

# **SIMULACION DE UN PIC16 EN VHDL**

**Centro de Investigación en Computación  
Instituto Politécnico Nacional**

**Santiago Villafuerte  
Diciembre 2008**

## **Introducción**

Los microcontroladores PIC16 de la compañía Microchip son microprocesadores que cuentan con una unidad lógica aritmética (ALU) de procesamiento de 8 bits, memoria de programa interna de hasta 8k instrucciones y memoria RAM interna de hasta 368 bytes. Son capaces de controlar dispositivos externos como LED's sin necesidad de contar con etapas de potencia adicionales. Su memoria de programa es del tipo FLASH, siendo así borrable y reprogramable sin necesidad de hardware especial. Los PIC16 cuentan con puertos de entrada-salida que pueden operar como simples terminales TTL o incluso funcionar como puertos seriales RS232.

Las siglas PIC significan Peripheral Interface Controller y son llamados microcontroladores porque en un principio fueron diseñados como etapas de control en circuitos digitales y ya incluían todo un sistema mínimo en un solo circuito integrado.

La arquitectura de memoria de estos microcontroladores es del tipo Harvard, lo cual permite que los micros puedan acceder a su memoria de programa y a la memoria de datos de forma simultánea.

## **Objetivo del proyecto**

Describir la arquitectura de un PIC16 mediante VHDL.

La descripción del PIC16 solo se realizará sobre la arquitectura del núcleo, incluyendo: decodificación de códigos de operación, generación de direcciones de memoria RAM, simulación de los bancos de memoria, simulación de la ALU, generación de la dirección de programa y simulación de la memoria EEPROM.

Los demás periféricos, tales como puertos de uso general, temporizadores, puertos seriales y paralelos, módulos de captura y comparación, no serán simulados en este proyecto aunque pueden simularse sin problema alguno.

La descripción y simulación del núcleo del PIC16 se realizará con el software Orcad 9 Capture CIS. La simulación se ejecutará en diseño y no en tiempos.

## **Entidades que conforman el núcleo de un PIC16**

El núcleo de un PIC16 está conformado como se ve en la figura 1. Esta arquitectura contempla las siguientes entidades:

- Oscilador
- Decodificador y control
- Generador de direcciones
- RAM

- ALU
- Contador de programa
- EEPROM

A continuación se describe el diseño en VHDL de cada una de ellas así como sus funciones específicas.

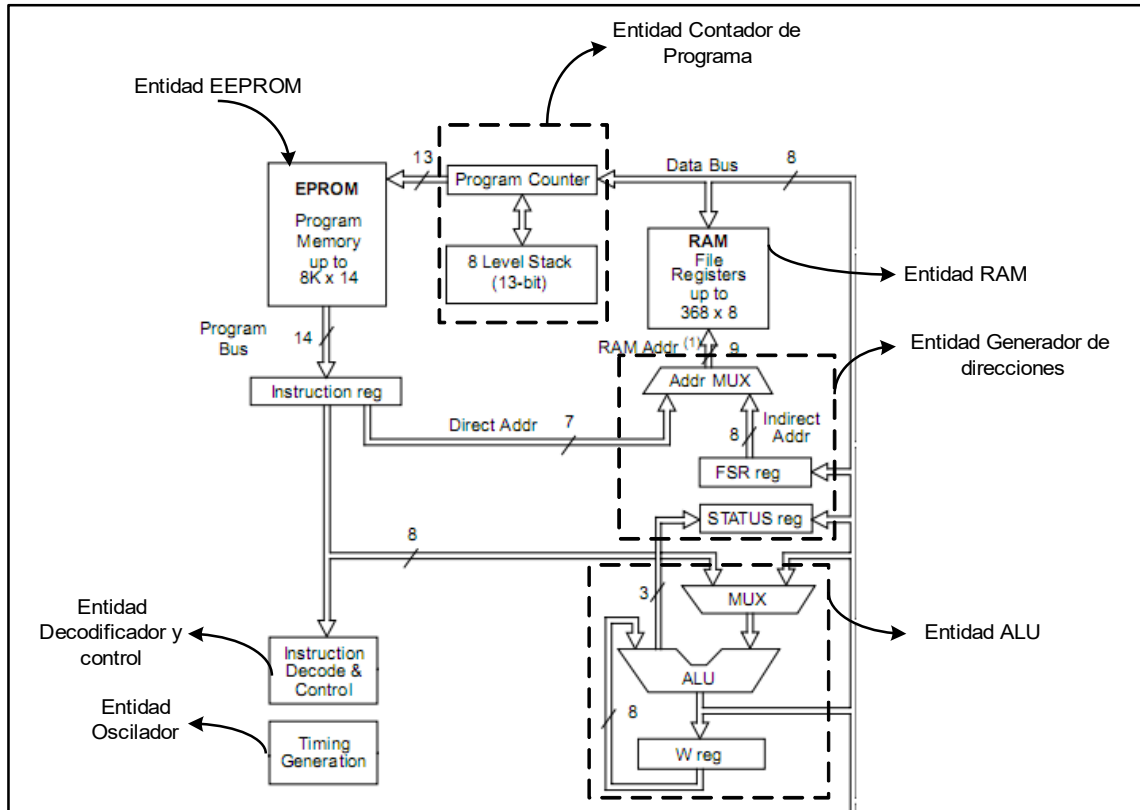


Figura 1. Arquitectura del núcleo de un PIC16.

### Entidad Oscilador

La entidad Oscilador se muestra en la figura 2. El PIC16 trabaja con un oscilador de hasta 20MHz pero internamente se divide esa frecuencia entre 4 para obtener los 4 ciclos de trabajo con los que las diferentes entidades operan.

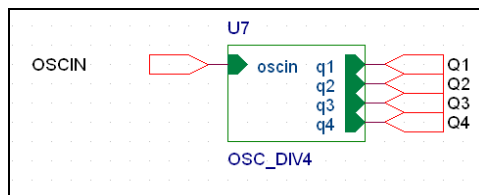


Figura 2. Entidad Oscilador.

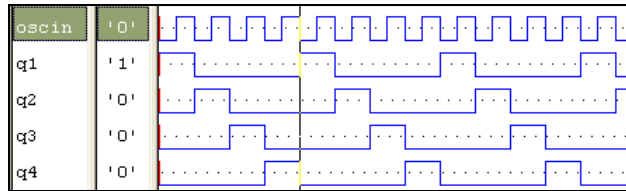


Figura 3. División entre 4 de OSCIN.

Las señales Q solo estarán en estado alto un tiempo igual al periodo de la señal OSCIN y durarán 3 periodos de OSCIN en estado bajo (ver figura 3). La operación de las señales Q se describe a continuación:

- Q1 está en alto en el ciclo 1 de OSCIN
- Q2 está en alto en el ciclo 2 de OSCIN
- Q3 está en alto en el ciclo 3 de OSCIN
- Q4 está en alto en el ciclo 4 de OSCIN

Las demás entidades del núcleo regirán su operación de acuerdo a las señales Q o sus correspondientes retrasos.

La operación básica de las entidades con respecto a los tiempos Q es la siguiente:

- Q1. Se decodifica la instrucción a realizar.
- Q2. Se entrega en el bus de datos el dato a calcular.
- Q3. Se procesa el dato.
- Q4. Se guarda el dato.

Código VHDL:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; --operaciones aritméticas

entity osc_div4 is
port (oscin: in std_logic;
      q1,q2,q3,q4: out std_logic);
end osc_div4;

architecture comp of osc_div4 is
signal t: std_logic_vector(1 downto 0):="00";
begin
  process(oscin)
  begin
    case oscin is
    when '0' => t <= t + 1;
      if t="00" then q1 <= '1'; else q1 <= '0'; end if;
      if t="01" then q2 <= '1'; else q2 <= '0'; end if;
      if t="10" then q3 <= '1'; else q3 <= '0'; end if;
      if t="11" then q4 <= '1'; else q4 <= '0'; end if;
    end case;
  end process;
end comp;

```

## Entidad Decodificador y Control

La entidad decodificador y control tiene 2 funciones principales: decodificar las instrucciones que salen de la memoria de programa y controlar las acciones que lleva a cabo la ALU y el generador de direcciones. La entidad se muestra en la figura 4.

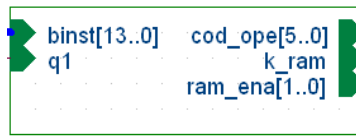


Figura 4. Entidad decodificador y control.

Cuenta con las siguientes entradas salidas:

**Entradas**

- binst. Recibe 14 bits que conforman el código de operación.
- q1. Recibe el pulso q1 con el que iniciará su operación.

**Salidas**

- cod\_ope. Entrega 6 bits que indican a la ALU qué operación lógica o aritmética ha de realizar.
- k\_ram. Indica a la ALU de dónde deberá tomar un dato para realizar su cálculo, lo puede tomar del código de operación (direccionamiento inmediato) o del bus de datos (direccionamiento directo o indirecto).
- ram\_ena. Entrega 2 bits a la entidad generador de direcciones.
  - Bit 1. Indica si un dato deberá ser escrito en el bus de direcciones en Q2.
  - Bit 0. Indica si un dato deberá ser leído desde el bus de direcciones en Q4.

La decodificación de los códigos de operación se hace primero revisando los 2 bits más significativos (MSb) de los 14 bits que tienen los códigos de operación de los PIC16.

- 00. Instrucciones orientadas a byte.
- 01. Instrucciones orientadas a bit.
- 10. Instrucciones de llamado o regreso de subrutina.
- 11. Instrucciones de direccionamiento inmediato.

Posteriormente se vuelve a hacer otra decodificación contemplando los 12 bits restantes del código de operación. La siguiente tabla muestra el código entregado en hexadecimal en la salida cod\_ope de acuerdo a la instrucción detectada.

Nemónico	cod_ope	Nemónico	cod_ope
<b>Instrucciones orientadas a byte</b>		<b>Instrucciones orientadas a bit</b>	
NOP	0	BCF	14
RETURN	1	BSF	15
RETFIE	2	BTFSC	16
MOVWF	3	BTFSS	17
CLRW	4	<b>Instrucciones de subrutina</b>	
CLRF	5	CALL	18
SUBWF	6	GOTO	19
DECF	7	<b>Instrucciones de direc. inmediato</b>	
IORWF	8	MOVLW	1A
ANDWF	9	RETLW	1B
XORWF	A	IORLW	1C
ADDWF	B	ANDLW	1D
MOVF	C	XORLW	1E
COMF	D	SUBLW	1F

INCF	E	ADDLW	20
DECFSZ	F		
RRF	10		
RLF	11		
SWAPF	12		
INCFSZ	13		

El dato que utilizará la ALU para realizar un cálculo se obtendrá del código de operación (literal k de 8 bits) o del bus de datos. El origen del dato siempre será k cuando la operación sea un direccionamiento inmediato y será el bus de datos en otros casos; k\_ram valdrá 0 cuando el dato sea una k y valdrá 1 cuando debe ser extraído del bus de datos.

Los 2 bits de salida ram\_ena le indicarán al generador de direcciones si deberá ordenar a la RAM o a otras entidades escribir el bus de datos o leerlo. El valor de ram\_ena dependerá del código de operación. En el caso de las instrucciones de direccionamiento inmediato el bus de datos nunca será escrito ni leído ya que el valor de entrada a la ALU vendrá desde la instrucción misma y el resultado se guardará en el registro de trabajo W.

Los valores de ram\_ena indican lo siguiente al generador de direcciones:

- 00 no escribe, no lee
- 01 no escribe, sí lee
- 10 sí escribe, no lee
- 11 sí escribe, sí lee

El generador de direcciones tendrá la tarea de indicar qué entidad trabajará sobre el bus de datos de acuerdo a lo que la entidad de decodificación indique.

A continuación se muestra el desempeño en tiempos de la entidad de decodificación, la cual se activa cuando q1 está en alto (ver figura 5).

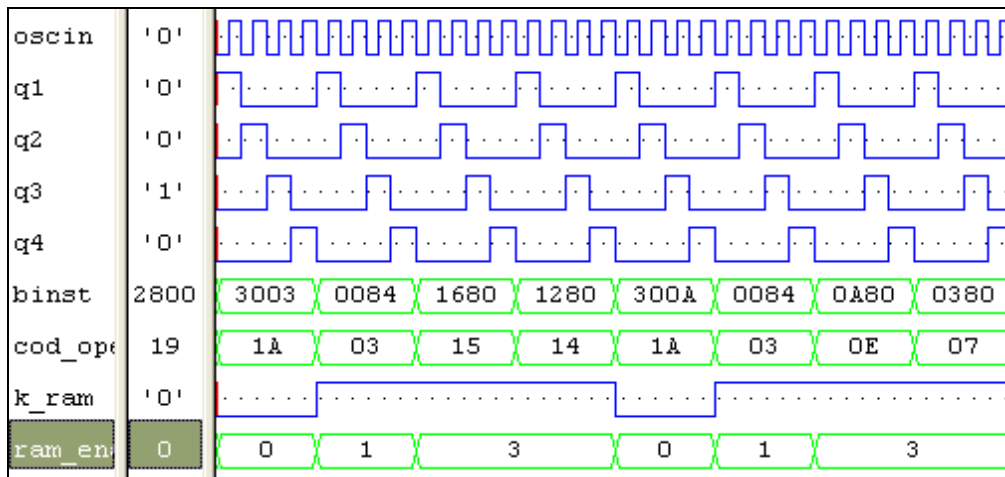


Figura 5. Diagrama de tiempos de la entidad Decodificador y Control.

Se observa que:

- 0x3003 es decodificado como MOVLW.
- 0x0084 es decodificado como MOVWF.
- 0x1680 es decodificado como BSF.

- 0x1280 es decodificado como BCF.
- 0x300A es decodificado como MOVLW.
- etc.

**Nota importante de la entidad:**

La entidad de decodificación no contempla las siguientes instrucciones del PIC: CLRWDT y SLEEP ya que son instrucciones que involucran el uso de periféricos como el temporizador y el perro guardián, las cuales fueron descartadas en la descripción del proyecto.

**Código VHDL:**

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

--Versión 2
--Genera el decodificador de instrucciones total
--Selecciona k o RAM para ALU
--Habilita o deshabilita el generador de direcciones

entity dec_tot is
port(binst: in std_logic_vector(13 downto 0);    --14 bits por inst.
     q1: in std_logic;                          --sensible a q1
     cod_ope: out std_logic_vector(5 downto 0); --2^6 oper = 33 oper
     k_ram: out std_logic;                      --selector de k o ram,
alu
     ram_ena: out std_logic_vector(1 downto 0)); --habilitador de ram (hacia gen_dir)
end dec_tot;

--Valores de ram_ena
--00 no escribe, no lee
--01 no escribe, si lee
--10 si escribe, no lee
--11 si escribe, si lee
--binst(7)=0 -> W
--binst(7)=1 -> F

architecture comp of dec_tot is
begin
  process(q1)
  begin
    if q1 = '1' then
      case binst(13 downto 12) is
        --Instrucciones con 00
        when "00" =>
          case (binst(11 downto 7) & binst(3) & binst(0)) is
            when "0000000" => cod_ope <= "000000"; ram_ena<="00"; --nop
            when "0000010" => cod_ope <= "000001"; ram_ena<="00"; --return
            when "0000011" => cod_ope <= "000010"; ram_ena<="00"; --retfie
          end case;
          case binst(11 downto 7) is
            when "00001" => cod_ope <= "000011"; k_ram <= '1'; ram_ena<="01"; --movwf
            when "00010" => cod_ope <= "000100"; ram_ena<="00"; --clrw
            when "00011" => cod_ope <= "000101"; k_ram <= '1'; ram_ena<="01"; --clrf
          end case;
          case binst(11 downto 8) is
            when "0010" => cod_ope <= "000110"; k_ram <= '1'; ram_ena<='1'&binst(7); --subwf
            when "0011" => cod_ope <= "000111"; k_ram <= '1'; ram_ena<='1'&binst(7); --decf
            when "0100" => cod_ope <= "001000"; k_ram <= '1'; ram_ena<='1'&binst(7); --iorwf
            when "0101" => cod_ope <= "001001"; k_ram <= '1'; ram_ena<='1'&binst(7); --andwf
            when "0110" => cod_ope <= "001010"; k_ram <= '1'; ram_ena<='1'&binst(7); --xorwf
            when "0111" => cod_ope <= "001011"; k_ram <= '1'; ram_ena<='1'&binst(7); --addwf
            when "1000" => cod_ope <= "001100"; k_ram <= '1'; ram_ena<='1'&binst(7); --movf
            when "1001" => cod_ope <= "001101"; k_ram <= '1'; ram_ena<='1'&binst(7); --comf
            when "1010" => cod_ope <= "001110"; k_ram <= '1'; ram_ena<='1'&binst(7); --incf
            when "1011" => cod_ope <= "001111"; k_ram <= '1'; ram_ena<='1'&binst(7); --decfsz
          end case;
        end case;
      end if;
    end process;
  end architecture;

