack
04	RB_selectSP	12	ALU_OP_0	20	PC_load	28	-
05	MM_enOut	13	ALU_OP_1	21	PC_inc	29	load_Int_microOp
06	MM_load	14	ALU_OP_2	22	PC_enOut	30	load_microOp
07	MM_enAddr	15	ALU_OP_3	23	-	31	reset_microOp

Procesador OrgaSmall: modules

El módulo del sistema se define como:

```
module OrgaSmallSystem(clk, reset, portOutput, portInput, portInterrupt);
```

El mismo está compuesto por los siguientes componentes:

```
module ArithmeticLogicUnit(clk, reset, A, B, O, enA, enB, enOut, OP, shift, flags, opW);
    ...
module Registers(clk, reset, inData, outData, enIn, enOut, selIn, selOut, setSP);
    ...
module ProgramCounter(clk, reset, inValue, outValue, PC_load, PC_inc, PC_enOut);
    ...
module Decode(clk, reset, halfInst, loadL, loadH, opcode, indexX, indexY, valueM);
    ...
module Memory(clk, reset, inData, outData, addr, enOut, load, enAddr, outAddr);
    ...
module IOports(clk, reset, inData, outData, load, addr, enOut, portOutput, portInput, portInterrupt);
    ...
module InterruptController(clk, reset, portInterrupt, intReq, intAck);
    ...
module ControlUnit(clk, reset, RB_enIn, RB_enOut, RB_selIndexIn, RB_selIndexOut, RB_setSP,
                   MM_enOut, MM_load, MM_enAddr, ALU_enA, ALU_enB, ALU_enOut, ALU_opW,
                   ALU_OP, PC_load, PC_inc, PC_enOut, DE_enOutImm, DE_loadL, DE_loadH,
                   ALU_flags, DE_opcode, IC_intReq, IC_intAck);
```

A continuación vamos a estudiarlos uno a uno.

Procesador OrgaSmall: module ArithmeticLogicUnit

```
module ArithmeticLogicUnit(clk, reset,
    A, B, O, enA, enB, enOut, OP,
    shift, flags, opW);
    input  clk, reset;
    input  [7:0] A, B;
    output [7:0] O;
    input  enA, enB, enOut;
    input  [3:0] OP;
    input  [2:0] shift;
    output [4:0] flags;
    input  opW;

    reg [8:0] qA, qB, qO;
    reg fI, fO, fN, fZ, fC;

    initial begin
        qA <= 0; qB <= 0; qO <= 0;
        fO <= 0; fN <= 0;
        fZ <= 0; fC <= 0;
        fI <= 0;
    end
    ...
```

Inicialización de todos los registros.
Registros de *flags* y de entrada y salida.

```
...
always @(posedge clk) begin
  case(OP)
    4'b0000 : begin qO <= qO;                                  end
    4'b0001 : begin qO <= qA + qB;                             end
    4'b0010 : begin qO <= qA + qB + {8'h0, fC};                end
    4'b0011 : begin qO <= qA - qB;                             end
    4'b0100 : begin qO <= qA & qB;                             end
    4'b0101 : begin qO <= qA | qB;                             end
    4'b0110 : begin qO <= qA ^ qB;                             end
    4'b0111 : begin qO <= {qA[7],qA[7:0]} >>> shift;           end
    4'b1000 : begin qO <= qA[7:0] >> shift;                    end
    4'b1001 : begin qO <= qA[7:0] << shift;                    end
    4'b1010 : begin qO <= {3'b000, fI, fO, fN, fZ, fC};        end
    4'b1100 : begin qO <= 9'h000; end // cte 00
    4'b1101 : begin qO <= 9'h001; end // cte 01
    4'b1110 : begin qO <= 9'h002; end // cte 02
    4'b1111 : begin qO <= 9'h0FF; end // cte ff
     default: begin qO <= 9'h000; end
  endcase
end
...
```

Los resultados de las operaciones se generan en 9 bits.
El cambio de q0 es en posedge (flanco ascendente).

Procesador OrgaSmall: module ArithmeticLogicUnit

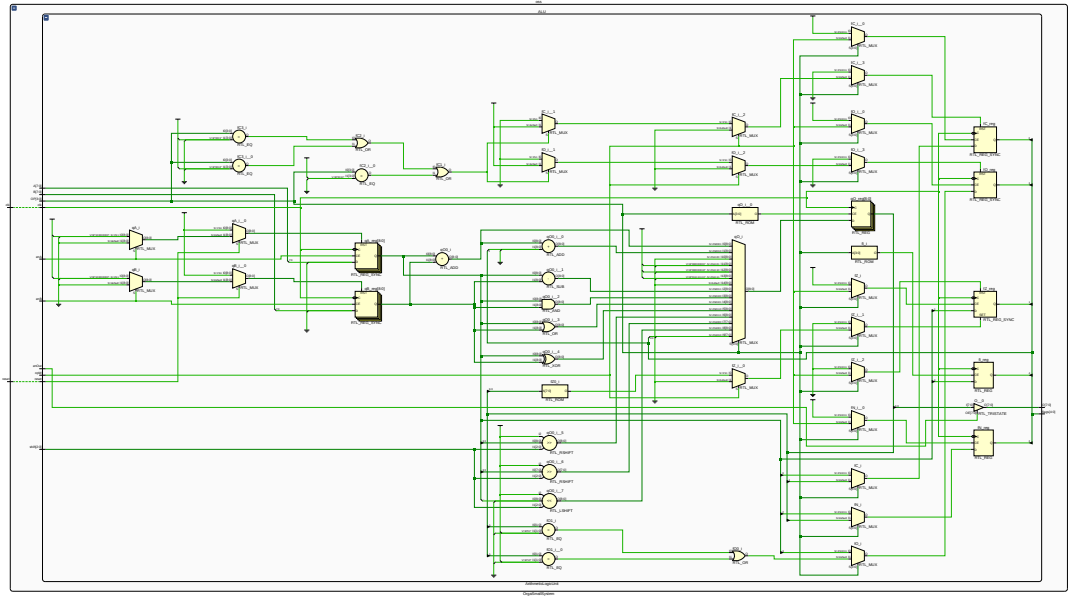
```
always @(negedge clk) begin
  if(enA) qA <= {1'h0,A};
  if(enB) qB <= {1'h0,B};
  if(reset) begin
    qA <= 0;
    qB <= 0;
  end
  if(opW) begin
    fN <= qO[7];
    fZ <= (qO[7:0] == 8'h0)? 1:0;
    if (OP==4'b0001 | OP==4'b0010 | OP==4'b0011)
      begin
          fC <= qO[8];
          fO <= (qO[8:7]==2'b01 || qO[8:7]==2'b10)? 1:0;
      end
    else
      begin
          fC <= 0;
          fO <= 0;
      end
  end
  if(OP==4'b1011) begin
    fI <= qA[4];
    fO <= qA[3];
    fN <= qA[2];
    fZ <= qA[1];
    fC <= qA[0];
  end
end
```

```
    ...

    assign flags = {fI, fO, fN, fZ, fC};
    assign O = enOut? qO[7:0] : 'bz;

endmodule
```

