Small8 Microprocessor

EL 4712 – Spring 2016

Objective:

The objective of this “mini-project” is to design, simulate, and implement a very simple computer, called Small8. Small8 consists of an 8-bit processor (with a 64K address space), a RAM, and I/O ports. *Note: some of the details are intentionally omitted*. You must use what you have learned throughout the semester to complete the project. You are free to implement the Small8 in VHDL any way that you like, as long as it can execute the provided test programs.

Other related files: (available on the EEL4712 Web site)

- Small8InstructionSetPage1.pdf and Small8InstructionsAddendum.pdf
- TestPackage.zip: contains a set of test programs and programs required to assemble an assembly source code program into a .mif file.
  - TestCase1.asm, TestCase1.mif – source code and .mif file to test LDAA, STAA, STAR, ANDR, ADCR, BEQA.
  - TestCase2.asm, TestCase2.mif – source code and .mif to test LDAI, CLRC, RORC, DECA, BNE.
  - TestCase3.asm – source code to test index addressing. You have to obtain the .mif file yourself.
  - mult.asm, mult.mif – a comprehensive test program (multiplication) in source code and .mif file.

Logistics:

As discussed in class, this is essentially a “mini-project”. It will be worth 350 points (3.5x more than a normal lab). The grading is based on the completion of a list of deliverables. When completed, each deliverable will earn the student some amount of points (toward the 350 total points). The list of deliverables, their due dates, and their worth in points will be described later.

General architecture for the Small8 computer:

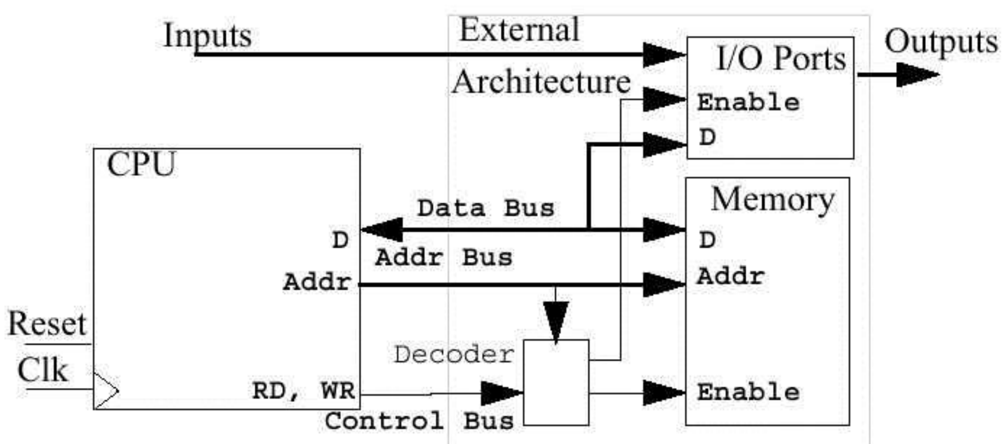


Figure 1. Overall architecture of the Small8 computer.

The Small8 computer consists of the following:

- An 8-bit processor (CPU) with 8-bit data registers and data bus and a 64K address space (16-bit address bus).
- A memory module
- Two 8-bit input ports and output ports, with the following addresses. The output ports connect to two separate 7-segment LEDs.

INPORT0	$FFFE	INPORT1	$FFFF	e.g., LDAA $FFFE means A ← (INPORT0)
OUTPORT0	$FFFE	OUTPORT1	$FFFF	e.g., STAA $FFFE means OUTPORT0 ← (A)

- Because the DE0 board does not have 16 switches, each input port will share the same 8 switches. To load a value into each port, you will use two buttons as enable signals for the input ports. In other words, you would set the switches for the desired value on inport 0, then press the enable button for inport 0. You would then change the switches for inport 1, and then press a second button to enable inport 1.
- A separate reset, controlled by the third button, for the CPU and memory. Note that this reset should *not* reset the input ports. This separate reset is used to restart an application after changing the values of the input ports.