```

```

when "1100" => cod_ope <= "010000"; k_ram <= '1'; ram_ena<='1'&binst(7); --rrf
when "1101" => cod_ope <= "010001"; k_ram <= '1'; ram_ena<='1'&binst(7); --rlf
when "1110" => cod_ope <= "010010"; k_ram <= '1'; ram_ena<='1'&binst(7); --swapf
when "1111" => cod_ope <= "010011"; k_ram <= '1'; ram_ena<='1'&binst(7); --incfsz
end case;
--Instrucciones con 01
when "01" =>
case binst(11 downto 10) is
when "00" => cod_ope <= "010100"; k_ram <= '1'; ram_ena<="11"; --bcf
when "01" => cod_ope <= "010101"; k_ram <= '1'; ram_ena<="11"; --bsf
when "10" => cod_ope <= "010110"; k_ram <= '1'; ram_ena<="10"; --btfsc
when "11" => cod_ope <= "010111"; k_ram <= '1'; ram_ena<="10"; --btfss
end case;
--Instrucciones con 10
when "10" =>
case binst(11) is
when '0' => cod_ope <= "011000"; k_ram <= '0'; ram_ena<="00"; --call
when '1' => cod_ope <= "011001"; k_ram <= '0'; ram_ena<="00"; --goto
end case;
--Instrucciones con 11
when "11" =>
case binst(11 downto 10) is
when "00" => cod_ope <= "011010"; k_ram <= '0'; ram_ena<="00"; --movlw
when "01" => cod_ope <= "011011"; k_ram <= '0'; ram_ena<="00"; --retlw
end case;
case binst(11 downto 8) is
when "1000" => cod_ope <= "011100"; k_ram <= '0'; ram_ena<="00"; --iorlw
when "1001" => cod_ope <= "011101"; k_ram <= '0'; ram_ena<="00"; --andlw
when "1010" => cod_ope <= "011110"; k_ram <= '0'; ram_ena<="00"; --xorlw
end case;
case binst(11 downto 9) is
when "110" => cod_ope <= "011111"; k_ram <= '0'; ram_ena<="00"; --sublw
when "111" => cod_ope <= "100000"; k_ram <= '0'; ram_ena<="00"; --addlw
end case;
end case;
end if;
end process;
end comp;

```

## Entidad Generador de direcciones

El generador de direcciones tiene la tarea de generar una dirección de memoria de 9 bits para que se entregue o se lea un dato del bus de datos. También es encargada de operar el contenido de los registros Status y FSR.

La memoria RAM de los PIC16 puede direccionar 512 localidades máximas de memoria, pero algunas localidades no están implementadas o están repetidas en otras localidades, teniendo solo 368bytes de memoria significativa. Para la descripción del PIC en este proyecto se considerará que la RAM es de 512 bytes para efectos de simplificación del código, accesados con 9 bits.

El PIC16 maneja 3 tipos de direccionamiento de memoria: directo, indirecto e inmediato. La entidad generador de direcciones es útil para los 2 primeros casos.

El direccionamiento directo recibe una dirección de memoria directamente de la instrucción actual. El direccionamiento indirecto recibe una dirección de memoria desde el contenido del registro FSR. El direccionamiento inmediato no recibe una dirección de memoria, recibe el dato a manipular directamente desde la instrucción actual.

Cuando se recibe un direccionamiento directo, los 7 bits menos significativos de la instrucción equivalen los 7 bits menos significativos de la dirección RAM a emplear y para completar los 9 bits de dirección se emplean 2 bits contenidos en el registro status (RP1:RP0).

Cuando se recibe un direccionamiento indirecto, los 8 bits menos significativos de la instrucción equivalen a los 8 bits del registro FSR y se completan 9 bits con un bit contenido en el registro status (IRP).

La operación de status y fsr es realizada en esta entidad ya que estos registros están mapeados en 4 localidades de memoria ram cada uno y esto implica dificultad en la simulación al tener que registrar cambios en 4 localidades de forma simultánea. Para ello se implementaron de forma interna los 2 registros y cuando una instrucción hace referencia a status o a fsr, el generador de direcciones desactiva la memoria ram y funge como una ram pequeña de 2 registros.

El generador también se encarga de controlar el contenido de los registros PCL y PCLATH que generan la dirección de memoria EEPROM y que también están mapeados 4 veces en la memoria RAM. Si una instrucción debe modificar a PCL o PCLATH, el generador desactiva la ram y permite que la entidad contador de programa tome control del bus de datos.

La entidad generador de direcciones se puede ver en la figura 6 y sus entradas y salidas se describen a continuación.

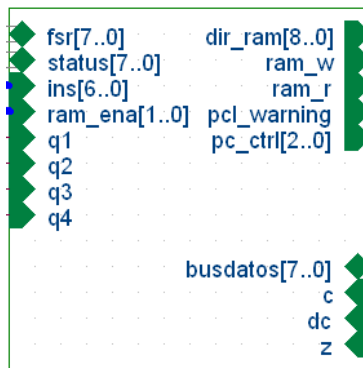


Figura 6. Entidad generador de direcciones.

#### Entradas

- ins. Recibe los 7 bits menos significativos de la instrucción.
- ram\_ena. Recibe las banderas de escritura en Q2 y lectura en Q4.
- q1, q2, q3 y q4. Entradas a las que es sensible esta entidad.
- z, dc, c. Bits del registro status que indican si hubo un resultado igual a cero, un acarreo de 4 bits o un acarreo de 9 bits.

#### Salidas

- dir\_ram. Salida de 9 bits de dirección de ram.
- ram\_w. Indica a la ram que debe escribir en el bus de datos.
- ram\_r. Indica a la ram que debe leer del bus de datos.
- pcl\_warning. Advertencia a la ALU de que el registro PCL podrá ser modificado.
- pc\_ctrl. Controla la escritura y lectura de los registros PCL y PCLATH en la entidad contador de programa.

#### Entrada/Salida

- busdatos. Escribe sobre o lee hacia el bus de datos el contenido de status o fsr.

En Q1 la entidad se encarga de poner el bus de datos en alta impedancia para evitar colisiones y genera dir\_ram de 9 bits. En Q2 escribe el contenido de status o fsr si es necesario. En Q3 pone el bus en alta impedancia para evitar colisión con el resultado que escribirá la ALU. En Q4 se copia el dato del bus en status o fsr si es necesario.

Los estados de pc\_ctrl son los siguientes:

- bit0 -> 0 no lee en Q2, 1 sí lee en Q2
- bit1 -> 0 no escribe en Q2, 1 sí escribe en Q2
- bit2 -> 0 es PCL, 1 es PCLATH

**Nota importante de la entidad:**

La entidad es capaz de detectar direccionamientos indirectos a status, fsr, pcl o pclath. En la práctica es poco común realizar direccionamientos en registros mapeados en los 4 bancos de memoria, pero con motivos de aprendizaje se implementó este caso.