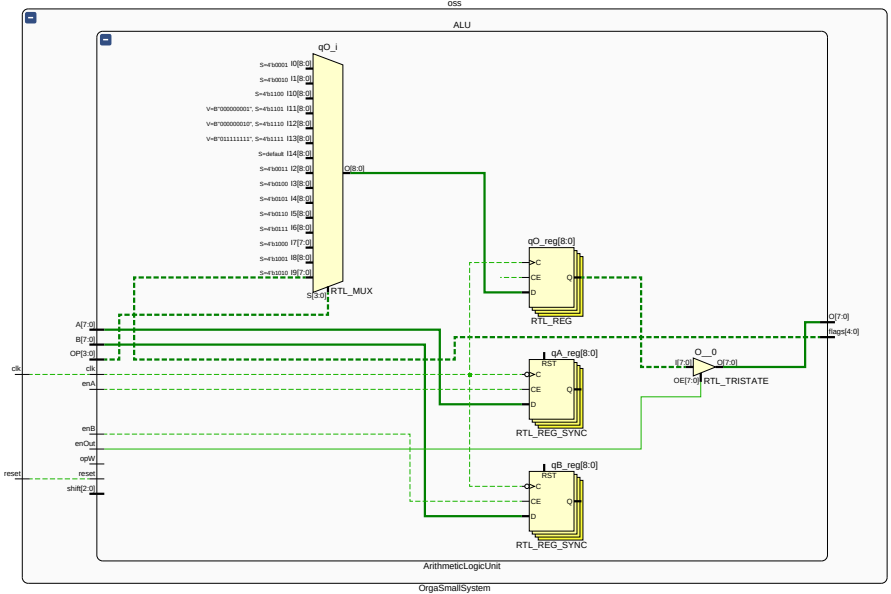
En flanco descendente se actualizan todos los estados de:

- Registros de entrada qA y qB, dependiendo de las señales enable.
- Los *flags* fO, fN, fZ y fC dependiendo del estado de la operación.
- Si la operación es 4, entonces se setean todos los *flags*.

Procesador OrgaSmall: module ArithmeticLogicUnit



Procesador OrgaSmall: module ArithmeticLogicUnit



Procesador OrgaSmall: module Registers

```
module Registers(clk, reset, inData, outData,
                 enIn, enOut, selIn, selOut, setSP);
    input  clk, reset;
    input  [7:0] inData;
    output [7:0] outData;
    input  enIn, enOut;
    input [2:0] selIn, selOut;
    input setSP;

    reg [7:0] q [0:7];

    initial begin
        q[0] <= 0; q[1] <= 0; q[2] <= 0; q[3] <= 0;
        q[4] <= 0; q[5] <= 0; q[6] <= 0; q[7] <= 0;
    end
    always @(negedge clk) begin
        if(enIn) begin
            if(setSP)
                q[7] <= inData;
            else
                q[selIn] <= inData;
        end
        if(reset) begin
            q[0] <= 0; q[1] <= 0; q[2] <= 0; q[3] <= 0;
            q[4] <= 0; q[5] <= 0; q[6] <= 0; q[7] <= 0;
        end
    end
    assign outData = enOut? (setSP? q[7] : q[selOut]) : 'bz;
endmodule
```

Los registros se declaran como un arreglo de registros, `q[0:7]` de tipo `reg [7:0]`

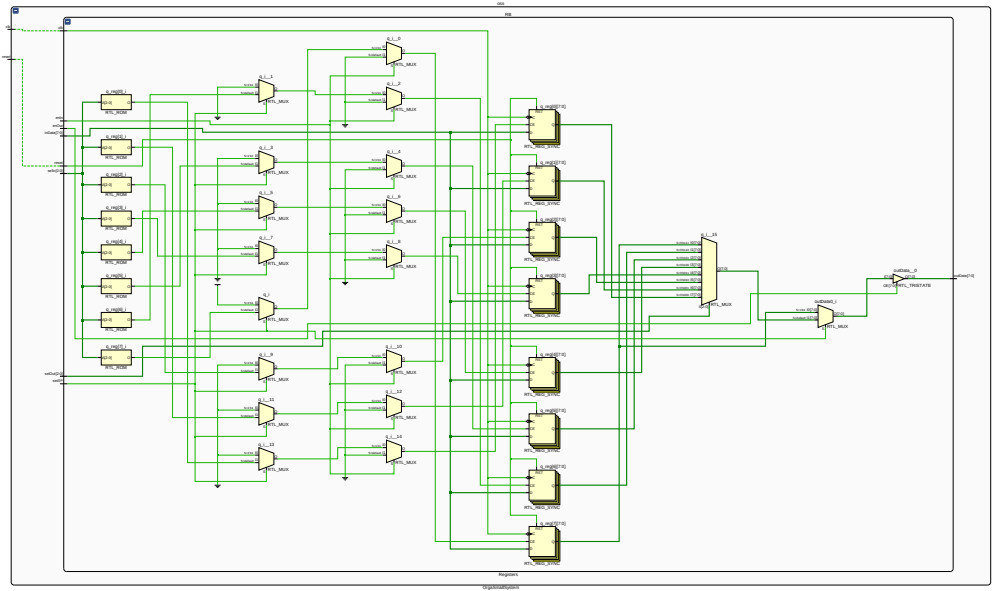
Inicialmente se setean todos los registros a cero.

