A Simple 9-bit Processor(I)

Figure 1 shows a digital system that contains a number of 9-bit registers, a multiplexer, an adder/subtracter unit, and a control unit(finite state machine). Data is input to this system via the 9-bit *DIN* input. This data can be loaded through the 9-bit wide multiplexer into the various registers, such as $R0, \ldots, R7$ and *A*. The multiplexer also allows data to be transferred from one register to another. The multiplexer's output wires are called a *bus* in the figure because this term is often used for wiring that allows data to be transferred from one location in a system to another.

Addition or subtraction is performed by using the multiplexer to first place one 9-bit number onto the bus wires and loading this number into register *A*. Once this is done, a second 9-bit number is placed onto the bus, the adder/subtracter unit performs the required operation, and the result is loaded into register *G*. The data in *G* can then be transferred to one of the other registers as required.

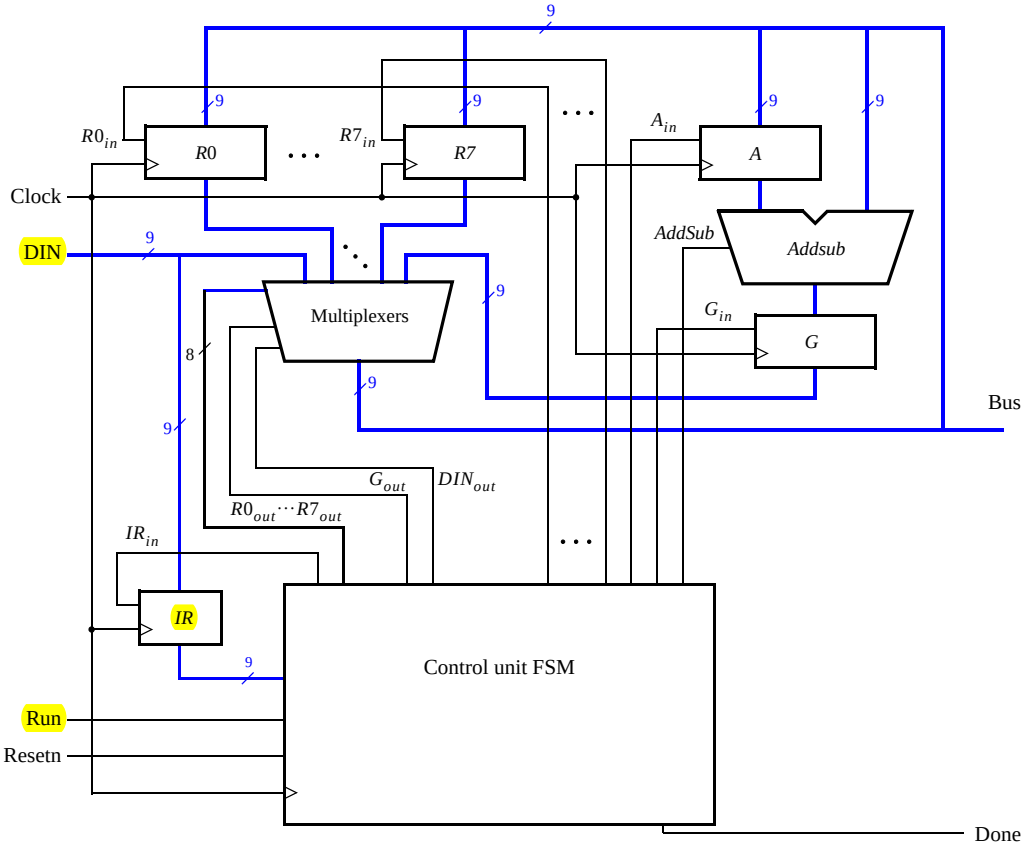

Figure 1: A digital system.

The system can perform different operations in each clock cycle, as governed by the *control unit*. This unit determines when particular data is placed onto the bus wires and it controls which of the registers is to be loaded with this data. For example, if the control unit asserts the signals $R0_{out}$ and A_{in}, then the multiplexer will place the contents of register $R0$ onto the bus and this data will be loaded by the next active clock edge into register *A*.

A system like this is often called a *processor*. It executes operations specified in the form of instructions.

Table 1 lists the instructions that the processor has to support for this exercise. The left column shows the name of an instruction and its operand. The meaning of the syntax RX ← [RY] is that the contents of register RY are loaded into register RX. The **mv** (move) instruction allows data to be copied from one register to another. For the **mvi** (move immediate) instruction the expression RX ← D indicates that the 9-bit constant D is loaded into register RX.

Operation	Function performed
mv Rx,Ry	$Rx \leftarrow [Ry]$
mvi Rx,#D	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Table 1. Instructions performed in the processor.

Each instruction can be encoded and stored in the *IR* register using the 9-bit format IIIXXXYYY, where III represents the instruction, XXX gives the RX register, and YYY gives the RY register. Although only two bits are needed to encode our four instructions, we are using three bits because other instructions will be added to the processor later. Hence *IR* has to be connected to the nine bits of the *DIN* input, as indicated in Figure 1. For the **mvi** instruction the YYY field has no meaning, and the immediate data #D has to be supplied on the 9-bit *DIN* input after the **mvi** instruction word is stored into *IR*.

Some instructions, such as an addition or subtraction, take more than one clock cycle to complete, because multiple transfers have to be performed across the bus. The finite state machine in the control unit "steps through" such instructions, asserting the control signals needed in successive clock cycles until the instruction has completed. The processor starts executing the instruction on the *DIN* input when the *Run* signal is asserted and the processor asserts the *Done* output when the instruction is finished. Table 2 indicates the control signals that can be asserted in each time step to implement the instructions in Table 1. Note that the only control signal asserted in time step 0 is IR_{in}, so this time step is not shown in the table.