**Código VHDL:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

--Genera la dirección de 9 bits para la ram
--Entrega o lee status o fsr si la dirección lo indica
--Habilita o deshabilita PCL y PCLATH cuando la dirección lo indica

--Versión 4
--Tiene salida pcl_warning para avisar a la alu que pcl se modifica
--Versión 3
--En Q3 ram_w y ram_r valen 0 (corrige problema de escritura de RAM)
--pc_ctrl también está en 0 en q3
--Versión 2
--Incluye la implementación poco común de un direccionamiento indirecto
--a Status, Fsr, PCL y PClath

entity gen_dir is
port(fsr: buffer std_logic_vector(7 downto 0);
      status: buffer std_logic_vector(7 downto 0);
      ins: in std_logic_vector(6 downto 0);          --7 LSB de instrucción actual (I6 a
I0)
      ram_ena: in std_logic_vector(1 downto 0);      --indica si ram lee, escribe o no
hace nada
      q1, q2, q3, q4: in std_logic;                  --relojes
      z, dc, c: in std_logic;                        --banderas desde la
alu
      busdatos: inout std_logic_vector(7 downto 0); --bus de datos
      dir_ram: out std_logic_vector(8 downto 0);    --dirección de 9 bits hacia la ram
      ram_w, ram_r: out std_logic;                  --q2 ram escribe en bus, q4 ram lee
de bus
      pcl_warning: out std_logic;                    --advierte a la alu si se va a
trabajar PCL
      pc_ctrl: out std_logic_vector(2 downto 0));    --indica si pc o pclath, lee o
escribe
end gen_dir;

--miniram
--bit2=0 status, bit2=1 fsr
--bit1=0 no escribe, bit1=1 sí escribe
--bit0=0 no lee, bit0=1 sí lee

--pc_ctrl
--bit0 -> 0 no lee en Q2, 1 sí lee en Q2
--bit1 -> 0 no escribe en Q2, 1 sí escribe en Q2
--bit2 -> 0 es PCL, 1 es PCLATH

architecture comp of gen_dir is
begin
  process (q1,q2,q3,q4)
    variable var_w: std_logic; --equivale a ram_w en q2
    variable var_r: std_logic; --equivale a ram_r en q4
    variable var_pc: std_logic_vector(2 downto 0); --equivale a pc_ctrl
```

```

variable miniram: std_logic_vector(2 downto 0); --indica si opera como status o fsr el bus
de datos, lee o escribe
variable primeravez: std_logic:='1'; --indica si alu se ejecuta por primera vez
begin

```

```

-----
--en Q1+retraso genera la direcci3n
--Nota:
--Debe llevar retraso ya que recibe ram_ena desde la entidad decod&control
if q1='1' then
  --Revisa si es su primera ejecuci3n
  if primeravez='1' then
    status<="00000000";
    fsr<="00000000";
    primeravez:='0';
  end if;

  ram_w<='0'; ram_r<='0'; busdatos<="ZZZZZZZZ"; pc_ctrl<="000";
  case (ram_ena) is
    --no escribe, no lee
    when "00" => var_w:='0'; var_r:='0'; miniram:="000"; var_pc:="000";
    --genera la direcci3n
    when others =>
      case ins is

```

```

-----
--dir. indirecto
when "0000000" => dir_ram<=status(7)&fsr; --IRP + FSR
  --subcasos de dir. indirecto
  case fsr(6 downto 0) is
    --status indirecto
    when "0000011" =>
      var_w:='0'; --ram no escribe en q2
      var_r:='0'; --ram no lee en q4
      miniram:='0'&ram_ena(1)&ram_ena(0); --status hace lo que ram_ena indique
      var_pc:="000"; --pcl y pclath no actúan
    --fsr indirecto
    when "0000100" =>
      var_w:='0'; --ram no escribe en q2
      var_r:='0'; --ram no lee en q4
      miniram:='1'&ram_ena(1)&ram_ena(0); --fsr hace lo que ram_ena indique
      var_pc:="000"; --pcl y pclath no actúan
    --pcl indirecto
    when "0000010" =>
      var_w:='0'; --ram no escribe en q2
      var_r:='0'; --ram no lee en q4
      miniram:="000"; --status ni fsr actúan
      var_pc:='0'&ram_ena(1)&ram_ena(0); --pcl hace lo que ram_ena indique
      pcl_warning<='1'; --advierde a alu que pcl va a trabajarse
    --pclath indirecto
    when "0001010" =>
      var_w:='0'; --ram no escribe en q2
      var_r:='0'; --ram no lee en q4
      miniram:="000"; --status ni fsr actúan
      var_pc:='1'&ram_ena(1)&ram_ena(0); --pclath hace lo que ram_ena indique
    when others =>
      var_w:=ram_ena(1); --escritura en q2
      var_r:=ram_ena(0); --lectura en q4
      miniram:="000"; --status y fsr no actúan
      var_pc:="000"; --pcl y pclath no actúan
  end case;

```

```

-----
--direcci3n status
when "0000011" =>
  var_w:='0'; --ram no escribe en q2
  var_r:='0'; --ram no lee en q4
  miniram:='0'&ram_ena(1)&ram_ena(0); --status hace lo que ram_ena indique

```

```

var_pc:="000"; --pcl y pclath no actúan
-----
--dirección fsr
when "0000100" =>
  var_w:='0'; --ram no escribe en q2
  var_r:='0'; --ram no lee en q4
  miniram:='1'&ram_ena(1)&ram_ena(0); --fsr hace lo que ram_ena indique
  var_pc:="000"; --pcl y pclath no actúan
-----
--dirección pcl
when "0000010" =>
  var_w:='0'; --ram no escribe en q2
  var_r:='0'; --ram no lee en q4
  miniram:="000"; --status ni fsr actúan
  var_pc:='0'&ram_ena(1)&ram_ena(0); --pcl hace lo que ram_ena indique
  pcl_warning<='1'; --advierete a alu que pcl va a trabajarse
-----
--dirección pclath
when "0001010" =>
  var_w:='0'; --ram no escribe en q2
  var_r:='0'; --ram no lee en q4
  miniram:="000"; --status ni fsr actúan
  var_pc:='1'&ram_ena(1)&ram_ena(0); --pclath hace lo que ram_ena indique
-----
--dir. directo
when others => dir_ram<=status(6)&status(5)&ins(6 downto 0); --RP1 + RP0 + inst6..0
  var_w:=ram_ena(1); --escritura en q2
  var_r:=ram_ena(0); --lectura en q4
  miniram:="000"; --status y fsr no actúan
  var_pc:="000"; --pcl y pclath no actúan
end case;
end case;
end if;
-----
--en q2 habilita la ram o a status y fsr (escriben al bus)
--o ninguno escribe y entidad pc escribe
if q2='1' then
  ram_w<=var_w;
  ram_r<='0'; --nunca lee en q2
  --status
  if miniram(2)='0' then
    if miniram(1)='0' then busdatos<="ZZZZZZZZ"; else busdatos<=status; end if;
  end if;
  --fsr
  if miniram(2)='1' then
    if miniram(1)='0' then busdatos<="ZZZZZZZZ"; else busdatos<=fsr; end if;
  end if;
  --pcl y pclath
  pc_ctrl<=var_pc(2)&var_pc(1)&'0'; --controla la entidad pc, no lee en q2
end if;
-----
--en Q3 pone en alta impedancia el busdatos y apaga otras entidades(ALU absorbe dato en
Q2+retraso)
if q3='1' then
  busdatos<="ZZZZZZZZ";
  ram_w<='0';
  ram_r<='0';
  pc_ctrl<=var_pc(2)&"00"; --var_pc(2) notifica a alu si modifica pcl o no para FLUSH
end if;

```

```

-----
--en Q4 la ram, status o fsr y c, dc y z se actualizan
if q4='1' then
  ram_r<=var_r;
  ram_w<='0'; --nunca escribe en q4
  --status
  if miniram(2)='0' then
    if miniram(0)='1' then status<=busdatos; end if;
  end if;
  --fsr
  if miniram(2)='1' then
    if miniram(0)='1' then fsr<=busdatos; end if;
  end if;
  --pcl y pclath
  pc_ctrl<=var_pc(2)&'0'&var_pc(0); --controla la entidad pc, no escribe en q4

  status(2)<=z;
  status(1)<=dc;
  status(0)<=c;

  --En q4 pcl_warning ya habrá sido leído por la alu, se predetermina a 0
  pcl_warning<='0';

end if;

end process;
end comp;

```

## Entidad RAM

La RAM en los PIC16 se maneja como 4 bancos de 128 localidades de memoria. Para el caso de la simulación se trabajará solo con un banco de 512 bytes que equivale a los 4 bancos de un PIC. La entidad ram se encarga de escribir o leer al bus de datos el contenido de una localidad de memoria apuntada por el generador de direcciones. Se activa en 4 casos:

- Cuando ram\_w está en alto
- Cuando ram\_r está en alto
- Cuando Q1 está en alto
- Cuando Q3 está en alto

En Q1 pone el bus de datos en alta impedancia. En ram\_w se encarga de escribir un dato al bus de la localidad de memoria que se le indique. En Q3 pone el bus de datos en alta impedancia para evitar colisiones con el resultado de la ALU. En ram\_r copia el contenido del bus a una localidad de memoria. Las banderas ram\_w y ram\_r se activan en Q2 y en Q4 respectivamente desde el generador de direcciones.

El diagrama de la entidad se ve en la figura 7 y sus salidas se describen a continuación.

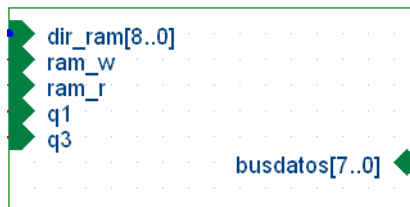


Figura 7. Entidad RAM.

Entradas

- dir\_ram. Recibe 9 bits que apuntan la localidad de ram a emplear.
  - ram\_w. Recibe la orden de escribir sobre el bus de datos.
  - ram\_r. Recibe la orden de leer desde el bus de datos.
  - q1 y q3. Recibe la orden de poner el bus de datos en alta impedancia.
- Entrada/Salida
- busdatos. 8 bits de dato a leer o escribir.

A continuación se muestra un diagrama de tiempos de trabajo de la ram (figura 8).

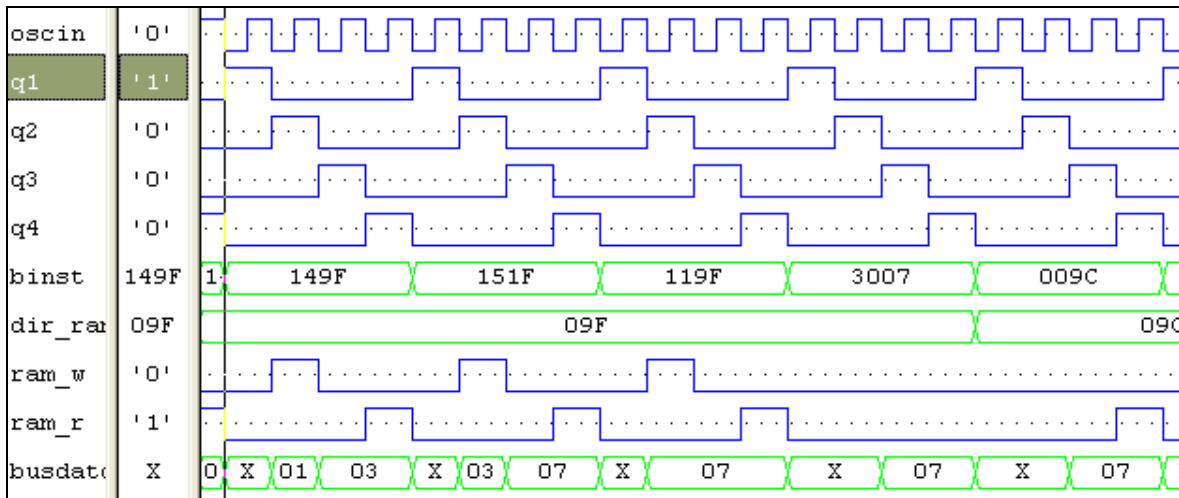


Figura 8. Diagrama de tiempos de la ram.

Se observa que la instrucción 0x149F pide que el contenido de la localidad 0x09F sea escrito sobre el bus de datos (0x01). La ALU la modifica y entrega 0x03. En Q4 la RAM debe guardar 0x03 en la localidad 0x09F. En otro ejemplo se tiene que la instrucción 0x009C no requiere que la ram escriba sobre el bus de datos en Q2, pero sí requiere que en Q4 la localidad ram 0x09C contenga el valor 0x07.

Código VHDL:

```
--Entidad RAM v1
--Simula la existencia de los 4 bancos del pic en 512 localidades continuas
--Sensible a Q3, ram_w, ram_r

LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; --operaciones aritméticas

--type matriz is array(0 to 511) of std_logic_vector(7 downto 0);

entity ent_ram is
port(dir_ram: in std_logic_vector(8 downto 0);
     ram_w: in std_logic;
     ram_r: in std_logic;
     q1,q3: in std_logic;
     busdatos: inout std_logic_vector(7 downto 0));
end ent_ram;

architecture comp of ent_ram is
type matriz is array(0 to 511) of std_logic_vector(7 downto 0);
begin
process(ram_w,ram_r,q1,q3)
variable rampic: matriz;
variable primeravez: std_logic:='1'; --indica si la ram fue recién encendida
```