La señal `setSP` sirve para setear el registro `r7`, usado como *stack pointer*.

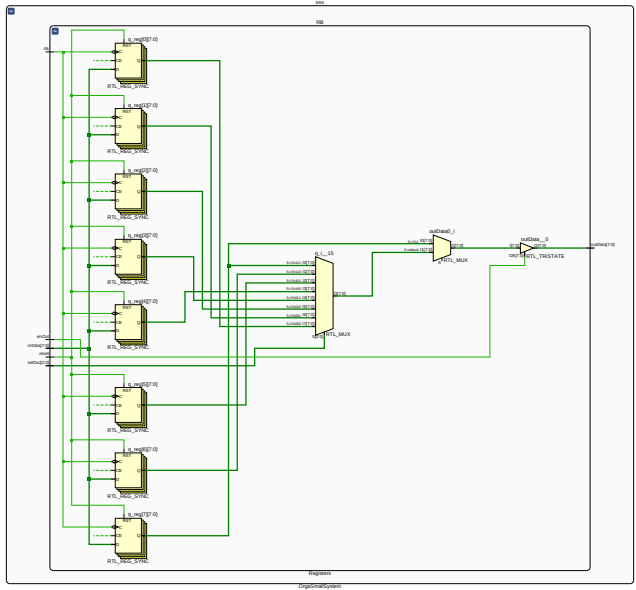
Si esta señal no está activa se utiliza `selIn` para seleccionar el registro a escribir.

La salida utiliza también la señal `setSP` o `selOut` según corresponda.

Procesador OrgaSmall: module Registers



Procesador OrgaSmall: module Registers



Procesador OrgaSmall: module ProgramCounter

```
module ProgramCounter(clk, reset, inValue, outValue,
                      PC_load, PC_inc, PC_enOut);

    input  clk, reset;
    input  [7:0] inValue;
    output [7:0] outValue;
    input  PC_load, PC_inc, PC_enOut;

    reg [7:0] q;

    initial begin
        q <= 'b0;
    end

    always @(negedge clk) begin
        if(reset)   q <= 'b0;
        if(PC_inc)  q <= q + 1;
        if(PC_load) q <= inValue;
    end

    assign outValue = PC_enOut? q : 'bz;

endmodule
```

El PC es un registro declarado dentro del módulo como q. Su tamaño es de 8 bits.

Inicialmente se setea el registro a cero.

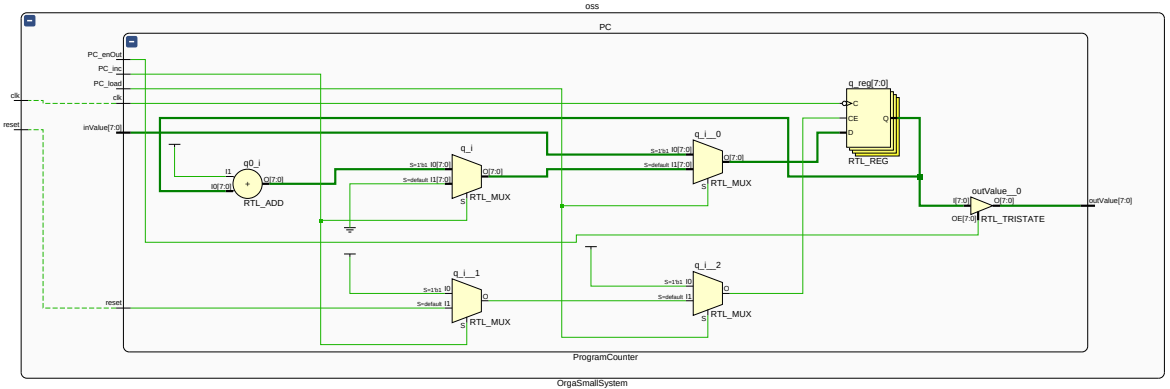
Tres señales modifican su estado:

- `reset`: Reset a cero.
- `PC_inc`: Incrementar en 1.
- `PC_load`: Cargar un valor arbitrario.

La salida solo se expone en función del valor de `PC_enOut`.

Este componente puede contener la lógica para incrementar de manera relativa el PC.
Incluso debe prover el valor del PC para poder ser utilizado en calculo de direcciones.

Procesador OrgaSmall: module ProgramCounter



Procesador OrgaSmall: module Decode

```
module Decode(clk, reset, halfInst, loadL, loadH,
              opcode, indexX, indexY, valueM);

    input  clk, reset;
    input [7:0] halfInst;
    input loadL, loadH;
    output [4:0] opcode;
    output [2:0] indexX, indexY;
    output [7:0] valueM;

    reg [15:0] q;

    initial begin
        q <= 'b0;
    end

    always @(negedge clk) begin
        if(loadL) q[7:0]  <= halfInst;
        if(loadH) q[15:8] <= halfInst;
        if(reset) q       <= 'b0;
    end

    //[15 14 13 12 11] [10 9 8] [7 6 5] [4 3 2 1 0]
    assign opcode = q[15:11];
    assign indexX = q[10:8];
    assign indexY = q[7:5];
    assign valueM = q[7:0];

endmodule
```

La decodificación debe tomar **dos datos de memoria** para armar una instrucción.

