

VHDL PROJECTS TO REINFORCE COMPUTER ARCHITECTURE CLASSROOM INSTRUCTION

Ronald J. Hayne
Department of Electrical and Computer Engineering
The Citadel
171 Moultrie Street
Charleston, SC 29409

Abstract

Exploration of various computer architecture constructs needs reinforcement beyond pencil and paper homework problems. Unfortunately, laboratory exercises based on microprocessor trainers are limited to a single architecture and a resolution of single assembly language instructions. A hardware description language, such as VHDL, can be used to provide simulation-based application of the classroom instruction regardless of the course text. Models of computer components such as registers, memory, and ALUs can be readily defined to match textbook examples and then combined to demonstrate multiple architectural concepts. Students with basic knowledge of VHDL from their prerequisite digital logic course are able to modify and use these models to simulate computer behavior at the register transfer level with data and control signal visibility at each clock cycle.

A program of instruction has been developed that uses VHDL homework exercises and a capstone design project to provide hands-on application of course concepts using modern design tools. Exercises include addressing modes, microprogrammed control, and computer arithmetic. The design project models a multi-bus architecture and hardwired control unit from the text to implement a basic instruction set. An example assembly language program can be loaded into memory and executed in simulation on the model computer. Results are verified by monitoring control signals, buses, and registers, as well as final dumps of memory and register contents.

Student feedback has been very positive that the VHDL exercises provided reinforcement of classroom concepts and allowed them to visualize results via simulation. Over two-thirds of the students were able to implement a completely functional design project computer model which successfully executed the test program. The methodology was to combine/modify instructor provided VHDL models, rather than turning the course into a “programming” class. This kept the focus on “hardware description” and did not rely on coverage of the language by the course text. The resulting combination of textbook and classroom instruction with VHDL modeling and simulation exercises provided students with a more robust learning experience and exposure to state-of-the-art design tools.

Introduction

Teaching computer architecture effectively requires reinforcement beyond pencil and paper homework problems. Some form of hands-on laboratory exercises are desirable to provide practical observation of the classroom constructs. Microprocessor trainers are one option, but unless investment is made in multiple systems, exercises are limited to a single architecture. With a resolution of single assembly language instructions, the focus often shifts to assembly language programming of the target architecture, rather than allowing exploration of various computer architecture concepts.

Hardware description languages such as VHDL provide another alternative for laboratory exercises to supplement classroom instruction. Models can be developed for

multiple computer architectures and their function can be observed through simulation. VHDL modeling techniques range from structural interconnection of hardware components to abstract behavioral models with little relation to the underlying hardware. Many texts have been written to teach VHDL, but their focus is most often on the language and its application to large-scale digital systems.

The approach taken here was to develop a set of VHDL exercises to supplement an existing course for teaching computer architecture. The current course text, *Computer Organization*[1], contains descriptions of hardware for multiple computer architectures, but no coverage of VHDL or other models of that hardware. A program of instruction has been developed which augments the classroom instruction with VHDL homework exercises and a capstone design project. The focus is kept on demonstration of architectural concepts, rather than turning into a VHDL “programming” course.

Course Content

The existing course and text cover a wide range of computer architecture constructs which can be supplemented with homework exercises. A fundamental concept is the discussion of various addressing modes used to specify the location of operands in an assembly language program. These addressing modes are first discussed generically and then specifically for several modern computer architectures. Visualization of the addresses and operands is often difficult for students, so an exercise was developed to provide a VHDL simulation to demonstrate the various results.

Another important concept is that of microprogrammed control of the operation of a simple processor. Several control schemes are discussed along with the basis for forming control words. Another homework exercise has been developed to allow the students hands-on application of these concepts. The computer architecture from the addressing mode exercise

is coupled with a VHDL model of a microprogram controller which generates the control signals from student coded microinstructions.

Additional exercises are provided to demonstrate control signals and data flow for various processor architectures. Of note is the discussion of computer arithmetic and the implementation of two’s complement multiplication via the Booth algorithm, which will not be covered in this paper. Hardware models from the text are implemented in VHDL, again allowing the students simulation-based application of the course concepts.

Many of the course concepts are tied together in a capstone design project which implements a simple processor using a multi-bus architecture and a hardwired control unit. The instruction set from previous homework exercises is supplemented with additional instructions and used to code an assembly language program which has been used as a design example throughout the course. The resulting simulation provides a functional model of processor with data and control signal visibility at each clock cycle.