	T_1	T_2	T_3
(**mv**): I_0	RY_{out}, RX_{in}, *Done*		
(**mvi**): I_1	DIN_{out}, RX_{in}, *Done*		
(**add**): I_2	RX_{out}, A_{in}	RY_{out}, G_{in}	G_{out}, RX_{in}, *Done*
(**sub**): I_3	RX_{out}, A_{in}	RY_{out}, G_{in}, *AddSub*	G_{out}, RX_{in}, *Done*

Table 2. Control signals asserted in each instruction/time step.

Design and implement the processor shown in Figure 1 using VHDL code as follows:

1. Generate the required VHDL file, include it in your project, and compile the circuit. A suggested skeleton of the VHDL code is shown in parts a and b of Figure 2, and some subcircuit entities that can be used in this code appear in Figure 2c.

2. Use functional simulation to verify that your code is correct. An example of the output produced by a functional simulation for a correctly-designed circuit is given in Figure 3. It shows the value $(040)_{16}$ being loaded into *IR* from *DIN* at time 30 ns. This pattern (the leftmost bits of *DIN* are connected to *IR*) represents the instruction **mvi** R0,#D, where the value $D = 5$ is loaded into $R0$ on the clock edge at 50 ns. The simulation then shows the instruction **mv** R1,R0 at 90 ns, **add** R0,R1 at 110 ns, and **sub** R0,R0 at 190 ns. Note that the simulation output shows *DIN* as a 3-digit hexadecimal number, and it shows the contents of *IR* as a 3-digit octal number.

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

ENTITY proc IS
   PORT ( DIN                 : IN     STD_LOGIC_VECTOR(8 DOWNTO 0);
          Resetn, Clock, Run  : IN     STD_LOGIC;
          Done                : BUFFER STD_LOGIC;
          BusWires            : BUFFER STD_LOGIC_VECTOR(8 DOWNTO 0));
END proc;

ARCHITECTURE Behavior OF proc IS
   . . . declare components
   . . . declare signals
   TYPE State_type IS (T0, T1, T2, T3);
   SIGNAL Tstep_Q, Tstep_D: State_type;
   . . .
BEGIN
   High <= '1';
   I <= IR(1 TO 3);
   decX: dec3to8 PORT MAP (IR(4 TO 6), High, Xreg);
   decY: dec3to8 PORT MAP (IR(7 TO 9), High, Yreg);
```

Figure 2a. Skeleton VHDL code for the processor.

```
    statetable: PROCESS (Tstep_Q, Run, Done)
    BEGIN
        CASE Tstep_Q IS
            WHEN T0 => IF(Run = '0') THEN Tstep_D <= T0;
                       ELSE Tstep_D <= T1;
                       END IF; -- data is loaded into IR in this time step
            . . . other states
        END CASE;
    END PROCESS;

    controlsignals: PROCESS (Tstep_Q, I, Xreg, Yreg)
    BEGIN
        . . . specify initial values
        CASE Tstep_Q IS
            WHEN T0 => -- store DIN in IR as long as Tstep_Q = 0
                IRin <= '1';
            WHEN T1 => -- define signals in time step T1
                CASE I IS
                    . . .
                END CASE;
            WHEN T2 => -- define signals in time step T2
                CASE I IS
                    . . .
                END CASE;
            WHEN T3 => -- define signals in time step T3
                CASE I IS
                    . . .
                END CASE;
        END CASE;
    END PROCESS;

    fsmflipflops: PROCESS (Clock, Resetn, Tstep_D)
    BEGIN
        . . .
    END PROCESS;

    reg_0: regn PORT MAP (BusWires, Rin(0), Clock, R0);
    . . . instantiate other registers and the adder/subtracter unit
    . . . define the bus
END Behavior;
```

Figure 2*b*. Skeleton VHDL code for the processor.

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

ENTITY dec3to8 IS
   PORT ( W     : IN     STD_LOGIC_VECTOR(2 DOWNTO 0);
          En    : IN     STD_LOGIC;
          Y     : OUT    STD_LOGIC_VECTOR(0 TO 7));
END dec3to8;

ARCHITECTURE Behavior OF dec3to8 IS
BEGIN
   PROCESS (W, En)
   BEGIN
      IF En = '1' THEN
         CASE W IS
            WHEN "000" => Y <= "10000000";
            WHEN "001" => Y <= "01000000";
            WHEN "010" => Y <= "00100000";
            WHEN "011" => Y <= "00010000";
            WHEN "100" => Y <= "00001000";
            WHEN "101" => Y <= "00000100";
            WHEN "110" => Y <= "00000010";
            WHEN "111" => Y <= "00000001";
         END CASE;
      ELSE
         Y <= "00000000";
      END IF;
   END PROCESS;
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regn IS
   GENERIC (n : INTEGER := 9);
   PORT ( R             : IN     STD_LOGIC_VECTOR(n-1 DOWNTO 0);
          Rin, Clock    : IN     STD_LOGIC;
          Q             : BUFFER STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END regn;

ARCHITECTURE Behavior OF regn IS
BEGIN
   PROCESS (Clock)
   BEGIN
      IF Clock'EVENT AND Clock = '1' THEN
         IF Rin = '1' THEN
            Q <= R;
         END IF;
      END IF;
   END PROCESS;
END Behavior;
```

Figure 2*c*. Subcircuit entities for use in the processor.

Figure 3: Simulation of the processor.