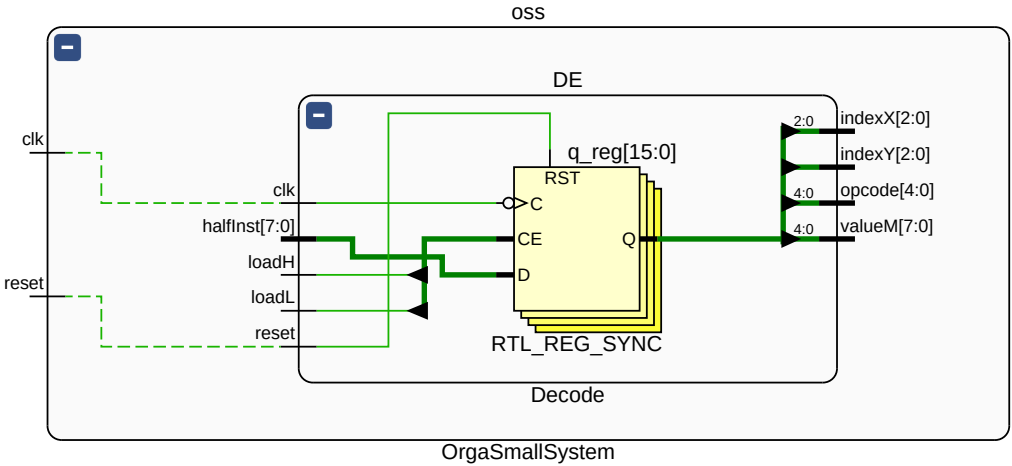
Se utiliza un solo registro de 16 bits que **se carga parcialmente**.

La decodificación consiste tan solo en tomar los bits de la instrucción según el **formato de instrucción**.

Notar que `valueM` se pisa con `indexY`.

En otros casos, se puede requerir que algunos campos se generen de forma condicional. Agregando lógica combinatoria en la decodificación.

Procesador OrgaSmall: module Decode



Procesador OrgaSmall: module Memory

```
module Memory(clk, reset, inData, outData, addr,
              enOut, load, enAddr, outAddr);
    input  clk, reset;
    input  [7:0] inData;
    output [7:0] outData;
    input  [7:0] addr;
    input  enOut, load, enAddr;
    output [7:0] outAddr;

    reg [7:0] mem_addr;
    reg [7:0] mem [0:255];

    initial begin
        mem_addr <= 'b0;
        $readmemh("test00Verilog.mem", mem);
    end

    always @(negedge clk) begin
        if(load & (mem_addr!=8'hfc & mem_addr!=8'hfd
           & mem_addr!=8'hfe)) mem[mem_addr] <= inData;
        if(enAddr) mem_addr <= addr;
        if(reset)  mem_addr <= 'b0;
    end

    assign outData = (enOut & (mem_addr!=8'hfc & mem_addr!=8'hfd
                      & mem_addr!=8'hfe))? mem[mem_addr] : 'bz;
    assign outAddr = mem_addr;

endmodule
```

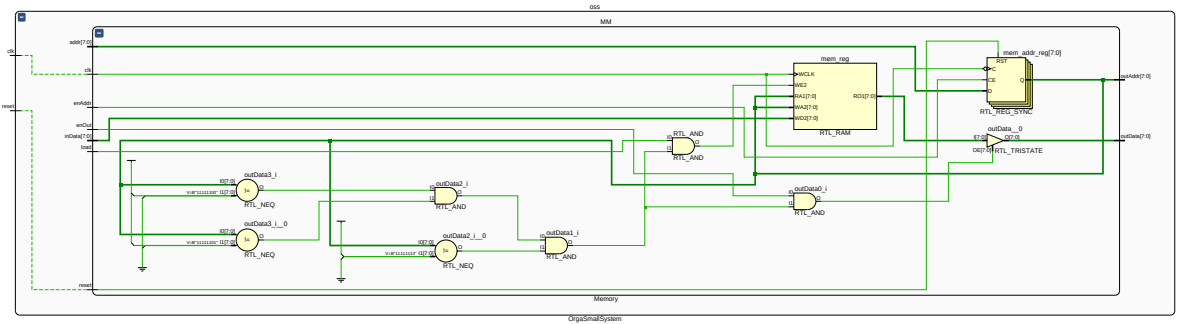
La memoria se declara como un **arreglo de 256 registros de 8 bits.** `reg [7:0] mem[0:255]`

La inicialización de la memoria se carga de un archivo utilizando la primitiva `$readmemh`.

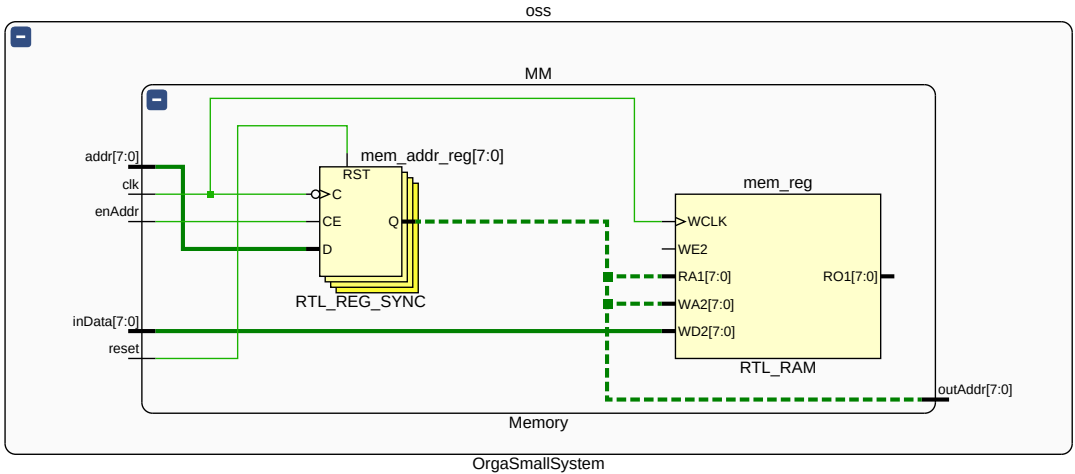
Tiene además un registro de dirección que se **expone al exterior** del modulo mediante `outAddr`.