Single-bus Architecture

Addressing mode and microprogrammed control exercises are based on a single-bus architecture from the course text, shown in Figure 1. Processor registers and memory are defined along with control signals and the datapath to create the resulting architecture is shown in Figure 2.

Addressing Modes

For the addressing mode homework exercise, a simple instruction format was created based on classroom and text examples. The instruction format, shown in Figure 3, is robust enough to demonstrate all the fundamental addressing modes and provides the basis for future exercises and the design project.

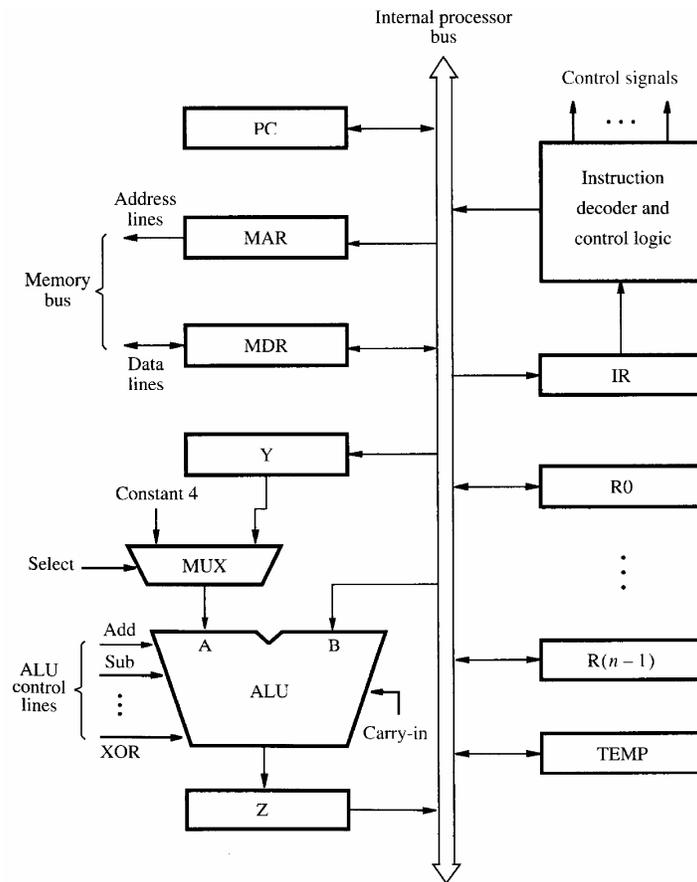


Figure 1: Single-bus architecture from text.

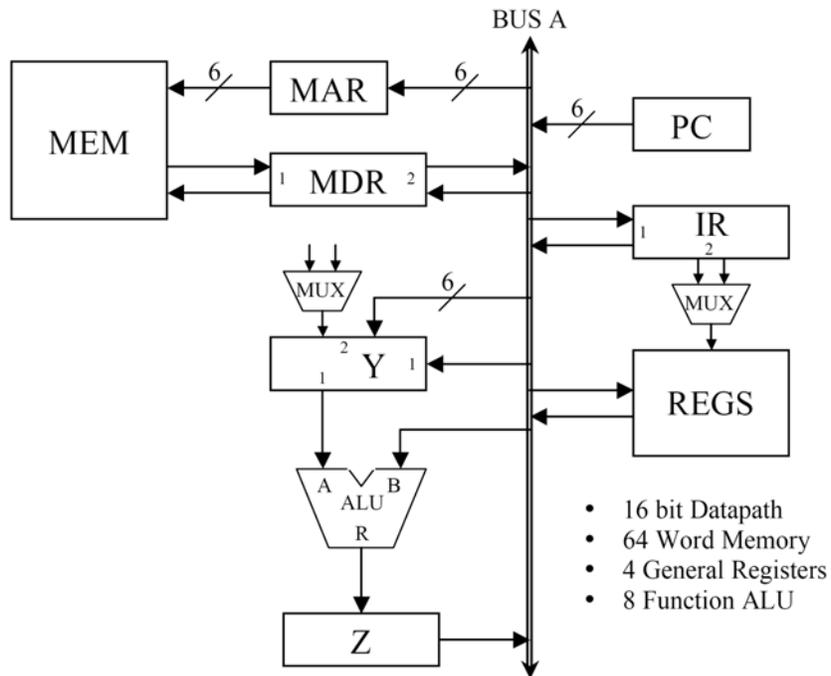


Figure 2: Single-bus architecture for homework.

