Pre Laboratory 5:

AN ENHANCED PROCESSOR

OBJECTIVES

- The purpose of this lab is to learn how to connect simple input (switches) and output devices (LEDs and 7-segment) to an FPGA chip and implement a circuit that uses these devices.
- Design an enhanced processor based on the simple processor in lab 4.

PREPARATION FOR LAB 5

- Students have to simulate all the exercises in Pre Lab 5 at home. All results (codes, waveform, RTL viewer, …) have to be captured and submitted to instructors prior to the lab session. *If not, students will not participate in the lab and be considered absent this session.*

REFERENCE

1. Intel FPGA training

Pre Laboratory 5:

AN ENHANCED PROCESSOR

Requirement: You will extend the capability of the processor so that the external counter is no longer needed. The processor has the ability to perform read and write operations using memory or other devices.

Instruction:

You will add three new types of instructions to the processor, as displayed in Table 1. The **ld** (load) instruction loads data into register RX from the external memory address specified in register RY. The **st** (store) instruction stores the data contained in register RX into the memory address found in RY. Finally, the instruction **mvnz** (move if not zero) allows a **mv** operation to be executed only under the condition that the current contents of register *G* are not equal to 0.

Operation	Function performed
ld Rx,$[Ry]$	$Rx \leftarrow [[Ry]]$
st Rx,$[Ry]$	$[Ry] \leftarrow [Rx]$
mvnz Rx, Ry	if G != 0, $Rx \leftarrow [Ry]$

Figure 1: New instructions performed in the processor.

A reference schematic of the enhanced processor is given in Figure 1. In this figure, registers *R*0 to *R*6 are the same as in Figure 1 of Laboratory 5, but register *R*7 has been changed to a counter. This counter is used to provide the addresses in the memory from which the processor's instructions are read; in the preceding lab exercise, a counter external to the processor was used for this purpose. We will refer to *R*7 as the processor's *program counter* (*PC*), because this terminology is common for real processors available in the industry. When the processor is reset, *PC* is set to address 0. At the start of each instruction (in time step 0) the contents of *PC* are used as an address to read an instruction from the memory. The instruction is stored in IR and the *PC* s automatically incremented to point to the next instruction (in the case of **mvi** the *PC* provides the address of the immediate data and is then incremented again).

The processor's control unit increments *PC* by using the *incr_PC* signal, which is just an enable on this counter. It is also possible to directly load an address into *PC* (*R*7) by having the processor execute a **mv** or **mvi** instruction in which the destination register is specified as *R*7. In this case

the control unit uses the signal *R7in* to perform a parallel load of the counter. In this way, the processor can execute instructions at any address in memory, as opposed to only being able to execute instructions that are stored in successive addresses. Similarly, the current contents of *PC* can be copied into another register by using a **mv** instruction.

