**3.4- Register File**

The register file has eight locations, each 8-bits wide. Three address lines, *rfaddr\_dp*2, *rfaddr\_dp*1, *rfaddr\_dp*0, are used to address the eight locations for both reading and writing. There are one read port and one write port. The read port is always active which means that it always has the value from the currently selected address location.

However, to write to the selected location, the write control line *rfwr\_dp* must be asserted before a value is written to the currently selected address location.

Note that a separate read and write address lines is not required because all the instructions either perform just a read from the register file or a write to the register file. There is no one instruction that performs both a read and a write to the register file. Hence, only one set of address lines is needed for determining both the read and write locations.

**3.5- ALU**

The ALU has eight operations implemented as defined by the following table. The operations are selected by the three select lines *alusel\_dp*2, *alusel\_dp*1, and *alusel\_dp*0.





The select lines are asserted by the corresponding ALU instructions as shown under the *Instruction* column in the above table. The pass through operation is used by all non-ALU instructions.

**3.6- Shifter / Rotator**

The Shifter has four operations implemented as defined by the following table. The operations are selected by the two select lines *shiftsel\_dp*1, and *shiftsel\_dp*0.



The select lines are asserted by the corresponding Shifter/Rotator instructions as shown under the *Instruction* column in the above table. The pass through operation is used by all non-Shifter/Rotator instructions.

**3.7- Output Buffer**

The output buffer is a register with an enable control signal connected to *outen\_dp*. Whenever the enable line is asserted, the output from the accumulator is stored into the buffer. The value stored in the output buffer is used as the output for the computer and is always available. The enable line is asserted either by the OUT A instruction or by the system reset signal.

**3.8- Control Word**

From Figure 2, we see that the control word for this custom datapath has fourteen bits, which maps to the control signals for the different datapath components. These fourteen control signals are summarized in Figure 3.



**Figure 3**. Control word signals for the datapath

By now, you should be able to trace through this datapath and see how it executes for each of the instructions. For example, to execute the ADD instruction, which adds the content of the accumulator with the content of the specified register file location and writes the result back into the accumulator, the value in the accumulator is passed to the A operand of the ALU. The B operand of the ALU comes from the register file, the location of which is selected from setting the register file address lines *rfaddr\_dp*2,1,0.

The appropriate ALU select lines *alusel\_dp*2,1,0 are set to select the ADD operation. The shifter is not needed and so the pass through operation is selected. The output of the shifter is routed back through input 0 of the multiplexer and finally written back to the accumulator.

So the control word for the instruction

**ADD A, 011**

is



**4- Control Unit**

The finite state machine for the control unit basically cycles through four main states: reset, fetch, decode, and execute, as shown in Figure 4. There is one execute state for each instruction in the instruction set.

**4.1- Reset**

The finite state machine starts executing from the reset state when the reset signal is asserted. On reset, the finite state machine initializes all its working variables and control signals. The variables include:

• PC – program counter

• IR – instruction register

• State – the state variable

In addition, the content of the memory, i.e., the program for the computer to execute is also loaded at this time.



**Figure 4**. State diagram for the control unit.

**4.2- Fetch**

In the fetch state, the memory content of the location pointed to by the PC is loaded into the instruction register. The PC is then incremented by one to prepare it for fetching the next instruction. If the fetched instruction is a jump instruction, then the PC will be changed accordingly during the execution phase.

**4.3- Decode**

The content that is stored in the instruction register is decoded according to the encoding that is assigned to the instructions as listed in Figure 1. As a result, the instruction will be executed starting at the beginning of the next clock cycle when the FSM enters this new state.

**4.4- Execute**

The execution state simply sets up the control word, which asserts the appropriate control signals for the datapath to carry out the necessary operations for executing a particular instruction. Each instruction, therefore, has its own execute state. For example, the execute state for the add instruction ADD A, 011 will set up the following control word.



For all the jump instructions, no actions need to be taken by the datapath. It simply determines whether to perform the jump or not depending on the particular jump instruction and by checking on the zero and positive flags. If a jump is needed then the target address is calculated and then assigned to the PC.

At the end of the execute state, the FSM goes back to the fetch state and the cycle repeats for the next instruction.

**5- Top-level Computer**

In order to actually test out our custom general microprocessor, we need to connect it to the three basic components as defined in the Von Neuman architecture of a computer system, namely, an input, an output and a memory.

**5.1- Input**

Our computer input consists of eight simple dip switches for binary input of a number.

**5.2- Output**

Our computer output consists of two 7-segment LEDs. The 8-bit output from the CPU datapath is decoded so that the eight bit binary value is displayed as two decimal digits on the two 7-segment LEDs.

**5.3- Memory**

**5.4- Clock**

For our system clock, we use the built-in 25MHz clock that is available on the development board. In order to see some intermediate actions by the CPU, we have used a clock divider to slow down the clock.

**6- Examples**

In this example, we will implement the multiplication program on our CPU. Figure 6(a) shows the disassembled code for the multiplication program.



**Figure 5**. Connections between the datapath and the control unit for our general microprocessor.



**Figure 6**. Multiplication of 13 × 11: (a) multiplication program;