La memoria solo se lee o escribe para direcciones que **no estén mapeadas a entrada/salida**.

Procesador OrgaSmall: module Memory



Procesador OrgaSmall: module Memory



Procesador OrgaSmall: module IOports

```
module IOports(clk, reset, inData, outData,
               load, addr, enOut, portOutput,
               portInput, portInterrupt);
    input  clk, reset;
    input  [7:0] inData;
    output [7:0] outData;
    input  load;
    input  [7:0] addr;
    input  enOut;
    output [7:0] portOutput;
    input  [7:0] portInput;
    input  [7:0] portInterrupt;

    reg [7:0] qPOutput;
    reg [7:0] qPInput;
    reg [7:0] qPInterrupt;

    initial begin
        qPortOutput <= 8'h00;
        qPortInput <= 8'h00;
        qPortInterrupt <= 8'h00;
    end
    ...
```

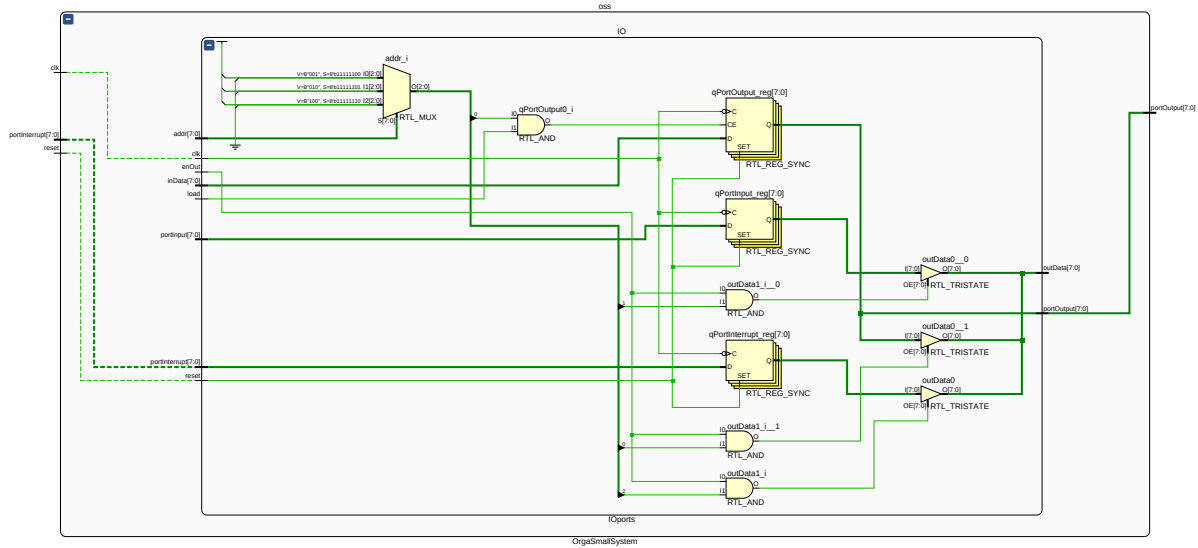
Sean puertos de entrada o salida, todos están declarados como **registros**.

```
    ...
    always @(negedge clk) begin
        if(addr==8'hfc && load)
            qPOutput <= inData;
        qPInput <= portInput;
        qPInterrupt <= portInterrupt;

        if(reset) begin
            qPOutput <= 8'hFF;
            qPInput <= 8'hFF;
            qPInterrupt <= 8'hFF;
        end
    end

    assign outData = (enOut & addr==8'hfc)? qPOutput   :'bz;
    assign outData = (enOut & addr==8'hfd)? qPInput    :'bz;
    assign outData = (enOut & addr==8'hfe)? qPInterrupt:'bz;
    assign portOutput = qPOutput;

endmodule
```

El puerto de salida es el único que se escribe por una señal.
El resto se **muestrean** sincrónicamente en cada *clock*.

La salida se expone si addr tiene el valor correspondiente a la dirección mapeada del puerto.

Procesador OrgaSmall: module IOports



Procesador OrgaSmall: module InterruptController

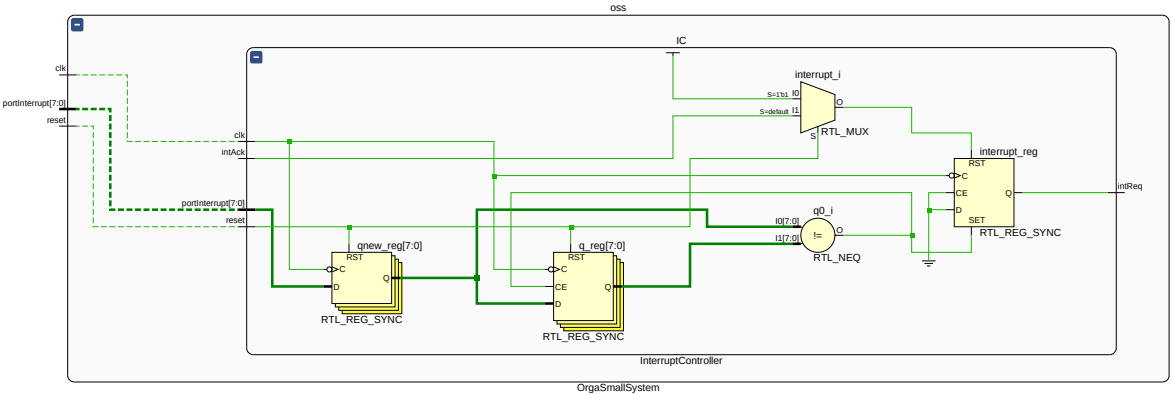
```
module InterruptController(clk, reset,
                           portInterrupt,
                           intReq, intAck);
    input  clk, reset;
    input  [7:0] portInterrupt;
    output intReq;
    input  intAck;

    reg [7:0] qnew;
    reg [7:0] q;
    reg interrupt;

    initial begin
        q <= 8'h00;
        qnew <= 8'h00;
        interrupt <= 0;
    end
    ...
```

```
    ...
    always @(negedge clk) begin
        qnew <= portInterrupt;
        if(qnew != q) begin
            q <= qnew;
            interrupt <= 1;
        end
        if(intAck) interrupt <= 0;
        if(reset) begin
            q <= 8'h00;
            qnew <= 8'h00;
            interrupt <= 0;
        end
    end

    assign intReq = interrupt;

endmodule
```