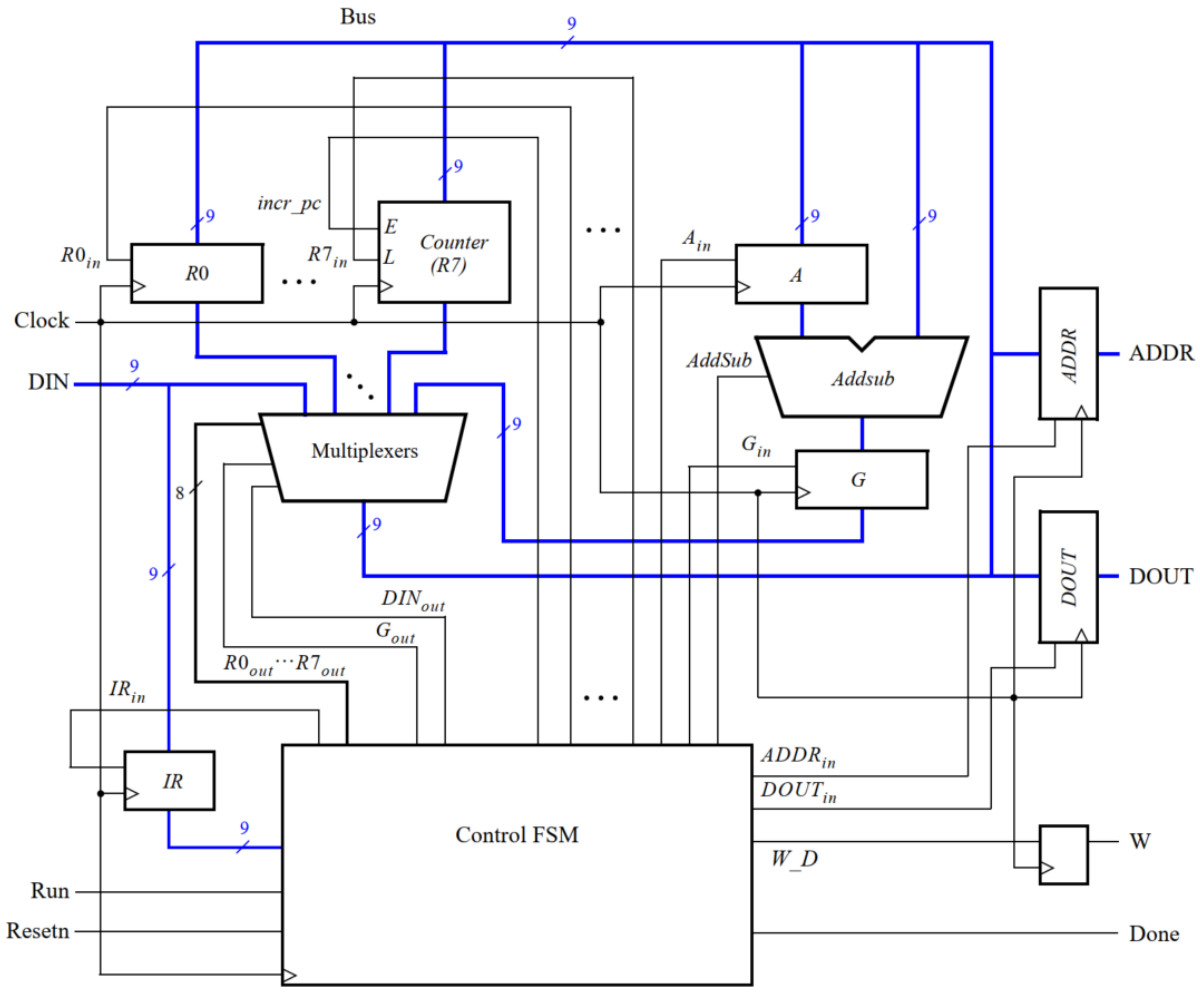


Figure 1: A reference schematic of the enhanced processor.

An example of code that uses the *PC* register to implement a loop is shown below, where the text after the % on each line is just a comment. The instruction **mv** R5,R7 places into *R5* the address in memory of the instruction **sub** R4,R2. Then, the instruction **mvnz** R7,R5 causes the **sub** instruction to be executed repeatedly until R4 becomes 0. This type of loop could be used in a larger program as a way of creating a delay.

```
mvi   R2,#1
mvi   R4,#10000000  % binary delay value
mv    R5,R7         % save address of next instruction
sub   R4,R2         % decrement delay count
mvnz  R7,R5         % continue subtracting until delay count gets to 0
```

Figure 1 also shows two registers in the processor that are used for data transfers. The *ADDR* register is used to send addresses to an external device, such as a memory module, and the *DOUT* register is used by the processor to provide data that can be stored outside the processor. One use of the *ADDR* register is for reading, or *fetching*, instructions from memory; when the processor wants to fetch an instruction, the contents of *PC* (*R7*) are transferred across the bus and loaded into *ADDR*. This address is provided to memory. In addition to fetching instructions, the processor can read data at any address by using the *ADDR* register. Both data and instructions are read into the processor on the *DIN* input port. The processor can write data for storage at an external address by placing this address into the *ADDR* register, placing the data to be stored into its *DOUT* register, and asserting the output of the *W* (write) flip-flop to 1.