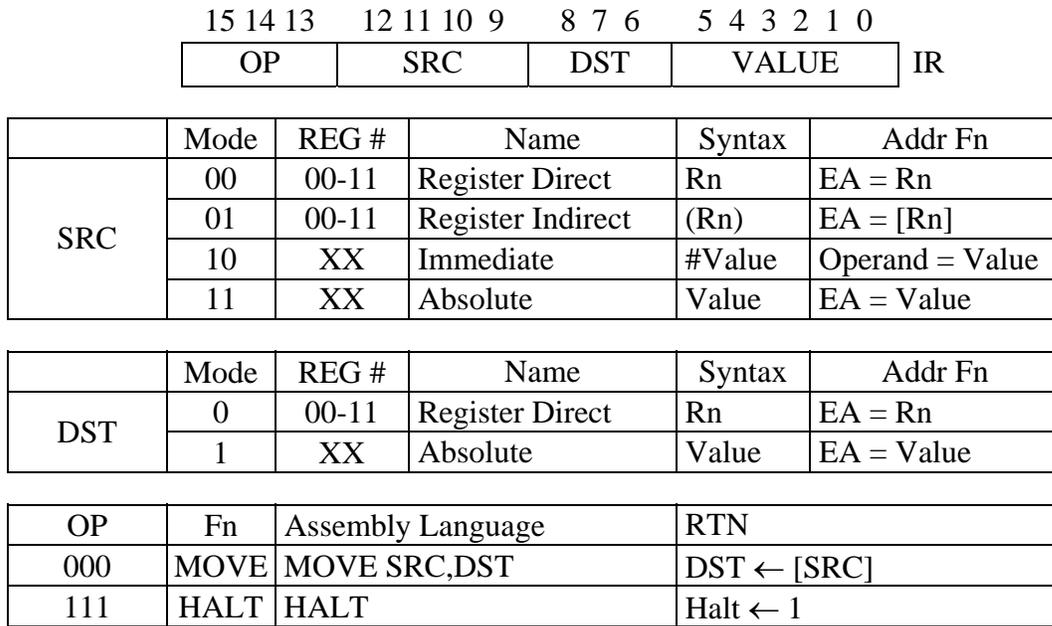


Figure 3: Simple instruction format.

An assembly language program, shown in Figure 4, is used to test the various addressing modes and provides the students with hands-on application of the classroom concepts. The resulting contents of the registers and memory must be determined and then verified by simulation.

```

MOVE #NUM2 ,R0
MOVE NUM1 ,R1
MOVE ( R0 ) ,R2
MOVE R0 ,NUM3
MOVE #-1 ,R3
MOVE ( R1 ) ,NUM4
HALT

```

Figure 4: Assembly language program.

VHDL Components

VHDL models of the various components of the architecture are provided to the students as a package. The students with basic knowledge of VHDL from their prerequisite digital logic course can use these models to create simulations of the architectural concepts. The most basic is the register, shown in Figure 5, which contains the data and control signals necessary to model the hardware component.

The key to successful simulation of various processor architectures and instruction sets is a memory model which can be programmed with machine code. This is accomplished using a VHDL generic which allows an initial contents to be set at time of instantiation. A memory dump feature has also been added to provide for easy verification of final memory contents after program execution. The VHDL memory model is shown in Figure 6 and a sample initialization is shown in Figure 7.

```

entity REG16 is
  port(CLK          : in std_logic;
        Enable_In   : in std_logic;
        DATA_IN    : in std_logic_vector(15 downto 0);
        Enable_Out  : in std_logic;
        DATA_OUT   : out std_logic_vector(15 downto 0));
end REG16;

architecture BEHAVE of REG16 is
  signal DATA : std_logic_vector(15 downto 0);
begin
  input : process
  begin
    wait until (CLK = '1' and not CLK'STABLE);
    if Enable_In = '1' then
      DATA <= DATA_IN;
    end if;
  end process;
  output : process(Enable_Out, DATA)
  begin
    if Enable_Out = '1' then
      DATA_OUT <= DATA;
    else
      DATA_OUT <= (others => 'Z');
    end if;
  end process;
end BEHAVE;