General architecture for the Small8 CPU:

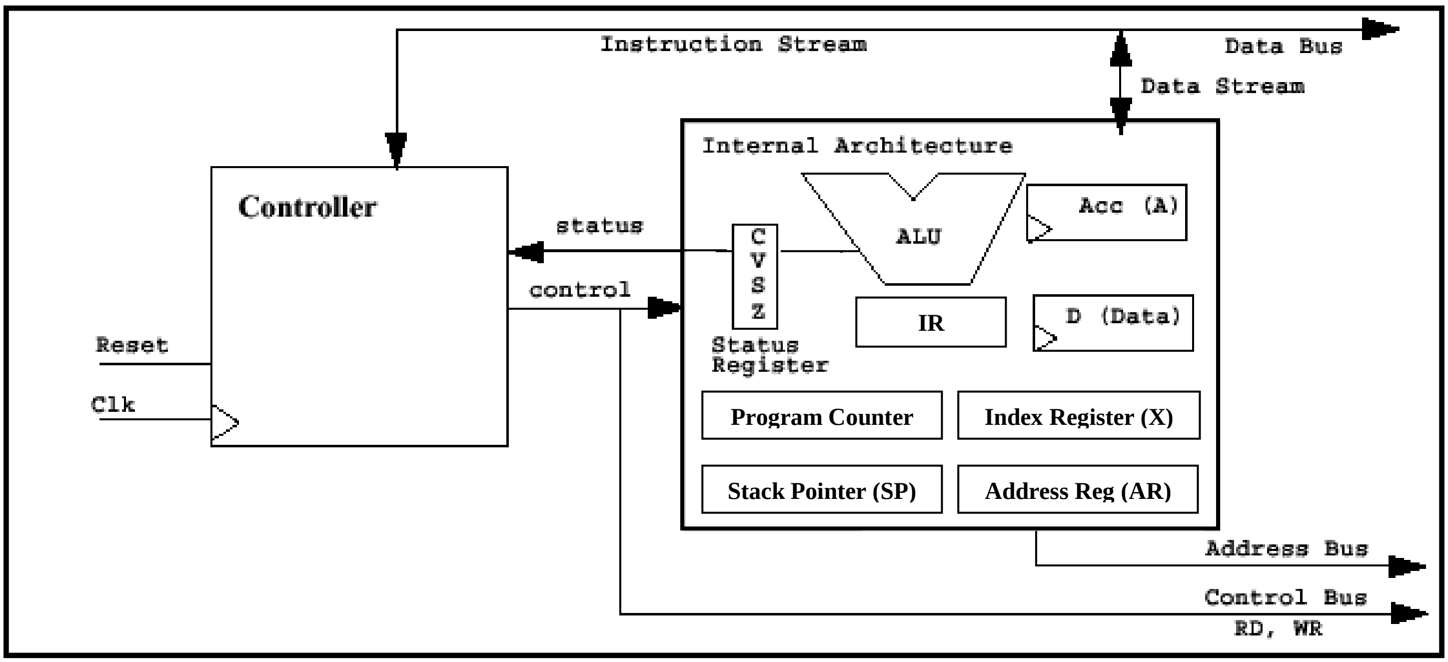