```

variable contador: std_logic_vector(9 downto 0); --solo se usa para resetear la ram
begin
--Sensibilidad a q1
if q1='1' then
busdatos<="ZZZZZZZZ";
--Primera operación de la ram
if primeravez='1' then
contador:="0000000000";
while contador<"1000000000" loop --512 veces
rampic(conv_integer(contador)):"00000000";
contador:=contador+1;
end loop;
primeravez:='0';
end if;
end if;

--Sensibilidad a ram_w (cuando q2 se activa, ram debe escribir)
if ram_w='1' then
busdatos<=rampic(conv_integer(dir_ram));
end if;

--Sensibilidad a ram_r (cuando q4 se activa, ram debe leer)
if ram_r='1' then
rampic(conv_integer(dir_ram)):=busdatos;
end if;

--Sensibilidad a q3 (busdatos es escrito por alu)
if q3='1' then
busdatos<="ZZZZZZZZ";
end if;

end process;
end comp;

```

## Entidad ALU

La entidad ALU es la más importante de todas las entidades ya que calcula la mayoría de datos que el microcontrolador maneja. Se encarga de actualizar el valor de las banderas c, dc y z contenidas en status. Se encarga de indicarle a la entidad eeprom cuándo debe entregar una instrucción NOP en vez de una instrucción normal (instrucciones que implican salto). Se encarga de indicarle al contador de programa cómo debe ser actualizado el apuntador de la memoria de programa.

La ALU es sensible a todos los pulsos Q. En Q1 pone el bus de datos en alta impedancia. En Q2 se encarga de copiar el contenido del bus de datos o de la literal k si es necesario. En Q3 entrega el resultado del cálculo e indica a la entidad EEPROM si debe ejecutar la siguiente instrucción como NOP. En Q4 actualiza el valor del registro de trabajo W si la instrucción lo amerita.

Se implementó en esta entidad de forma interna el registro W ya que este registro solo es accesado por la ALU de forma privada. Ninguna otra entidad tiene acceso al registro W.

La entidad ALU se puede ver en la figura 9 y sus entradas y salidas se comentan a continuación.

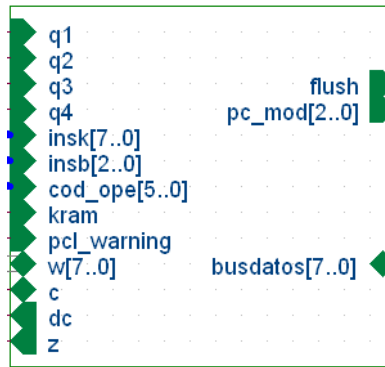


Figura 9. Entidad ALU.

#### Entradas

- q1, q2, q3 y q4. Pulsos que activan las diferentes actividades de la ALU.
- insk. Recibe los 8 bits de la literal k de la instrucción (solo en casos de direccionamiento inmediato).
- insb. Recibe los 3 bits de la instrucción que indican que bit se ha de modificar de un dato (bit 9, bit 8 y bit 7 de la instrucción).
- cod\_ope. Recibe 6 bits desde la entidad de decodificación y control. Indican que operación ha de realizar la ALU en Q3.
- k\_ram. Indica a la ALU qué dato tomar: literal k para dir. inmediato o dato del bus de datos.
- pcl\_warning. Advertencia desde la entidad generador de direcciones. Indica si PCL será modificado en Q4.

#### Salidas

- c, dc y z. Envía el estado de las banderas a la entidad generador de direcciones que contiene internamente a status.
- flush. Ordena a la entidad EEPROM que la siguiente instrucción deberá ser ejecutada como NOP (saltos).
- pc\_mod. Determina la manera en que la nueva dirección de programa de la EEPROM será generada.

#### Entrada/Salida

- busdatos. Lee el bus en Q2 y escribe resultados en Q3. En Q4 actualiza W si la instrucción lo amerita.

La salida pc\_mod le indicará a la entidad contador de programa cómo manipular la dirección de programa según la operación que la ALU haya ejecutado. Se tienen 5 casos de modificación de la dirección de programa:

- "000" -> CALL
- "001" -> GOTO
- "010" -> RETURN, RETFIE y RETLW
- "011" -> BTFSC y BTFSS
- "100" -> INCFSZ con destino PCL, DECFSZ con destino PCL y cualquier instrucción con destino PCL
- "101" -> Cualquier otra instrucción que no afecte PCL

A continuación se muestra un diagrama de ejecución de la ALU mostrando algunas instrucciones y los resultados que genera (ver figura 10).

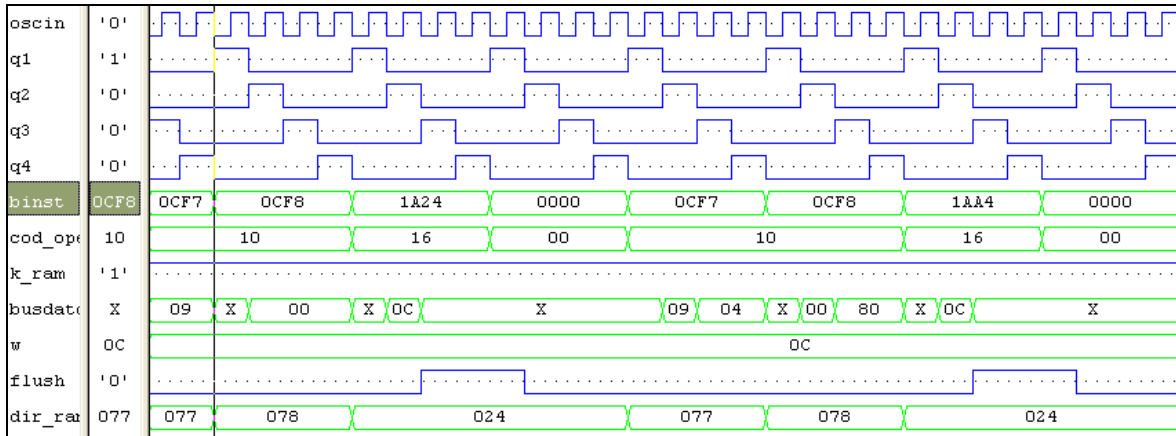


Figura 10. Diagrama de trabajo de la entidad ALU.

Operaciones descritas en la figura 10:

- Instrucción 0x0CF8. Decodificada como 10, RRF.
  - Q1. Bus en alta impedancia.
  - Q2. Localidad ram 0x078 escribe 0x00 en el bus y ALU lo copia después.
  - Q3. ALU rota a la derecha el valor 0x00 a través de la bandera C. Resulta 0x00.
  - Q4. Localidad ram 0x078 ahora vale 0x00.
- Instrucción 0x1A24. Decodificada como 16, BTFSC.
  - Q1. Bus en alta impedancia.
  - Q2. Localidad 0x024 escribe 0x0C en bus y ALU lo copia después.
  - Q3. ALU determina que el bit 4 de 0x0C es cero e indica a la EEPROM que debe ejecutar FLUSH.
  - Q4. No hay actividad.
- Instrucción 0x0000. Decodificada como 0, NOP.
 

Esta instrucción es ejecutada como NOP ya que la ALU pidió que así se realizara (hubo salto). No hay movimiento de datos.
- Instrucción 0x0CF7. Decodificada como 10, RRF.
  - Q1. Bus en alta impedancia.
  - Q2. Localidad ram 0x077 escribe 0x09 en el bus y ALU copia el dato.
  - Q3. ALU rota a la derecha el valor 0x09 a través de la bandera C. Resulta 0x04.
  - Q4. Localidad ram 0x077 ahora vale 0x04.
- Instrucción 0x0CF8. Decodificada como 10, RRF.
  - Q1. Bus en alta impedancia.
  - Q2. Localidad ram 0x078 escribe 0x00 en el bus y ALU copia el dato.
  - Q3. ALU rota a la derecha el valor 0x00 a través de la bandera C. Resulta 0x80.
  - Q4. Localidad ram 0x077 ahora vale 0x80.
- Instrucción 0x1AA4. Decodificada como 16, BTFSC.
  - Q1. Bus en alta impedancia.
  - Q2. Localidad 0x024 escribe 0x0C en bus y ALU lo copia después.
  - Q3. ALU determina que el bit 4 de 0x0C es cero e indica a la EEPROM que debe ejecutar FLUSH.
  - Q4. No hay actividad.
- Instrucción 0x0000. Decodificada como 0, NOP.
 

Esta instrucción es ejecutada como NOP ya que la ALU pidió que así se realizara (hubo salto). No hay movimiento de datos.

Se presenta ahora otra secuencia de operación de la ALU en la figura 11.

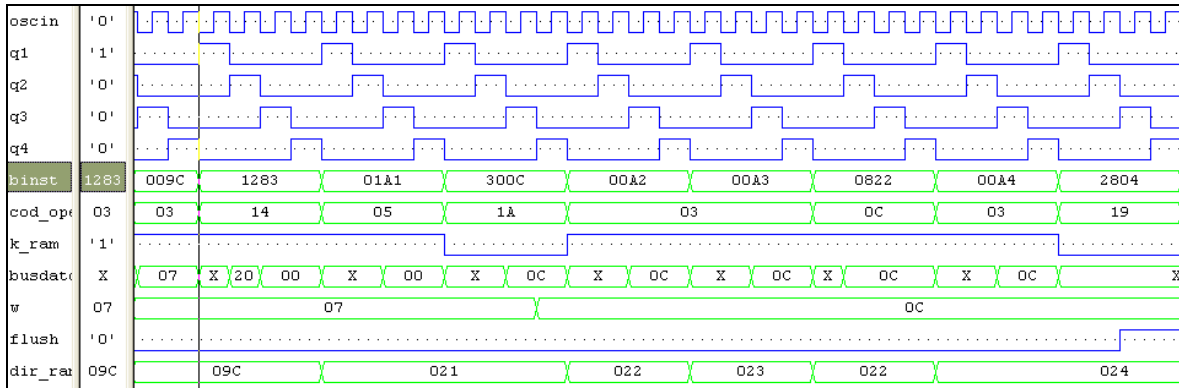


Figura 11. Diagrama de tiempos de la ALU.

Operaciones descritas en la figura 11:

- Instrucción 0x1283. Decodificada como 14, BCF.

Q1. Bus en alta impedancia.

Q2. Localidad ram 0x09C escribe 0x20 sobre el bus y ALU lo copia.

Q3. ALU pone un cero en el bit 5 del dato. Resulta 0x00.

Q4. Localidad ram ahora vale 0x00.

- Instrucción 0x01A1. Decodificada como 5, CLRF.

Q1. Bus en alta impedancia.

Q2. No hay actividad, la ALU no necesita dato de entrada.

Q3. ALU entrega un 0x00 en el bus.

Q4. Localidad 0x021 ahora vale 0x00.

- Instrucción 0x300C. Decodificada como 1A, MOVLW.

Q1. Bus en alta impedancia.

Q2. ALU copia la literal k de la instrucción (0x0C).

Q3. ALU entrega 0x0C en el bus.

Q4. El valor de W se actualiza con 0x0C.

- Instrucción 0x00A2. Decodificada como 3, MOVWF.

Q1. Bus en alta impedancia.

Q2. No hay actividad, ALU no requiere dato de entrada.

Q3. ALU entrega el valor de W en el bus (0x0C).

Q4. La localidad 0x022 ahora vale 0x0C.

- Instrucción 0x00A3. Decodificada como 3, MOVWF.

Q1. Bus alta impedancia.

Q2. No hay actividad, ALU no requiere dato de entrada.

Q3. ALU entrega el contenido de W en el bus (0x0C).

Q4. La localidad 0x023 ahora vale 0x0C.

- Instrucción 0x0822. Decodificada como C, MOVF.

Q1. Bus en alta impedancia.

Q2. La localidad 0x022 escribe su contenido en el bus (0x0C) y ALU lo copia.

Q3. ALU entrega 0x0C en el bus.

Q4. W copia el valor del bus.

- Instrucción 0x00A4. Decodificada como 0x03, MOVWF.

Q1. Bus en alta impedancia.

Q2. No hay actividad, ALU no requiere dato de entrada.

Q3. ALU entrega el contenido de W en el bus (0x0C).

Q4. La localidad 0x024 ahora vale 0x0C.

- Instrucción 0x2804. Decodificada como 19, GOTO.

Q1. Bus en alta impedancia.

- Q2. No hay actividad en la ALU.  
 Q3. ALU notifica que EEPROM debe ejecutar un FLUSH.  
 Q4. No hay actividad.

En los dos últimos ejemplos se pusieron en prueba instrucciones de byte, instrucciones de direccionamiento indirecto, instrucciones de salto e instrucciones de tipo bit.

Código VHDL:

```
--ALU para todas las instrucciones
--El registro w se maneja internamente
--proceso sensible a Q3 (PIC calcula) y Q4 (w se escribe)

--Versión 4
--Envía bandera FLUSH para brincar una instrucción en INCFSZ, DECFSZ, BTFSC y BTFSS
--Versión 3
--Corrige adquisición de datos haciéndolo hasta q2

LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; --operaciones aritméticas

entity alu_tot is
port(q1,q2,q3,q4: in std_logic;
     insk: in std_logic_vector(7 downto 0);           --8 lsb de inst
     insb: in std_logic_vector(2 downto 0);           --I9..I7 para instr. de bit
     cod_ope: in std_logic_vector(5 downto 0);        --33 inst = 2 a la 6
     kram: in std_logic;                               --0 es k, 1 es ram
     pcl_warning: in std_logic;                       --indica si pcl probablemente sea
modificado
     w: buffer std_logic_vector(7 downto 0);
     busdatos: inout std_logic_vector(7 downto 0);
     z,dc: out std_logic;
     flush: out std_logic;                             --flush indica si eeprom debe sacar NOP
     pc_mod: out std_logic_vector(2 downto 0);         --indica a entidad cont_prog cómo
manipular a PC
     c: buffer std_logic);                             --c es buffer porque
debe leerse en rrf y rlf
end alu_tot;

architecture compo of alu_tot is
begin
  process(q1,q2,q4,q3)
    variable res: std_logic_vector(7 downto 0);       --útil para ver si se levanta z
    variable t4add,t4sub,w4,k4: std_logic_vector(4 downto 0); --4 bits y carry para operaciones
ADDLW
    variable t8add,t8sub,w8,k8: std_logic_vector(8 downto 0); --8 bits y carry para operaciones
SUBLW
    variable t4addw,t4subw,bus4: std_logic_vector(4 downto 0); --4 bits y carry para
operaciones ADDWF
    variable t8addw,t8subw,bus8: std_logic_vector(8 downto 0); --8 bits y carry para
operaciones SUBWF
    variable muxout: std_logic_vector(7 downto 0);    --guarda lo que el mux
seleccionó (w o bus)
    variable cc: std_logic;                           --copia
de c
    variable ww: std_logic;                           --si es
'0', actualiza w en q4
    variable primeravez: std_logic:='1';             --indica si alu corre por primera
vez
  begin

    --pc_mod
    --0 call
    --1 goto
    --2 return, retfie, retlw
```

```

--3 btfsc, btfss
--4 incfsz, decfsz, cualquier otra con destino pcl
--5 cualquier instrucción ajena a pcl

```

```

-----
--Q1 - busdatos alta impedancia
if q1='1' then
  busdatos<="ZZZZZZZ";
  --Si ALU se ejecuta por primera vez, inicializa banderas c, dc y z
  if primeravez='1' then
    c<='0';
    dc<='0';
    z<='0';
    primeravez:='0';
  end if;
end if;

```

```

-----
--Q2 - Después del retraso con buffer, lee busdatos
--Nota:
--cuando q3 es 1, q2 sigue siendo 1 por el retraso del buffer
--q2&q3 evita que muxout se escriba en q3
if (q2&q3)="10" then
  --actualización de variables internas de process
  k4:= '0'&insk(3 downto 0); --LW
  w4:= '0'&w(3 downto 0);
  bus4:= '0'&busdatos(3 downto 0); --WF
  w8:= '0'&w;
  k8:= '0'&insk; --LW
  bus8:= '0'&busdatos; --WF
  t4add:= k4 + w4;
  t4sub:= k4 - w4;
  t8add:= k8 + w8;
  t8sub:= k8 - w8;
  t4addw:= bus4 + w4;
  t4subw:= bus4 - w4;
  t8addw:= bus8 + w8;
  t8subw:= bus8 - w8;
  --k entra a la alu
  if kram='0' then muxout:=insk; end if;
  --busdatos entra a la alu
  if kram='1' then muxout:=busdatos; end if;
  --hace una copia de c en cc
  cc:=c;
  --Predetermina flush = 0, pero en q3 se actualiza correctamente
  flush<='0';
  --Revisa si la ALU modificará el valor de Q3
  --Si es el caso entonces avisa a cont_prog que concatene pclath(4..0)&pcl
  --poniendo pc_mod="100"
  if pcl_warning&insk(7)="11" then
    pc_mod<="100";
  --Predetermina pc_mod a instrucción ajena a pc, pero en q3 se actualiza correctamente
  else
    pc_mod<="101";
  end if;
end if;

```

```

-----
--Q3 - Calcula
case (q3&cod_ope) is
--NOP
when "1000000" => null; ww:='1';
--RETURN
when "1000001" => null; ww:='1'; pc_mod<="010";

```

```

--RETFIE
when "1000010" => null; ww:='1'; pc_mod<="010";
--MOVWF
when "1000011" => busdatos<=w; ww:='1';
--CLRWF
when "1000100" => busdatos<="00000000"; z<='1'; ww:='0';
--CLRF
when "1000101" => busdatos<="00000000"; z<='1'; ww:='1';
--SUBWF
when "1000110" =>
  res:=muxout-w;
  busdatos<=res;
  if res="00000000" then z<='1'; else z<='0'; end if;
  c<=t8subw(8); dc<=t4subw(4);
  ww:=insk(7);
--DECF
when "1000111" =>
  res:=muxout-1;
  busdatos<=res;
  if res="00000000" then z<='1'; else z<='0'; end if;
  ww:=insk(7);
--IORWF
when "1001000" =>
  res:=muxout or w;
  busdatos<=res;
  if res="00000000" then z<='1'; else z<='0'; end if;
  ww:=insk(7);
--ANDWF
when "1001001" =>
  res:=muxout and w;
  busdatos<=res;
  if res="00000000" then z<='1'; else z<='0'; end if;
  ww:=insk(7);
--XORWF
when "1001010" =>
  res:=muxout xor w;
  busdatos<=res;
  if res="00000000" then z<='1'; else z<='0'; end if;
  ww:=insk(7);
--ADDWF
when "1001011" =>
  res:=muxout + w;
  busdatos<=res;
  if res="00000000" then z<='1'; else z<='0'; end if;
  c<=t8addw(8); dc<=t4addw(4);
  ww:=insk(7);
--MOVF
when "1001100" =>
  busdatos<=muxout;
  if muxout="00000000" then z<='1'; else z<='0'; end if;
  ww:=insk(7);
--COMF
when "1001101" =>
  res:=not muxout;
  busdatos<=res;
  if res="00000000" then z<='1'; else z<='0'; end if;
  ww:=insk(7);
--INCF
when "1001110" =>
  res:=muxout+1;
  busdatos<=res;
  if res="00000000" then z<='1'; else z<='0'; end if;
  ww:=insk(7);
--DECFSZ
when "1001111" =>
  res:=muxout-1;
  busdatos<=res;
  if res="00000000" then flush<='1'; end if; --avisa a entidad EEPROM que debe poner NOP

```

```

ww:=insb(7); pc_mod<="100";
--RRF
when "1010000" =>
res:=muxout srl 1; --rota a la derecha 1 vez
res(7):=cc; --el bit nuevo es igual a c
busdatos<=res;
c<=muxout(0);
ww:=insb(7);
--RLF
when "1010001" =>
res:=muxout sll 1; --rota a la izquierda 1 vez
res(0):=cc;
busdatos<=res;
c<=muxout(7);
ww:=insb(7);
--SWAPF
when "1010010" =>
busdatos<=muxout(3 downto 0)&muxout(7 downto 4);
ww:=insb(7);
--INCFSZ
when "1010011" =>
res:=muxout+1;
busdatos<=res;
if res="00000000" then flush<='1'; end if; --avisa a entidad EEPROM que debe poner NOP
ww:=insb(7); pc_mod<="100";
--BCF
when "1010100" =>
res:=muxout;
res(conv_integer(insb)):= '0';
busdatos<=res;
ww:= '1';
--BSF
when "1010101" =>
res:=muxout;
res(conv_integer(insb)):= '1';
busdatos<=res;
ww:= '1';
--BTFSC
when "1010110" =>
if muxout(conv_integer(insb))='0' then
flush<='1';
end if;
ww:= '1'; pc_mod<="011";
--BTFSS
when "1010111" =>
if muxout(conv_integer(insb))='1' then
flush<='1';
end if;
ww:= '1'; pc_mod<="011";
--CALL
when "1011000" => ww:= '1'; pc_mod<="000"; flush<='1';
--GOTO
when "1011001" => ww:= '1'; pc_mod<="001"; flush<='1';
--MOVLW
when "1011010" => busdatos<=muxout; ww:= '0';
--RETLW
when "1011011" => busdatos<=muxout; ww:= '0'; pc_mod<="010";
--IORLW
when "1011100" =>
res:=muxout or w;
busdatos<=res;
if res="00000000" then z<='1'; else z<='0'; end if;
ww:= '0';
--ANDLW
when "1011101" =>
res:=muxout and w;
busdatos<=res;
if res="00000000" then z<='1'; else z<='0'; end if;

```

```

    ww='0';
--XORLW
when "1011110" =>
    res:=muxout xor w;
    busdatos<=res;
    if res="00000000" then z<='1'; else z<='0'; end if;
    ww='0';
--SUBLW
when "1011111" =>
    res:=muxout-w;
    busdatos<=res;
    if res="00000000" then z<='1'; else z<='0'; end if;
    c<=t8sub(8); dc<=t4sub(4);
    ww='0';
--ADDLW
when "1100000" =>
    res:=muxout+w;
    busdatos<=res;
    if res="00000000" then z<='1'; else z<='0'; end if;
    c<=t8add(8); dc<=t4add(4);
    ww='0';
end case;
-----
-----
--Q4 - actualiza w si debe hacerlo
if q4='1' then
    if ww='0' then w <= busdatos; end if;
end if;