```

Figure 5: VHDL model of a register.

```

type MEM64x16 is array(0 to 63) of
  std_logic_vector(15 downto 0);
entity RAM64x16 is
  generic(MEM_INIT : MEM64x16);
  port(CLK          : in std_logic;
        Write       : in std_logic;
        ADDRESS     : in std_logic_vector(5 downto 0);
        DATA_IN    : in std_logic_vector(15 downto 0);
        Read        : in std_logic;
        DATA_OUT   : out std_logic_vector(15 downto 0);
        MEM_DUMP    : out MEM64x16);
end RAM64x16;

```

Figure 6: VHDL memory model.

```

architecture BEHAVE of RAM64x16 is
  signal MEMORY : MEM64x16 := MEM_INIT;
begin
  input : process
  begin
    wait until (CLK = '1' and not CLK'STABLE);
    if Write = '1' then
      MEMORY(conv_integer(ADDRESS)) <= DATA_IN;
    end if;
  end process;
  output : process(READ, ADDRESS)
  begin
    if READ = '1' then
      DATA_OUT <= MEMORY(conv_integer(ADDRESS));
    else
      DATA_OUT <= (others => 'Z');
    end if;
  end process;
  MEM_DUMP <= MEMORY;
end BEHAVE;

```

Figure 6 (cont): VHDL memory model.

```

MEM : RAM64x16 generic map ((
  X"100D", -- 00  MOVE #NUM2,R0
  X"184C", -- 01  MOVE NUM1,R1
  X"0880", -- 02  MOVE (R0),R2

  X"E000", -- 06  Halt

  X"000A", -- 0C  NUM1  10
  X"0007", -- 0D  NUM2   7

  others => X"0000")) -- xx-63
port map( CLK      => CLK,
  Write   => MEM_Write,
  ADDRESS => MAR_OUT,
  DATA_IN => MDR_OUT1,
  Read    => MEM_Read,
  DATA_OUT => MDR_IN1,
  MEM_DUMP => MEM_DUMP );

```

Figure 7: Memory instantiation and initialization.

Control Signals

The main VHDL is a structural model of the components to construct the architecture from Figure 2. Most of this VHDL is provided to the

students, so they can concentrate on the control signals. The instruction set is implemented by modeling the control flow chart, shown in Figure 8, as a VHDL case statement, shown in Figure 9.