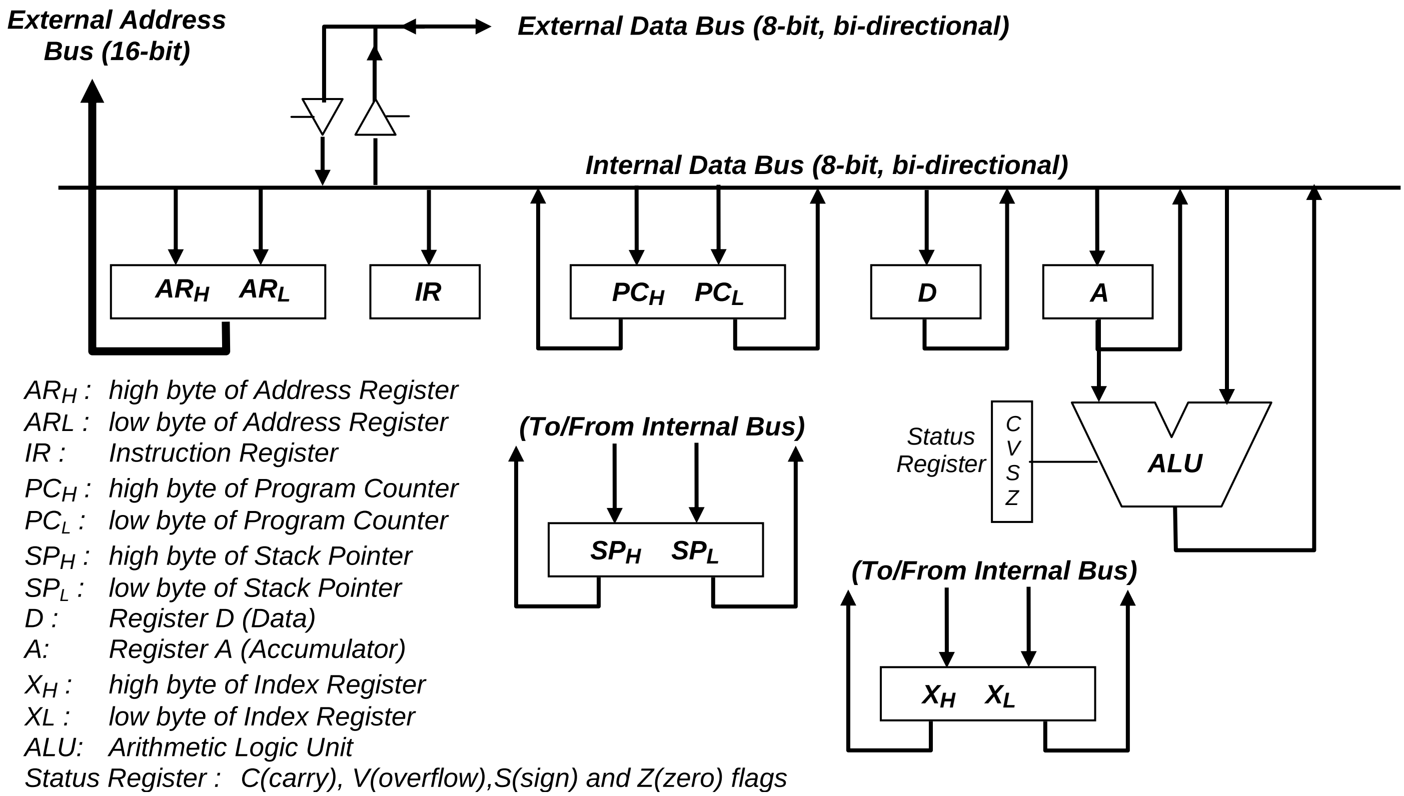