end process;
end compo;

```

## Entidad Contador de Programa

El contador de programa se encarga de pedirle a la entidad EEPROM que entregue la siguiente instrucción a ejecutar mediante una dirección de programa de acuerdo a la orden que la ALU envíe.

Se tienen 5 modos de modificación de la dirección de programa (ver figura 12). A continuación se describe el funcionamiento de cada uno de ellos.

**CALL.** Se guarda la dirección actual de memoria en la pila. Se toman los 11 LSB de la instrucción y se concatenan con los bits 3 y 4 del registro PCLATH, posteriormente se decrementa en uno la dirección para simular el NOP de salto.

**GOTO.** Se toman los 11 LSB de la instrucción y se concatenan con los bits 3 y 4 del registro PCLATH, posteriormente se decrementa en uno la dirección para simular el NOP de salto.

**RETURN, RETFIE y RETLW.** Se toma la dirección contenida en la posición superior de la pila y se incrementa en uno.

**BTFSC y BTFSS.** Se incrementa en uno la dirección actual. Si la ALU determina que debe haber salto, la entidad EEPROM no entregará la siguiente instrucción en memoria, entregará una instrucción NOP (00000000000000).

**INCFSZ con destino PCL, DECFSZ con destino PCL y cualquier instrucción con destino PCL.** Si cualquier operación de la ALU involucró el registro PCL, la nueva dirección de programa será la concatenación de los 5 MSb de PCLATH y los 8 bits del nuevo contenido de PCL incrementado en

uno. Además si la entidad ALU notifica que hay que hacer salto, la entidad EEPROM entregará un NOP.

Cualquier otra instrucción que no involucre a PCL. Se incrementa en uno la dirección actual.

En la figura 13 se observa la entidad Contador de programa y sus entradas y salidas se describen a continuación.

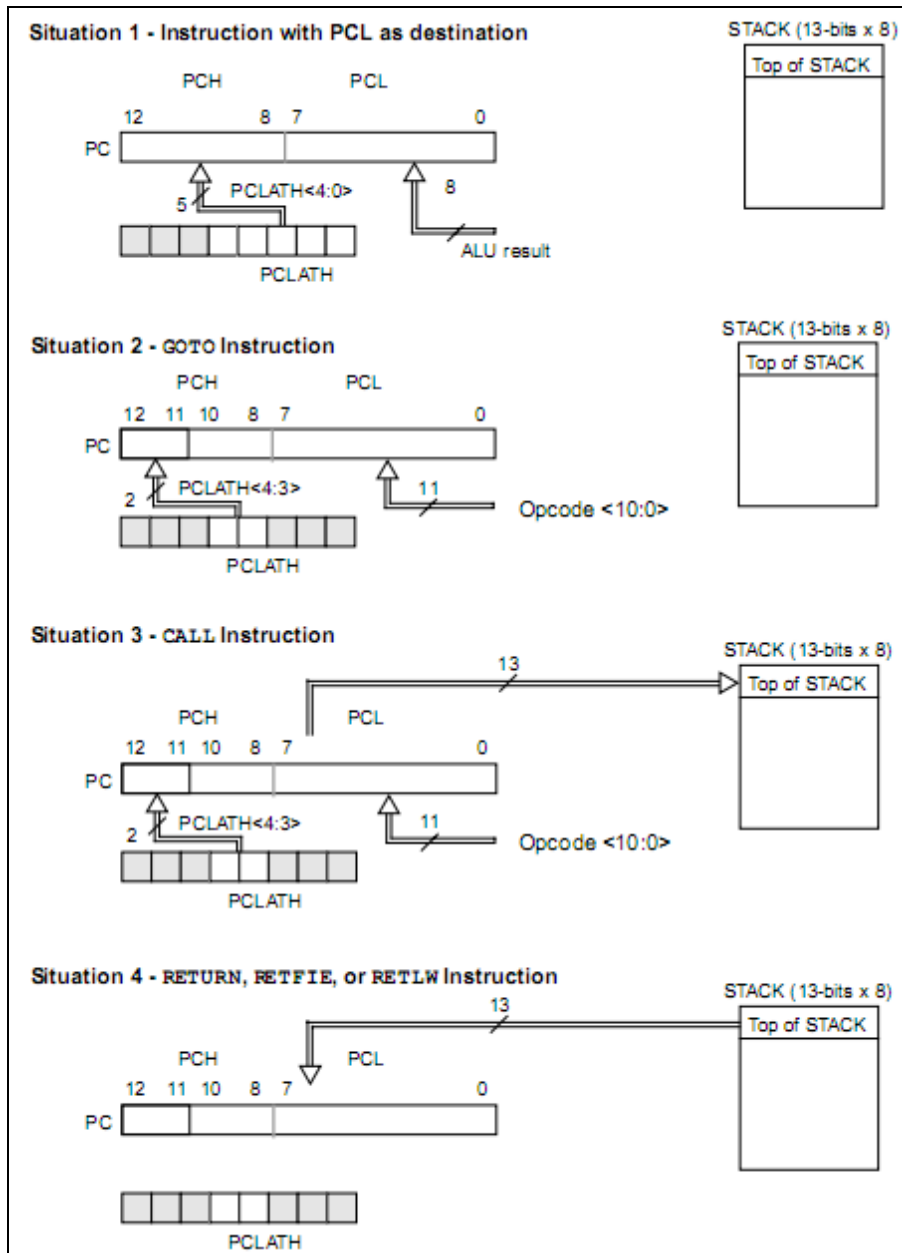


Figura 12. Cuatro modificaciones principales de la dirección de programa.

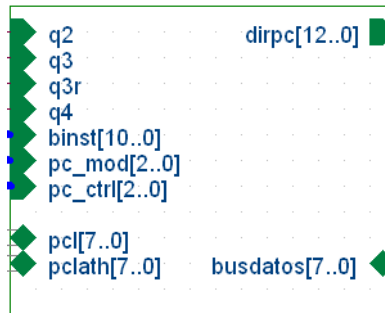


Figura 13. Entidad contador de programa.

#### Entradas

- q2, q3, q3r, q4. Reciben los pulsos de reloj de Q; q3r es una señal con retraso de q3 (útil para leer el resultado de la ALU si PCL o PCLATH están involucrados).
- binst. Recibe 11 bits de la instrucción actual (útil para CALL y GOTO).
- pc\_mod. La ALU indica mediante estas líneas qué modificación ha de realizarse sobre la dirección de programa.
- pc\_ctrl. El generador de direcciones indica mediante estas líneas cuándo debe escribirse o leerse el contenido de PCL o PCLATH sobre el bus de datos.

#### Salidas

- dirpc. Entrega 13 bits de dirección de programa a la entidad EEPROM.

#### Entrada/Salida

- busdatos. Entrega o lee el contenido de PCL o PCLATH sobre el bus de datos.

La pila que contiene las direcciones de memoria de llamados a subrutinas es del tipo FIFO (primero que entra, primero que sale).

#### *Notas importantes sobre la entidad:*

Existe un error en la asignación del valor de PCLATH que no pudo ser depurado, el empleo de saltos entre páginas de memoria es por lo tanto inestable.

La entidad simula correctamente el segundo ciclo de instrucción de un PIC al existir salto, ejecutándolo como una instrucción NOP a excepción de las instrucciones RETURN, RETLW y RETFIE que no lo tienen implementado.

El empleo de tablas en un programa se realiza correctamente al emplear la modificación pc\_mod "100".

A continuación se muestra un diagrama de tiempos con el comportamiento de la entidad contador de programa (figura 14).

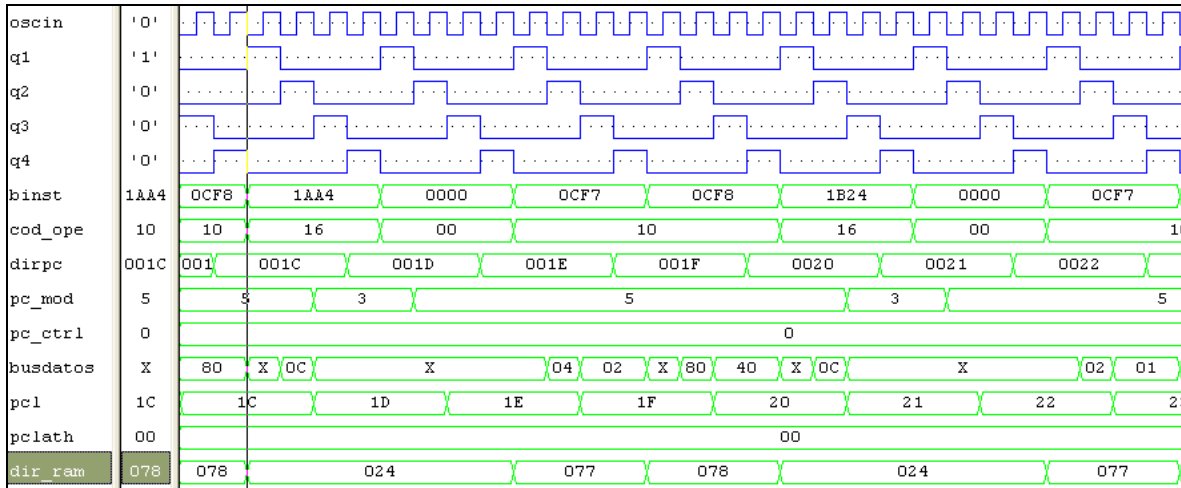


Figura 14. Diagrama de tiempos de la entidad contador de programa.

Actividad de la entidad contador de programa:

- Instrucción 0x1AA4. Decodificada como 16, BTFSC.
  - Q1. No hay actividad de la entidad.
  - Q2. No hay actividad de la entidad.
  - Q3. Recibe la orden de la ALU de manipular el PC como una instrucción BTFSC.
  - Q4. Incrementa en uno la dirección de programa. La entidad EEPROM fue notificada con un Flush y entrega NOP en vez de la siguiente instrucción.
- Instrucción 0x0000. Decodificada como 0, NOP.
  - Q1. No hay actividad de la entidad.
  - Q2. No hay actividad de la entidad.
  - Q3. Recibe la orden de la ALU de manipular el PC como una instrucción NOP.
  - Q4. Incrementa en uno la dirección de programa.
- Instrucción 0x0CF7. Decodificada como 10, RRF.
  - Q1. No hay actividad de la entidad.
  - Q2. No hay actividad de la entidad.
  - Q3. Recibe la orden de la ALU de manipular el PC como una instrucción RRF.
  - Q4. Incrementa en uno la dirección de programa.
- Instrucción 0x0CF8. Decodificada como 10, RRF.
  - Q1. No hay actividad de la entidad.
  - Q2. No hay actividad de la entidad.
  - Q3. Recibe la orden de la ALU de manipular el PC como una instrucción RRF.
  - Q4. Incrementa en uno la dirección de programa.
- Instrucción 0x1BA4. Decodificada como 16, BTFSC.
  - Q1. No hay actividad de la entidad.
  - Q2. No hay actividad de la entidad.
  - Q3. Recibe la orden de la ALU de manipular el PC como una instrucción BTFSC.
  - Q4. Incrementa en uno la dirección de programa. La entidad EEPROM fue notificada con un Flush y entrega NOP en vez de la siguiente instrucción.
- Instrucción 0x0000. Decodificada como 0, NOP.
  - Q1. No hay actividad de la entidad.
  - Q2. No hay actividad de la entidad.
  - Q3. Recibe la orden de la ALU de manipular el PC como una instrucción NOP.
  - Q4. Incrementa en uno la dirección de programa.

Código VHDL:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; --operaciones aritméticas

--Entidad contador de programa
--Incluye los registros PCL y PCLATH internamente
--Incluye la PILA de 8 posiciones

--Versión 1

entity cont_prog is
port(q2,q3,q3r,q4: in std_logic;
      binst: in std_logic_vector(10 downto 0); --instrucción actual
      pc_mod: in std_logic_vector(2 downto 0); --operación sobre pc a hacer
      pc_ctrl: in std_logic_vector(2 downto 0); --pc_ctrl actividad de PCL y PCLATH sobre
busdatos
      busdatos: inout std_logic_vector(7 downto 0); --busdatos
      dirpc: out std_logic_vector(12 downto 0); --dirección hacia la eeprom
      pcl: buffer std_logic_vector(7 downto 0); --pcl
      pclath: buffer std_logic_vector(7 downto 0)); --pclath
end cont_prog;

architecture compo of cont_prog is
type matriz is array(0 to 7) of std_logic_vector(12 downto 0);
begin
  process(q2,q3,q4)
    variable pc: std_logic_vector(12 downto 0); --variable interna de apuntador de programa
    variable stack: matriz; --pila de 8 niveles
    variable vpcl: std_logic_vector(7 downto 0); --copia de pcl
    variable destpcl: std_logic; --indica si pcl fue modificado por la alu
    variable primeravez: std_logic:='1'; --indica si se ejecuta por
    primera vez
  begin
    -----
    -----
    --Q2+r
    --Se verifica escritura de PCL y PCLATH
    if q2&q3&q3r&q4="1000" then
      --Si entidad cont_prog se ejecuta por primera vez
      if primeravez='1' then
        pcl<="00000000";
        pclath<="00000000";
        pc:="00000000000000";
        primeravez:='0';
      end if;
      --PCL escribe
      if pc_ctrl(2 downto 1)="01" then
        busdatos<=pcl;
      end if;
      --PCLATH escribe
      if pc_ctrl(2 downto 1)="11" then
        busdatos<=pclath;
      end if;
      --Respalda valor de pcl
      vpcl:=pcl;
      --Resetea bandera destino pcl
      destpcl:='0';
    end if;
    -----
    -----
    --Q3
    --Nota: Q3 se traslapa con Q2+r
    --Se pone el busdatos a Z
    if q2&q3&q3r&q4="1100" then
      busdatos<="ZZZZZZZZ";
    end if;
  end process;
end architecture compo;