El objetivo es reconocer si el puerto de interrupciones cambio su valor, entonces se levantará la señal de interrupciones. Para esto se guarda el valor anterior y el actual, **y se los compara**.

La señal de interrupción se **debe reiniciar** indicando que la interrupción fue atendida.

Procesador OrgaSmall: module InterruptController



Procesador OrgaSmall: module ControlUnit

```
module ControlUnit(clk, reset,
    RB_enIn, RB_enOut, RB_selIndexIn, RB_selIndexOut, RB_setSP,
    MM_enOut, MM_load, MM_enAddr,
    ALU_enA, ALU_enB, ALU_enOut, ALU_opW, ALU_OP, ALU_flags,
    PC_load, PC_inc, PC_enOut,
    DE_enOutImm, DE_loadL, DE_loadH, DE_opcode,
    IC_intReq, IC_intAck);

    input  clk, reset;
    output RB_enIn, RB_enOut, RB_selIndexIn, RB_selIndexOut, RB_setSP;
    output MM_enOut, MM_load, MM_enAddr;
    output ALU_enA, ALU_enB, ALU_enOut, ALU_opW;
    output [3:0] ALU_OP;
    output PC_load, PC_inc, PC_enOut;
    output DE_enOutImm, DE_loadL, DE_loadH;
    input  [4:0] ALU_flags;
    input  [4:0] DE_opcode;
    input  IC_intReq;
    output IC_intAck;

    wire JC_microOp, JZ_microOp, JN_microOp, JO_microOp;
    wire load_int_microOp, load_microOp, reset_microOp;
    reg [31:0] rom [0:511];
    reg [8:0] microOp = 0;

    initial begin
        $readmemh("microCodeVerilog.mem", rom);
    end
    ...
```

Este módulo contiene una gran memoria para estados.

Por diseño se tienen 512 estados con 32 señales cada uno. Aunque la mayor parte no se utiliza.

Los estados se carga inicialmente de un archivo que contiene el conjunto de microinstrucciones o señales, que programan la máquina Orgasmall.

Alterando estas señales es posible contruir diferentes instrucciones.

Procesador OrgaSmall: module ControlUnit

```
...
always @(negedge clk) begin
    if(load_microOp & JC_microOp & ALU_flags[0])
       microOp <= microOp + 2;
    else
    if(load_microOp & JZ_microOp & ALU_flags[1])
        microOp <= microOp + 2;
    else
    if(load_microOp & JN_microOp & ALU_flags[2])
        microOp <= microOp + 2;
    else
    if(load_microOp & JO_microOp & ALU_flags[3])
        microOp <= microOp + 2;
    else
    if(load_microOp & load_int_microOp & ALU_flags[4] & IC_intReq)
        microOp <= 9'h1f0;
    else
    if(load_microOp & !JC_microOp & !JZ_microOp
       & !JN_microOp & !JO_microOp & !load_int_microOp)
        microOp <= { DE_opcode, 4'b0000 };
    else
        microOp <= microOp + 1;

    if(reset | reset_microOp) microOp <= 0;
end
...
```

El microPC o contador de estados, siempre se incrementa en 1.

A excepción de tres situaciones.

- Operaciones de salto condicional.
- Salto a la rutina de interrupciones.
- Salto a la instrucción decodificada.

Procesador OrgaSmall: module ControlUnit

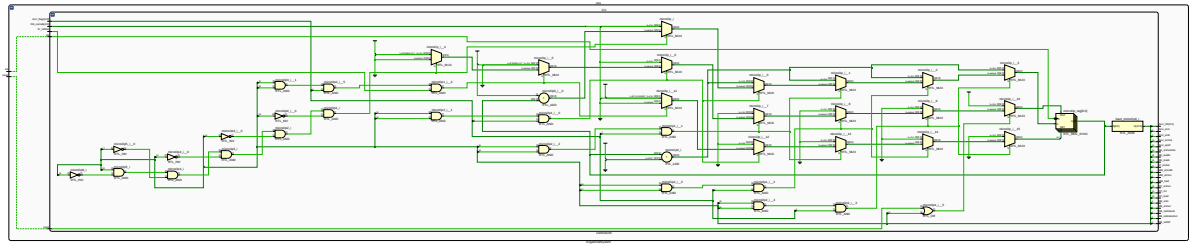
```
    ...
    assign RB_enIn          = rom[microOp][0];
    assign RB_enOut         = rom[microOp][1];
    assign RB_selIndexIn    = rom[microOp][2];
    assign RB_selIndexOut   = rom[microOp][3];
    assign RB_setSP         = rom[microOp][4];

    assign MM_enOut         = rom[microOp][5];
    assign MM_load          = rom[microOp][6];
    assign MM_enAddr        = rom[microOp][7];

    assign ALU_enA          = rom[microOp][8];
    assign ALU_enB          = rom[microOp][9];
    assign ALU_enOut        = rom[microOp][10];
    assign ALU_opW          = rom[microOp][11];

    assign ALU_OP           = { rom[microOp][15],
                                rom[microOp][14],
                                rom[microOp][13],
                                rom[microOp][12] };

    assign JC_microOp       = rom[microOp][16];
    assign JZ_microOp       = rom[microOp][17];
    assign JN_microOp       = rom[microOp][18];
    assign JO_microOp       = rom[microOp][19];
    ...
```

```
    ...
    assign PC_load          = rom[microOp][20];
    assign PC_inc           = rom[microOp][21];
    assign PC_enOut         = rom[microOp][22];
                            // rom[microOp][23];

    assign DE_enOutImm      = rom[microOp][24];
    assign DE_loadL         = rom[microOp][25];
    assign DE_loadH         = rom[microOp][26];
    assign IC_intAck        = rom[microOp][27];
                            // rom[microOp][28];

    assign load_int_microOp = rom[microOp][29];
    assign load_microOp     = rom[microOp][30];
    assign reset_microOp    = rom[microOp][31];

endmodule
```