Figure 2. General architecture of the Small8 CPU.

The Small8 CPU is an 8-bit processor consisting of:

- A set of components comprising the “Internal Architecture”, which corresponds to a datapath as discussed in class:
  - An ALU (arithmetic/logic unit) with the associated Status Register of four flags (carry C, overflow V, sign S, and zero Z).
  - Two general-purpose registers: Accumulator (A) and Data (D) which are viewed by the programmer as persistent and should only change if an instruction specifically changes them.
  - Some special-purpose registers, including Index Register (X), Program Counter (PC), Address Register (AR), Instruction Register (IR), and Stack Pointer (SP).
- A Controller which controls both the components of the Internal Architecture and the external components such as the memory module and the I/O ports. The design of the Controller is one of the main tasks of this project.
  - The Controller controls the components of the Internal Architecture via the control signals as shown in Figure 2. Examples of these control signals include PC.INC, A.LD, AR.LD, D.OE, etc.
  - The Controller interacts with the external components (memory and I/O ports) via three buses: 16-bit address bus, 8-bit bi-directional data bus, and a control bus.



* Additional (temp) register(s) and logic may be added as necessary to the architecture.

Figure 3. Preliminary design for Internal Architecture of the Small8 CPU.

A preliminary design for the Internal Architecture of the Small8 CPU is shown in Figure 3. The components of the Internal Architecture are interconnected through a bi-directional internal data bus designed to move data among components in a flexible and efficient manner. The definitions of the components are as follows:

- A and D: The accumulator (A) and data (D) registers are 8-bit general-purpose registers used for data storage and computation.
- X is a 16-bit index register used for index addressing mode of the LDAA and STAA instructions.
- ALU: The ALU performs all the necessary arithmetic/logic/shift operations required to implement the Small8 instruction set. The Status Register consists of 4 flags that are generated from the ALU:
  - C is the carry flag, the carry out of the ALU after an addition (or subtraction) operation.
  - V is the overflow flag out of the ALU. For two's complement arithmetic, V is the exclusive OR of the two most significant bits of the carries (cout XOR c7). *See Section 5.3.5 in the textbook for details. This will also be discussed in class*.
  - S is the sign flag, the most significant bit of the ALU result.
  - Z is the zero flag. Z is '1' when the ALU result is all zeros.
- PC: The Program Counter (PC) is a 16-bit register that contains the memory address of the next instruction to be executed.
- AR: The Address Register (AR) is a 16-bit register used to hold the address of the memory location to be read or written.
- SP: The Stack Pointer (SP) is a 16-bit register to store the pointers to the return addresses of a subroutine call.
- IR: The Instruction Register (IR) holds the instruction once it is fetched from memory.

Let me stress that this is a ***preliminary*** design. You can modify it in order to perform the required operations to execute the instruction set and/or to increase the flexibility and efficiency if necessary. For example, one or more temporary registers may need to be added. Or, since the Opcode fetch is such an important operation, it may be best to connect the Program Counter directly to the external address bus. Be aware that any changes you make to the datapath will likely also require corresponding changes to the controller.

Opcode fetch, decode, execute cycle for the CPU controller:

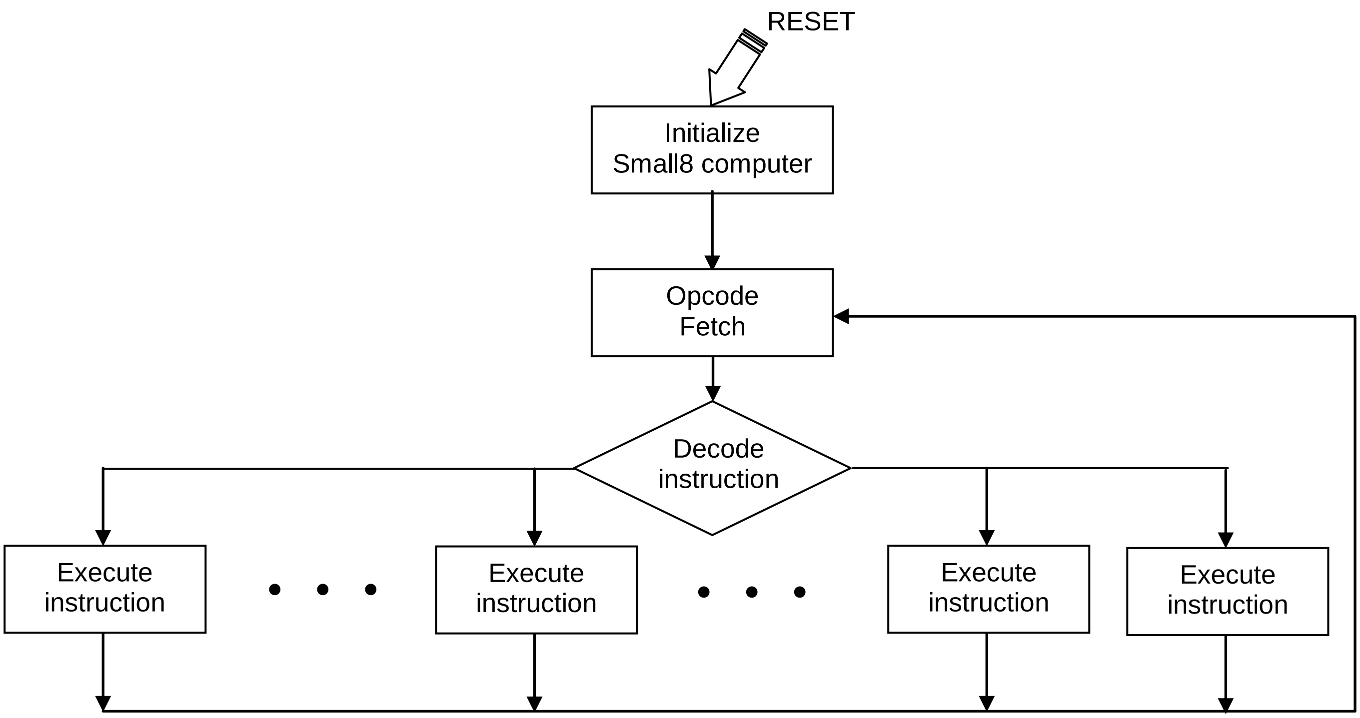


Figure 4. General algorithm for the CPU controller.