```

```

-----
-----
--Q3 + retraso
--Nota: Q3+r se traslapa con Q3
  if q2&q3&q3r&q4="0110" then
    case pc_mod is

      --CALL
      when "000" =>
        stack(7):=stack(6);
        stack(6):=stack(5);
        stack(5):=stack(4);
        stack(4):=stack(3);
        stack(3):=stack(2);
        stack(2):=stack(1);
        stack(1):=stack(0);
        stack(0):=pc;

      --GOTO
      when "001" =>
        pc:=pclath(4 downto 3)&(binst-1);

      --RETURN, RETFIE, RETLW
      when "010" =>
        pc:=stack(0)+1;
        stack(0):=stack(1);
        stack(1):=stack(2);
        stack(2):=stack(3);
        stack(3):=stack(4);
        stack(4):=stack(5);
        stack(5):=stack(6);
        stack(6):=stack(7);

      --BTFSC, BTFSS
      when "011" =>
        pcl<=vpcl+1;
        pc:=pc+1;

      --INCFSZ, DECFSZ, destino PCL
      when "100" =>
        --Si el resultado ALU se guarda en PCL
        if binst(7)&pc_ctrl(2)="10" then
          pcl<=busdatos+1;
          destpcl:='1'; --avisa que el destino fue pcl
        else
          pc:=pc+1;
          pcl<=vpcl+1;
        end if;
        --Si el resultado ALU se guarda en PCLATH
        if binst(7)&pc_ctrl(2)="11" then
          pclath<=busdatos;
          pc:=pc+1;
          pcl<=vpcl+1;
        end if;

      --Instrucciones que no afectan a PC
      when "101" =>
        pcl<=vpcl+1;
        pc:=pc+1;

    end case;
  end if;

-----
-----
--Q4
--Sale la nueva dirección de programa

```

```

if q2&q3&q3r&q4="0011" then
  case pc_mod is

    --CALL
    when "000" =>
      pc:=pclath(4 downto 3)&(binst-1); --menos uno para simular el flush NOP
      dirpc<=pclath(4 downto 3)&(binst-1);
      pcl<=binst(7 downto 0)-1;

    --GOTO
    when "001" =>
      dirpc<=pc;
      pcl<=pc(7 downto 0);

    --RETURN, RETFIE, RETLW
    when "010" =>
      dirpc<=pc;
      pcl<=pc(7 downto 0);

    --BTFSZ,BTFSS
    when "011" =>
      dirpc<=pc;

    --INCFSZ, DECFSZ, destino PCL
    when "100" =>
      --Si el destino fue PCL
      if destpcl='1' then
        pc:=pclath(4 downto 0)&pcl;
        dirpc<=pclath(4 downto 0)&pcl;
      end if;
      --Si el destino no fue PCL
      if destpcl='0' then
        dirpc<=pc;
      end if;

    --Instrucciones ajenas a pc
    when "101" =>
      dirpc<=pc;

  end case;
end if;

end process;
end compo;

```

## Entidad EEPROM

La EEPROM se encarga de proveer la siguiente instrucción en Q1 a todas las entidades que la requieran.

Entrega la instrucción localizada en la dirección de memoria que la entidad contador de programa le indique. Cuando la ALU indica que ejecute un flush (al requerir un salto), la EEPROM entregará una instrucción NOP (00000000000000), según lo marca la hoja de especificaciones de los PIC16.

El contenido de la EEPROM se obtiene a través del código hexadecimal contenido en un archivo .hex. El contenido del archivo .hex puede ser obtenido mediante compiladores de lenguaje C como CCS, compiladores de lenguaje Basic o el mismo ensamblador MPLAB.

Posteriormente con la aplicación hex2txt se convierte el archivo .hex a cadenas de texto en código ascii que el simulador Orcad 9 puede procesar (ver código VHDL de la entidad).

La figura 15 muestra las entradas y salidas de la entidad EEPROM.

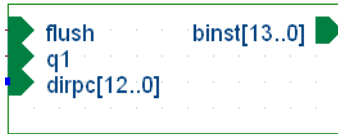


Figura 15. Entidad EEPROM.

**Entradas**

- flush. Indica si la siguiente instrucción a entregar será un NOP.
- q1. En el pulso q1 se deberá entregar la nueva instrucción.
- dirpc. El contador de programa le indica a la EEPROM que instrucción entregar (13 bits generan un máximo de 8192 localidades).

**Salida**

- binst. Entrega los 14 bits de la nueva instrucción.

En la siguiente figura se muestra un diagrama de tiempos con el comportamiento de la entidad EEPROM (figura 16).

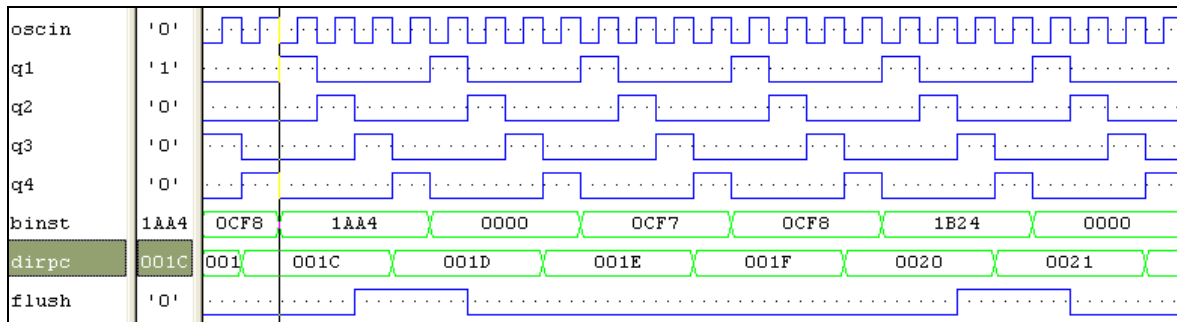


Figura 16. Diagrama de tiempos de la entidad EEPROM.

La instrucción 0x1AA4 (BTFSC) invoca un salto, el cual se notifica a la EEPROM mediante la activación de flush desde la ALU. La EEPROM no entrega el contenido de la dirección 0x001D, entrega un NOP.

**Nota importante de la entidad:**

La ejecución de un segundo ciclo de instrucción en la simulación no es necesaria ya que la simulación puede ejecutar toda instrucción en un ciclo de instrucción al no existir exigencias de tiempo. La segunda instrucción como NOP se implementó con la intención de acercar la simulación a la ejecución real de un PIC16.

**Código VHDL:**

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; --operaciones aritméticas
use ieee.numeric_std.all;
use std.textio.all;
use IEEE.std_logic_textio.all;

--Entidad que simula la memoria de programa del PIC
--8k x 14

--Versión 1

entity eeprom is

```

```

port(flush: in std_logic;          --flush indica a eeprom sacar un
NOP
    q1: in std_logic;              --sensible a q1
    dirpc: in std_logic_vector(12 downto 0); --dirección de memoria generada
por cont_prog
    binst: out std_logic_vector(13 downto 0)); --salida de la localidad apuntada
por dirpc
end eeprom;

architecture compo of eeprom is
type matriz is array(0 to 8191) of std_logic_vector(13 downto 0);
begin
    process(q1)
        variable prom: matriz;
        variable primeravez: std_logic:='1'; --indica si la eeprom está trabajando por
primera vez
        --variables para archivo.txt
        file arch: text;
        variable ptr: line;
        variable instru: std_logic_vector(13 downto 0);
        variable localidad: std_logic_vector(12 downto 0);
        begin
            if q1='1' then

                --Primera vez
                if primeravez='1' then
                    --carga la rom usando un archivo.txt (convertidor hex a txt)
                    localidad:="00000000000000";
                    file_open(arch,"C:\vhdl\cont_prog\progtxt.txt",read_mode);
                    while not (endfile(arch)) loop
                        readline(arch,ptr);
                        read(ptr,instru);
                        prom(conv_integer(localidad)):=instru;
                        localidad:=localidad+1;
                    end loop;
                    file_close(arch);
                    --la primera instrucción será la localidad 0x0000
                    binst<=prom(0);
                    primeravez:='0';
                end if;

                --Si la ALU no pide brinco, entrega la instrucción normalmente
                if flush='0' then
                    binst<=prom(conv_integer(dirpc));
                end if;
                --Si la ALU pide brinco, la sig. instrucción se ejecuta como NOP
                if flush='1' then
                    binst<="00000000000000"; --NOP
                end if;
            end if;
        end process;
    end compo;

```

## Integración de las entidades

Las entidades se interconectan como se ve en la figura 17. Las terminales de cada entidad se conectan conforme se fue indicando en la descripción independiente de cada entidad.

En algunas entidades es necesario colocar buffers a la entrada de sus terminales Q ya que algunas entidades dependen de la salida de otra entidad pero la ejecución de ambas es simultánea; el retraso de la entidad que recibe un dato asegura que en Q+1 el dato ya exista a la entrada.

Entidades con retrasos:

- Decodificador y control.

Q1 + retraso. Debe esperar a que la entidad EEPROM libere la siguiente instrucción.

- Generador de instrucciones.

Q1 + 2 retrasos. Espera a que la EEPROM libere una nueva instrucción. Posteriormente espera a que el decodificador le entregue ram\_ena.

- ALU.

Q2 + 2 retrasos. Espera a que el generador de direcciones ordene a la RAM escribir un dato en el bus. Luego esperar a que la RAM entregue el dato.

- Contador de programa.

Q2 + retraso. Espera a que el generador de direcciones le indique si debe escribir sobre el bus de datos.

Q3 + retraso. Espera a que la ALU entregue un resultado en el bus y si es necesario, lo copia a PCL o PCLATH.

Para la puesta a prueba de la entidad total se pondrán 2 ejemplos de programa. El primero será un programa que utiliza una tabla para calcular la potencia de un número almacenado en W y estará escrito en lenguaje ensamblador. El segundo programa será una multiplicación de 2 números de 8 bits y será escrito en lenguaje C.

Para la puesta en marcha del programa solo hizo falta implementar un estímulo a la simulación: OSCIN. Se generó un reloj de 4MHz (125ns en bajo y 125ns en alto).

### Programa Tabla en ensamblador.

```
                LIST P=16f84
                include "p16f84a.inc"

                org 0x0000
inicio          goto inicio2
                org 0x05
inicio2        movlw 0x03
                call tabla
                movlw 0x05
                call tabla
                movlw 0x09
```



Figura 17. Conexión de todas las entidades que conforman el PIC16.

```
call tabla
goto inicio2

tabla      addwf PCL,f
           retlw 0
           retlw 1
           retlw 4
           retlw 9
           retlw 16
           retlw 25
           retlw 36
           retlw 49
           retlw 64
           retlw 81
           end
```

El código en hexadecimal del mismo programa se muestra a continuación.