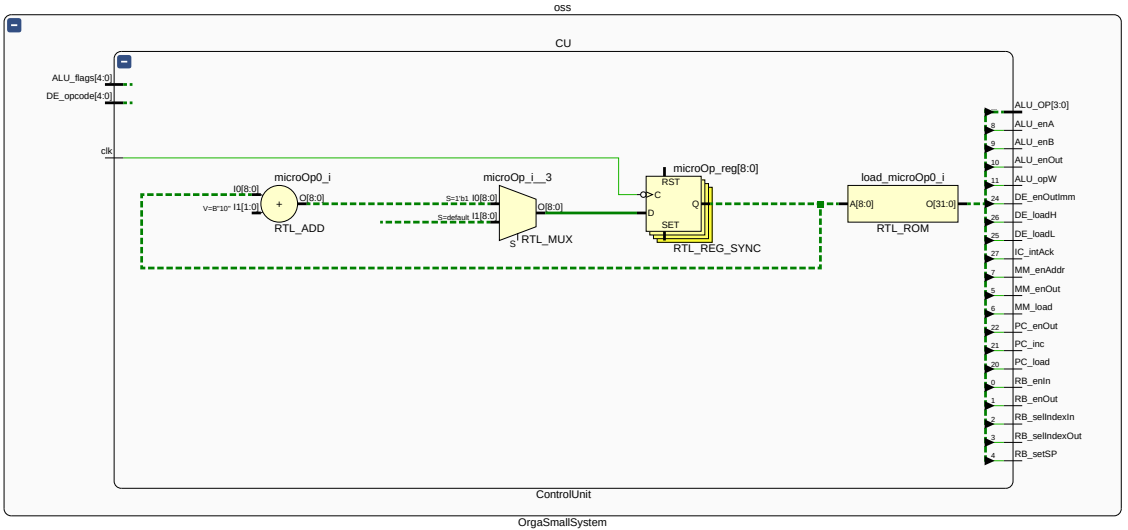
Todas las señales leídas de la memoria son asignadas a cables o registros del circuito.

Luego son utilizadas fuera y dentro del circuito.

Procesador OrgaSmall: module ControlUnit



Procesador OrgaSmall: module ControlUnit



Procesador OrgaSmall: module OrgaSmallSystem

```
module OrgaSmallSystem(clk, reset, portOutput,
                       portInput, portInterrupt);

    input clk, reset;

    output [7:0] portOutput;
    input  [7:0] portInput;
    input  [7:0] portInterrupt;

    wire [7:0] BUS;
    wire ALU_enA, ALU_enB, ALU_enOut, ALU_opW;
    wire [3:0] ALU_OP;
    wire [4:0] ALU_flags;
    wire RB_enIn, RB_enOut, RB_setSP;
    wire [2:0] RB_selIn, RB_selOut;
    wire PC_load, PC_inc, PC_enOut;
    wire DE_enOutImm, DE_loadL, DE_loadH;
    wire [4:0] DE_opcode;
    wire MM_enOut, MM_load, MM_enAddr;
    wire [2:0] DE_indexX, DE_indexY;
    wire [7:0] DE_valueM;
    wire [7:0] outAddr;
    wire IC_intReq, IC_intAck;
    wire RB_selIndexIn, RB_selIndexOut;
    ...
```

Declaración de todos los cables requeridos.

```
...
ArithmeticLogicUnit ALU(clk, reset, BUS, BUS, BUS,
  ALU_enA, ALU_enB, ALU_enOut, ALU_OP, DE_indexY,
  ALU_flags, ALU_opW);

Registers RB(clk, reset, BUS, BUS, RB_enIn,
  RB_enOut, RB_selIn, RB_selOut, RB_setSP);

ProgramCounter PC(clk, reset, BUS, BUS,
  PC_load, PC_inc, PC_enOut);

Decode DE(clk, reset, BUS, DE_loadL, DE_loadH,
  DE_opcode, DE_indexX, DE_indexY, DE_valueM);

Memory MM(clk, reset, BUS, BUS, BUS,
  MM_enOut, MM_load, MM_enAddr, outAddr);

IOports IO(clk, reset, BUS, BUS, MM_load, outAddr,
  MM_enOut, portOutput, portInput, portInterrupt);

InterruptController IC(clk, reset, portInterrupt,
  IC_intReq, IC_intAck);
...
```

Se instancia uno a uno los componentes del *datapath*.

Se respeta el orden de los parámetros, aunque se dificulta la lectura. **Ejemplo: señal BUS duplicada.**

Procesador OrgaSmall: module OrgaSmallSystem

```
    ...
    ControlUnit CU(clk, reset,
        RB_enIn, RB_enOut, RB_selIndexIn, RB_selIndexOut, RB_setSP,
        MM_enOut, MM_load, MM_enAddr,
        ALU_enA, ALU_enB, ALU_enOut, ALU_opW, ALU_OP, ALU_flags,
        PC_load, PC_inc, PC_enOut,
        DE_enOutImm, DE_loadL, DE_loadH, DE_opcode,
        IC_intReq, IC_intAck);

    assign RB_selIn  = RB_selIndexIn?  DE_indexY : DE_indexX;
    assign RB_selOut = RB_selIndexOut? DE_indexY : DE_indexX;
    assign BUS       = DE_enOutImm?    DE_valueM : 'bz;

endmodule
```

Por último se instancia la únidad de control.