- You are to develop a detailed finite state machine for the CPU controller based on the algorithm in Figure 4 and the detailed design of the datapath (“internal architecture”) components you plan to use for Small8.
- Although the execution of each instruction is shown in Figure 4 as conceptually requiring a separate “path” in the finite state machine, your FSM should be optimized by using shared states whenever possible. Hint: similar types of instructions will have similar control requirements. For example, for the “and”, “or”, and “xor” instructions, the only difference in the control will be the select value for the ALU. If you can extract the select value from the opcode, you can use the same FSM states for each of these instructions.
- Also use conditional outputs when possible to reduce the number of states and improve performance.
- You need to implement the instructions used in the test case programs and mult.asm.

Deliverables: (prepare to show to your TA)

For each deliverable, do the following:

- Create a neat drawing of your circuit, or a finite state machine for the controller..
- Submit your VHDL files on e-learning.

- **<u>Have simulations prepared to demo the correct functionality.</u>** These simulations should make it easy to see the functionality of each deliverable. Add annotations to explain. For larger simulations (e.g., multiply test case), selectively show some key parts of the waveform. <u>Turn in these simulations on e-learning along with your code.</u>
- On e-learning, there will be a submission link for each week's deliverables. I'd suggest creating a separate folder for each deliverable to make it easy to find your code. If you work ahead, turn in the deliverables in the specified weeks (not the week you finished it).

<u>Part of the grading of the deliverable is your understanding/explanation of your design</u>. Of course, blatant inability to explain your finite state machine and/or your code is evidence of cheating and will be dealt with as such.

NOTE: You must attend lab each week unless you have demoed all deliverables. Missing a lab will result in -20 points. Unless you are completely finished, you have to stay and work on the project with the help of your TA.

<u>Week 1</u>: At a minimum, you are to complete Deliverables 1 and 2 by the end of the lab.

Deliverable 1 (15 points): Design and simulation of the ALU with the 4 flags. No demonstration on the UF-4712 board is necessary. Show the TA a simulation waveform that shows the correct operation of each operation and the correct operation of each of the flags. Turn in all files and the simulation on e-learning.

Extra Credit (10 points): create an exhaustive testbench that tests every possible input combination using assert statements and show the TA that no assertions fail.

Deliverable 2 (20 points): Design and simulation of the datapath ("internal architecture") and ports, including both the internal and external buses. You must illustrate and explain to the TA the operation of each control signal that you are using for the datapath. At a minimum, you must show each component reading and writing data to/from the bus. Turn in all files and the simulation on e-learning.

<u>Week 2:</u> At a minimum, you are to complete Deliverable 3 by the end of the lab.

Deliverable 3 (20 points): Design and simulation of the basic opcode fetch cycle, which will require you to connect the controller to the datapath and RAM. To demonstrate, you do not need to execute the fetched instructions, but you must show the IR for consecutive instructions, assuming they execute sequentially. Branch instructions will not be tested because they require non-sequential execution. Note that different instructions will increase the program counter by different amounts. Turn in all files and the simulation on e-learning.

<u>Week 3:</u> Turn in all files and the simulations on e-learning for each of the deliverables.

Deliverable 4 (140 points): Simulation and demonstration (on the UF-4712 board) of the following test case programs:

- TestCase 1 – tests LDAA, STAA, STAR, ANDR, ADCR, CLRC, BEQA, BCCA **(% of 55 points)**.
- TestCase 2 – new instructions to be tested: LDAI, RORC, DECA, BNEA **(% of 30 points)**.
- TestCase 3 – tests index addressing **(% of 55 points)**. Hand-assemble TestCase3.asm and create a .mif file. Use the .mif file to test index addressing.

Deliverable 5 (35 points): Make any of the above programs a subroutine and demonstrate the calling to and return from the subroutine.