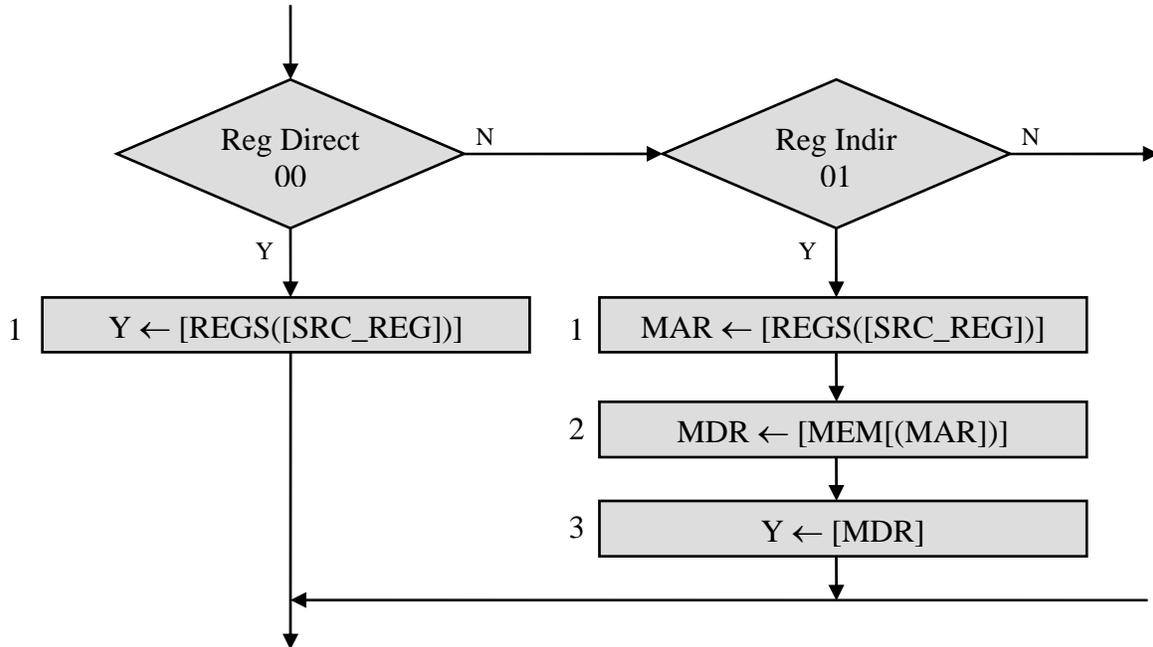


Figure 8: Control flow chart.

```

Case Cnt is
  when "001" =>
    case SRC_MODE is
      when "00" => --Reg Dir
        REGS_Read <= '1';
        Y_En_In1 <= '1';
      when "01" => --Reg Ind
        REGS_Read <= '1';
        MAR_En_In <= '1';

    when "010" =>
      case SRC_MODE is
        when "00" => -- Reg Dir "null"

        when "01" => -- Reg Ind
          MEM_Read <= '1';
  
```

Figure 9: VHDL control model (abbreviated).

VHDL Simulation

Verification of the VHDL models is accomplished via simulation using an appropriate testbench. The testbench instantiates a copy of the main VHDL model and provides visibility of the key signals such as

the control step count and instruction register. When execution is complete the final contents of the registers and memory are checked for the expected results. A portion of the testbench is shown in Figure 10 and an example simulation waveform is shown in Figure 11.

```

test : process
begin
  wait until (DONE = '1');
  for I in 0 to 3 loop
    REGS_DATA <= REGS_DUMP(I);
    wait for 20 ns;
    case I is
      when 0 =>
        assert (REGS_DATA = X"000D")
          report "Incorrect DATA at REGS ADDR 0,
                Failure: MOVE #NUM2,R0";
      when 1 =>
        assert (REGS_DATA = X"000A")
          report "Incorrect DATA at REGS ADDR 1,
                Failure: MOVE NUM1,R1";
    end case;
  end loop;
end test;

```

Figure 10: VHDL testbench.