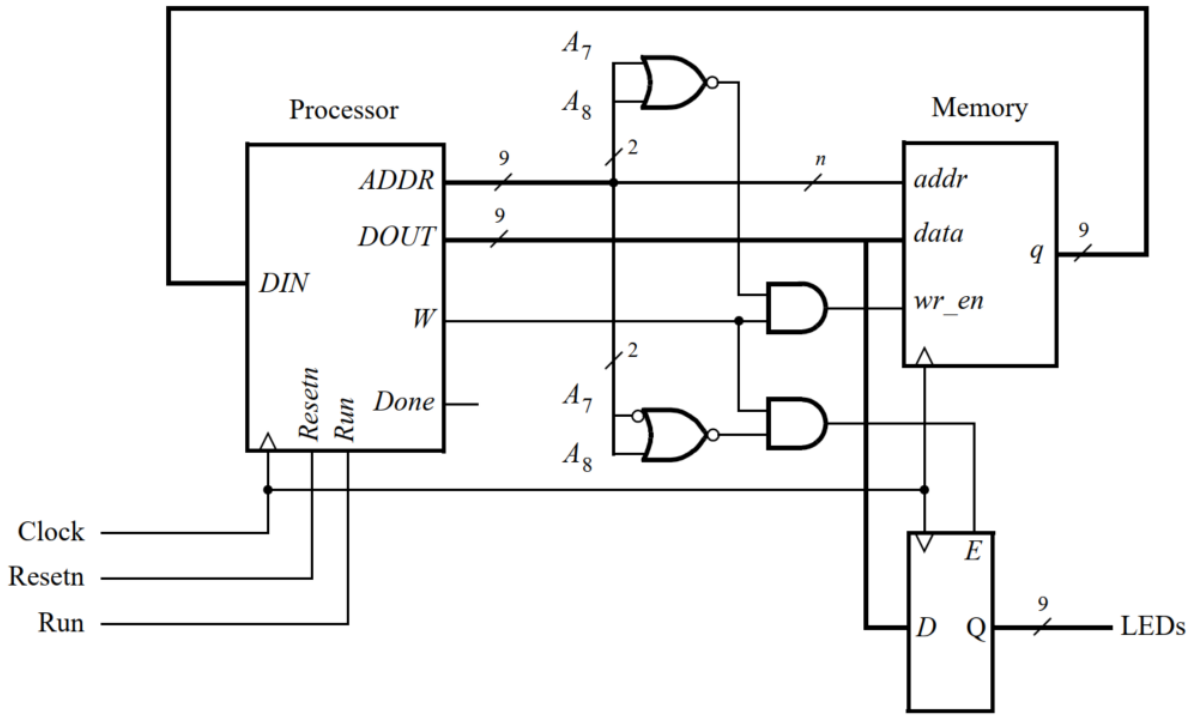


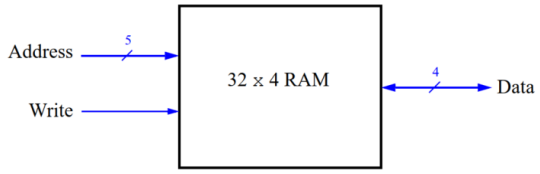
Figure 2: Connecting the enhanced processor to a memory and output register.

Pre Laboratory 5:

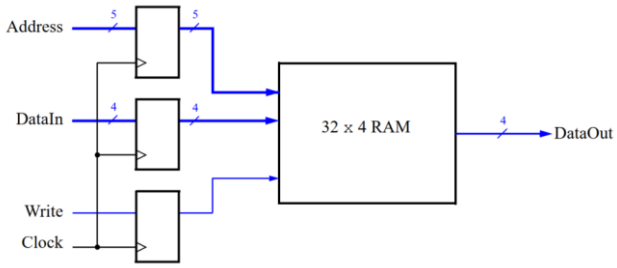
AN ENHANCED PROCESSOR

Figure 2 illustrates how the enhanced processor is connected to memory and other devices. The memory unit in the figure supports both read and write operations and therefore has both address and data inputs, as well as a write enable input. The memory also has a clock input, because the address, data, and write enable inputs must be loaded into the memory on an active clock edge. This type of memory unit is usually called a *synchronous static random access memory (synchronous SRAM)*. Figure 2 also includes a 9-bit register that can be used to store data from the processor; this register might be connected to a set of LEDs to allow display of data on your DE-10 board. To allow the processor to select either the memory unit or register when performing a write operation, the circuit includes some logic gates that perform *address decoding*: if the upper address lines are *A*8*A*7 = 00, then the memory module will be written at the address given on the lower address lines. Figure 2 shows *n* lower address lines connected to the memory; for this exercise a memory with 128 words is probably sufficient, which implies that *n* = 7 and the memory address port is driven by *A*6…*A*0. For addresses in which *A*8*A*7 = 01, the data written by the processor is loaded into the register whose outputs are called *LEDs* in Figure 2.

A diagram of the random access memory (RAM) module that we will implement is shown in Figure 3. It contains 32 four-bit words (rows), which are accessed using a five-bit *address* port, a four-bit *data* port, and a *write* control input.



(a) RAM organization



(b) RAM implementation

Figure 3: A 32 x 4 RAM module.