Deliverable 6a (% of 100 points): Simulation and demonstration (on the UF-4712 board) of the execution of a comprehensive test program (mult.mif).

- The multiplicand and multiplier is currently "hardcoded" in the given program using dc.s commands. You should change the program to input the operands from the input ports (i.e., switches).
- The result will be displayed the 7-segment displays connected to OUTPORT0 and OUTPORT1.

Deliverable 6b (% of 20 points):

- Implements a MULTIPLY instruction
- Function: AD <= A * D; The answer (i.e., 16-bit product) is split up with A containing the higher-order byte and D containing the lower-order byte. The original contents of A and D are displaced.
- Write a test case program (.mif) to demonstrate it.
- You must use a Cyclone II embedded-multiplier component to implement this instruction. This component is inferred if you use the * operator.

Special Notes:

- You only need to implement the instructions used in the four TestCase programs and mult.asm.

Memory Map and Programming Model

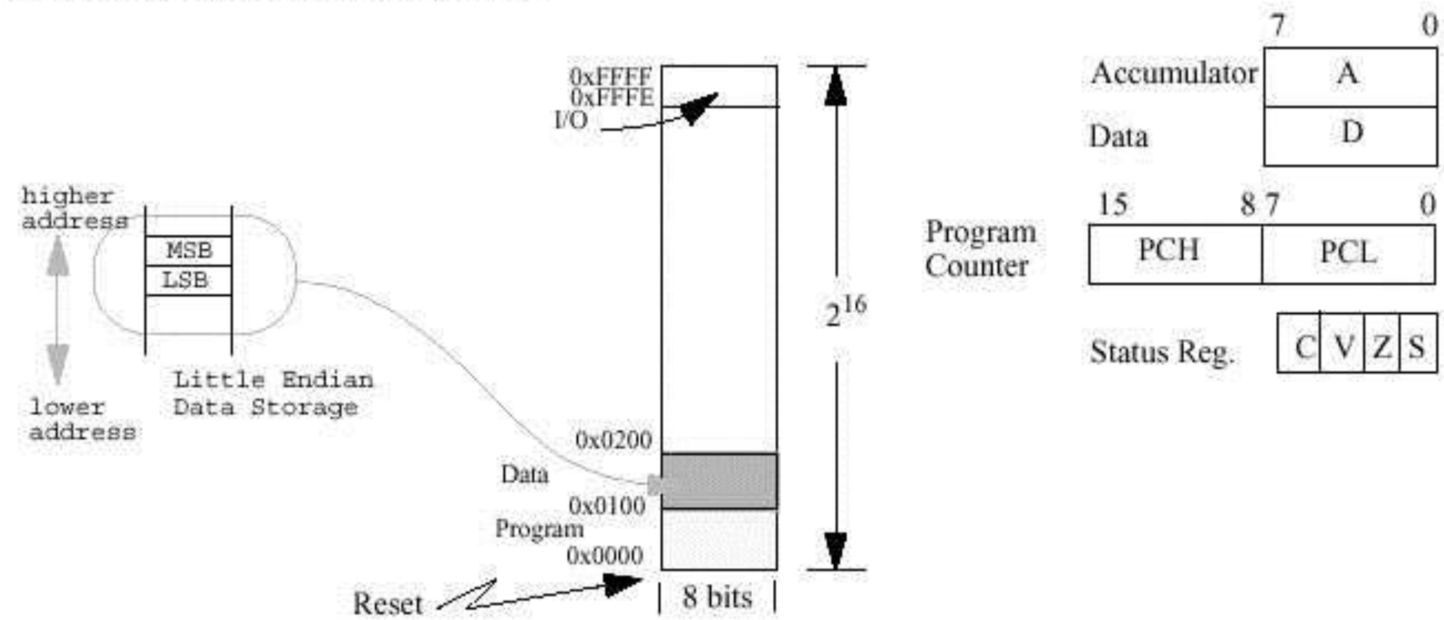


Figure 5. Memory map and programming model of the Small8 computer.

Machine Instruction Format:

- Arithmetic, logic, shift

7 0
opcode

- Load immediate

7 0	7 0
opcode	data

- Load, store, branch (absolute address mode)

7 0	7 0	7 0
opcode	addr LSB	addr MSB