```
0x0000 : 0x2805 goto 0x05
0x0001 : 0x3FFF Data 0x3FFF ; ?
0x0002 : 0x3FFF Data 0x3FFF ; ?
0x0003 : 0x3FFF Data 0x3FFF ; ?
0x0004 : 0x3FFF Data 0x3FFF ; ?
0x0005 : 0x3003 movlw 0x03
0x0006 : 0x200C call 0x0C
0x0007 : 0x3005 movlw 0x05
0x0008 : 0x200C call 0x0C
0x0009 : 0x3009 movlw 0x09
0x000A : 0x200C call 0x0C
0x000B : 0x2805 goto 0x05
0x000C : 0x0782 addwf 0x02 , F
0x000D : 0x3400 retlw 0x00
0x000E : 0x3401 retlw 0x01 ; .
0x000F : 0x3404 retlw 0x04 ; .
0x0010 : 0x3409 retlw 0x09 ; .
0x0011 : 0x3416 retlw 0x16 ; .
0x0012 : 0x3425 retlw 0x25 ; %
0x0013 : 0x3436 retlw 0x36 ; 6
0x0014 : 0x3449 retlw 0x49 ; I
0x0015 : 0x3464 retlw 0x64 ; d
0x0016 : 0x3481 retlw 0x81 ; .
0x0017 : 0x3FFF Data 0x3FFF ; ?
0x0018 : 0x3FFF Data 0x3FFF ; ?
```

El diagrama de tiempos muestra que el resultado de las operaciones RETLW se ejecuta correctamente y que los saltos de CALL y GOTO ejecutan un NOP en el segundo ciclo de instrucción.

En el tiempo 0ns se tiene la instrucción 0x2805, GOTO 0x05. La ejecución salta a 0x04 y lo ejecuta como NOP (segundo ciclo en saltos) y posteriormente ejecuta el contenido de 0x05.

En el tiempo 16000ns, se tiene la instrucción 0x3481, RETLW 0x81. W vale 0x81 en Q4 y el programa regresa a donde fue llamada la instrucción más una localidad (RETLW no ejecuta segundo ciclo de instrucción como NOP por error de descripción, no porque el PIC lo haga de esa forma).

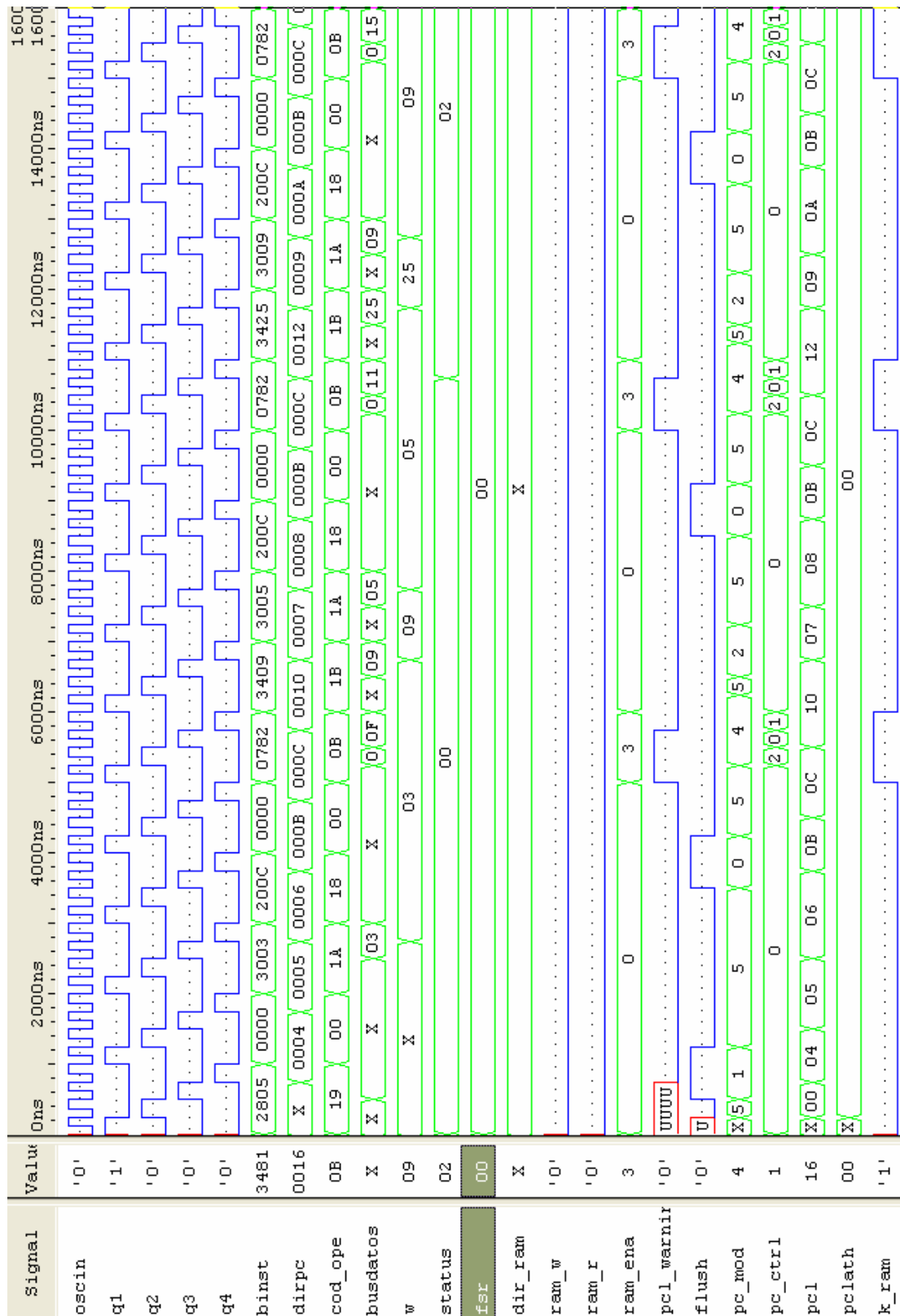


Figura 18. Ejecución del programa tabla de 0ns hasta 16000ns.





Figura 19. Ejecución de tabla de 16000ns a 32000ns.

### Programa multiplicación en lenguaje C.

```
#include "16f877a.h"

void main()
{
  int8 resultado=0;
  int8 mult=12;

  resultado=12*mult;

  //ahora para poder ver en la simulación de vhdl el resultado pongo
  //código en asm para pasar el valor a w
  while(1)
  {
    #asm
    movf 0x21,w
    #endasm
    //sé que resultado está en 0x21 porque lo indica el archivo multi.lst
  }
}
```

### Código generado en ensamblador a partir del lenguaje en C

```
CCS PCM C Compiler, Version 3.245, 28193                07-Dic-08 17:53

      Filename: C:\shared_s\programas\multi.lst

      ROM used: 66 words (1%)
                Largest free fragment is 2048
      RAM used: 7 (4%) at main() level
                9 (5%) worst case
      Stack:    1 locations

*
0000: MOVLW 00
0001: MOVWF 0A
0002: GOTO 02B
0003: NOP
..... #include "16f877a.h"
..... //Standard Header file for the PIC16F877A
device //
..... #device PIC16F877A
..... #list
.....
..... void main()
..... {
002B: CLRF 04
002C: MOVLW 1F
002D: ANDWF 03,F
002E: BSF 03.5
002F: BSF 1F.0
0030: BSF 1F.1
0031: BSF 1F.2
0032: BCF 1F.3
0033: MOVLW 07
0034: MOVWF 1C
..... int8 resultado=0;
0035: BCF 03.5
0036: CLRF 21
..... int8 mult=12;
0037: MOVLW 0C
```

```

0038: MOVWF 22
.....
..... resultado=12*mult;
0039: MOVWF 23
003A: MOVF 22,W
003B: MOVWF 24
003C: GOTO 004
003D: MOVF 78,W
003E: MOVWF 21
.....
..... //ahora para poder ver en la simulación de vhdl el resultado pongo
..... //código en asm para pasar el valor a w
..... while(1)
..... {
..... #asm
..... movf 0x21,w
003F: MOVF 21,W
..... #endasm
..... //sé que resultado está en 0x21 porque lo indica el archivo
multi.lst
..... }
0040: GOTO 03F
..... }
0041: SLEEP

```

En el código se calcula una multiplicación de  $12 \cdot 12$  y el resultado se deposita en la dirección ram 0x21. El resultado de  $12 \cdot 12 = 144$  equivale a 0x90 en hexadecimal.

Los diagramas de tiempo se muestran en las figuras 20, 21, 22, 23 y 24.

El método que utiliza el compilador para realizar la multiplicación es la comparación y rotaciones (RRF). El resultado de la multiplicación se pone en W y se entra a un ciclo sin fin.

El resultado ya está listo en W hasta poco antes de 66000ns, siendo 0x90 el contenido de W demostrando que la descripción es capaz de correr programas simples en lenguaje C.





Figura 21. Diagrama de ejecución del programa Multi.c de 16000ns a 32000ns.



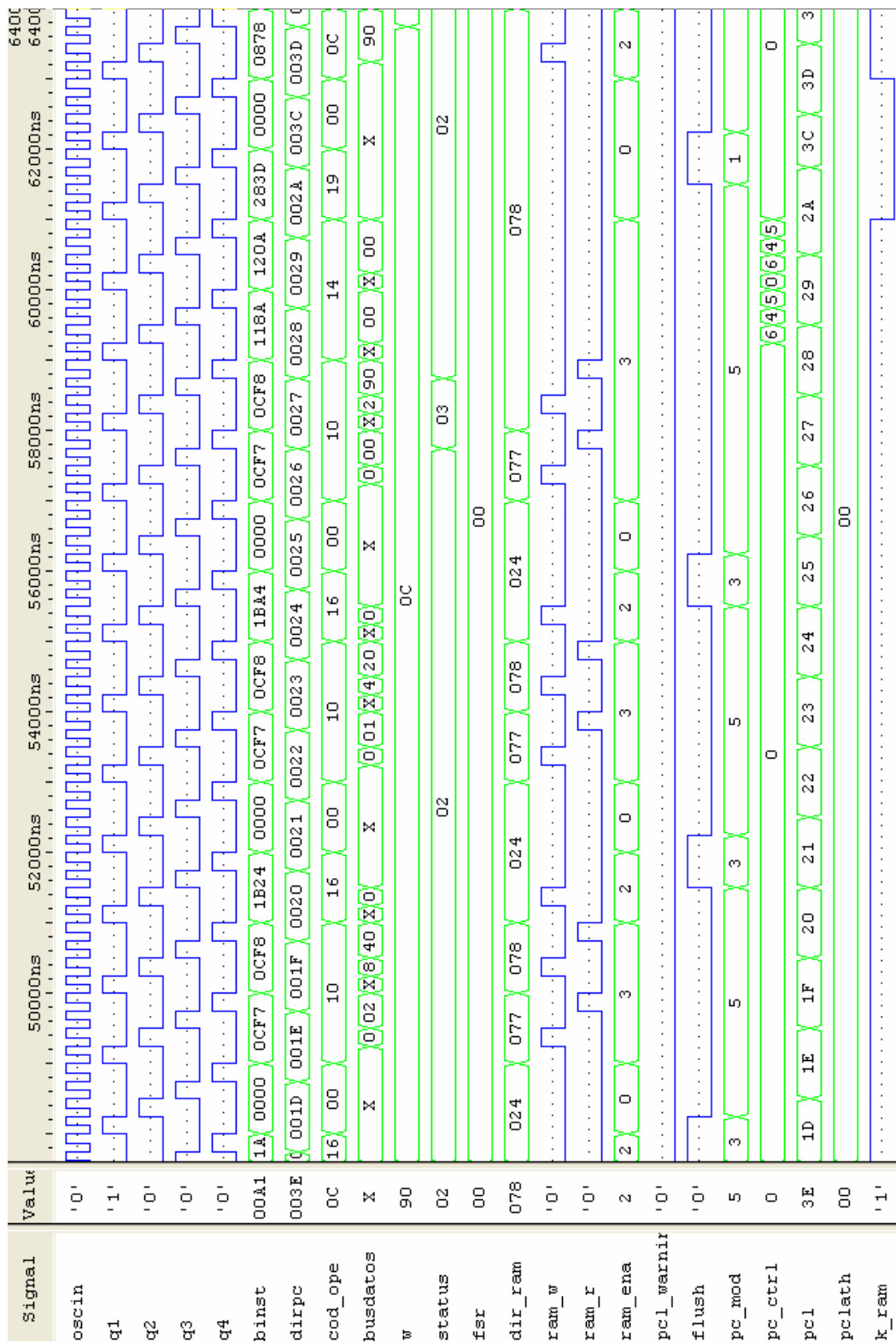


Figura 23. Diagrama de ejecución del programa Multi.c de 48000ns a 64000ns.



## Conclusiones

Se detectaron 3 errores o deficiencias en la descripción del modelo:

- El registro PCLATH no fue correctamente actualizado al leer datos del bus. Se desconoce el porqué de la falla. Esto implica una deficiencia de la descripción a hacer saltos de página de memoria de programa.
- Las instrucciones RETURN, RETLW y RETFIE no implementan un segundo ciclo de operación como flush. Esto se puede corregir fácilmente y no implica errores de ejecución del programa.
- La memoria RAM del PIC cuenta con registros mapeados en los 4 bancos o solo en 2 bancos. Los registros del banco 0 que van de 0x70 a 0x7f están mapeados en los 4 bancos de memoria. La entidad RAM no hace esto y genera errores de ejecución en programas que hacen uso efectivo de los mapeos.

Se intentó ejecutar el programa en C del cálculo de un logaritmo pero la falta de registros mapeados en la memoria RAM impidió la ejecución correcta del programa.

El segundo ciclo de instrucción que ejecutan los PICs al existir un salto en la memoria de programa puede ser fácilmente eliminado en una descripción de VHDL ya que no afecta el cálculo de datos y solo afecta el tiempo de ejecución.

La activación de entidades por medio de relojes Q abarata la construcción de las mismas ya que no deben ejecutarse en un solo ciclo de reloj y prolonga el tiempo de cambio de los estados de los transistores que las pueden componer. Si toda la ejecución se basara en un solo ciclo de reloj habría que implementar buffers por hardware con tiempo de cambio más rápidos que el mismo ciclo de instrucción para administrar el orden de las entidades.

El lenguaje VHDL es lo suficientemente poderoso para describir cualquier circuito digital que se tenga en mente. La simulación en diseño es suficiente como para definir la operación del circuito y la simulación en tiempo es necesaria para conocer los tiempos de cambio de los transistores que componen el circuito.