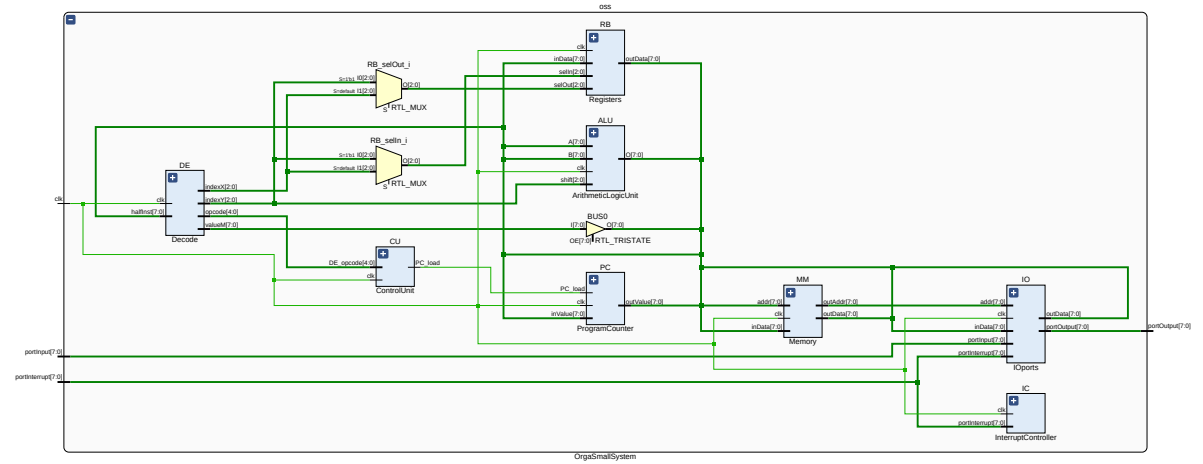
Donde se conecta el resto de las señales declaradas como cables.

El *datapath* se completa generando las señales desde el decodificador a la selección de registros en el banco de registros. Se identifica como los dos multiplexores de selección de registros.

Además generando la señal desde el decodificador al bus principal del valor `M`. Leer el valor inmediato.

Notar que estos elementos podian formar parte de los módulos, sin embargo se busco respetar el esquema del *datapath*.

Procesador OrgaSmall: module OrgaSmallSystem



Ejemplo Completo: Contador con salida serie

```
SET R7, 0xFB  ; set STACK
SET R0, interrupt_handler
STR  [0xFF], R0 ; Set Interrupt Rutine
SET R0, 0x10
LOADF R0 ; Set Interrupt Flag

; main
SET R0, 0x0
SET R1, 0x1
whileTrue:
    STR [0xFC], R1 ; out <- 0001
    CALL |R7|, sleep
    STR [0xFC], R0 ; out <- 0000
    SET R3, 0x7
    whileSerie:
        CALL |R7|, sleep
        LOAD R2, [current]
        AND R2, R1
        SHL R2, 1
        STR [0xFC], R2 ; out <- 00X0
        LOAD R2, [current]
        SHR R2, 1
        STR [current], R2
        SUB R3, R1
        JZ continuar
        JMP whileSerie
    continuar:
    LOAD R2, [data]
    STR [current], R2
JMP whileTrue
```

```
sleep:
    PUSH |R7|, R0
    PUSH |R7|, R1
    PUSH |R7|, R2
    PUSH |R7|, R3
    PUSH |R7|, R4
    SET R2, 10
    ciclo:
        CMP R2, R0
        JZ end_sleep
        SUB R2, R1
        JMP ciclo
    end_sleep:
    POP  |R7|, R4
    POP  |R7|, R3
    POP  |R7|, R2
    POP  |R7|, R1
    POP  |R7|, R0
    RET  |R7|

current:  DB 0x00
data:     DB 0x00

; Memory
; 0x00 = entry point
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Rutine pointer
```

```
interrupt_handler:
    PUSH |R7|, R0
    PUSH |R7|, R1
    PUSH |R7|, R2
    PUSH |R7|, R3
    PUSH |R7|, R4
    SET R0, 0x1
    SET R1, 0x2
    LOAD R3, [0xFE]
    CMP R3, R0
    JZ val_inc
    CMP R3, R1
    JZ val_dec
    end_interrupt:
    POP  |R7|, R4
    POP  |R7|, R3
    POP  |R7|, R2
    POP  |R7|, R1
    POP  |R7|, R0
    RETI |R7|
    val_inc:
        LOAD R3, [data]
        ADD R3, R0
        STR [data], R3
        JMP end_interrupt
    val_dec:
        LOAD R3, [data]
        SUB R3, R0
        STR [data], R3
        JMP end_interrupt
```

Observaciones `OrgaSmall`

La arquitectura del sistema como su microarquitectura están **diseñadas con fines didácticos**.

Su espacio de direccionamiento y tipos de direccionamiento son **muy limitados**.

No está pensado para soportar desplazamientos a memoria, todas **direcciones absolutas**.

El diseño de las instrucciones es regular para simplificar **su lectura y decodificación.**

Soporta instrucciones complejas, ya que **no tiene seudoinstrucciones**.

El diseño de la microarquitectura tiene múltiples **registros innecesarios**.

Es un diseño escalable y adaptable para **nuevas instrucciones**.

Simple de modificar y agregar **nuevos componentes**.

No esta pensado para ser **eficiente**, incluso ni para ser implementado en un FPGA.

Bibliografía

- **“Arquitectura OrgaSmall”**
  `https://github.com/fokerman/microOrgaSmall`

¡Gracias!