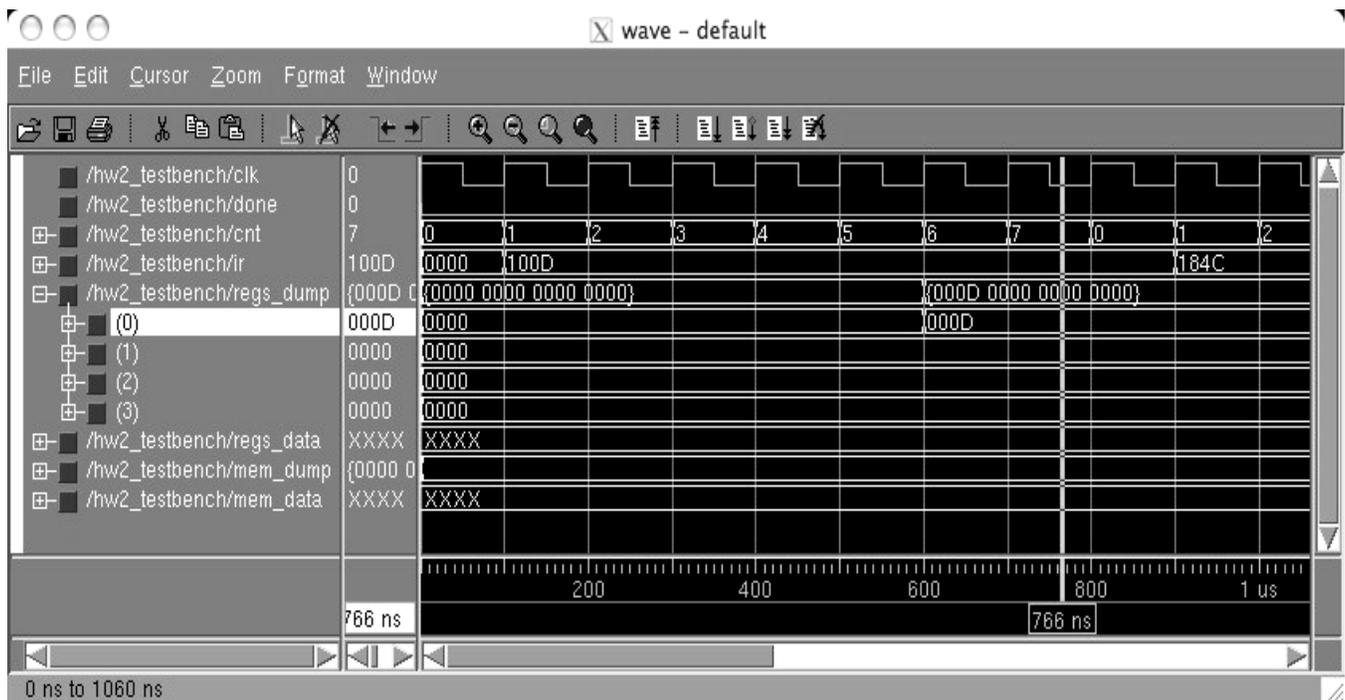


Figure 11: VHDL simulation waveform.

Microprogrammed Control

The next homework exercise builds on the previous one by adding the concept of microprogrammed control. The controller is again based on a text example, shown in Figure 12, which is readily adapted to the specific design, shown in Figure 13.

The single-bus architecture and instruction format remain the same, with the addition of the microinstruction format shown in Figure 14. The microinstruction format allows the students to write microcode to implement the control sequences. These microinstructions are loaded into the control store ROM of the microcontroller.

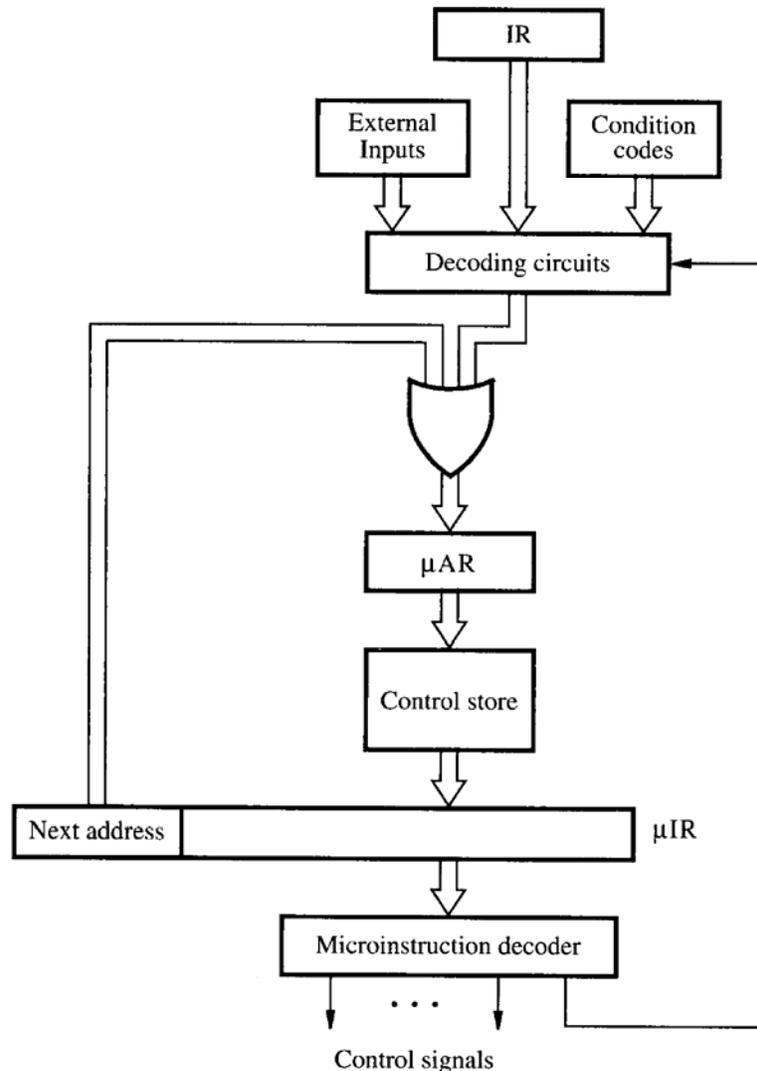


Figure 12: Microprogram controller from text.

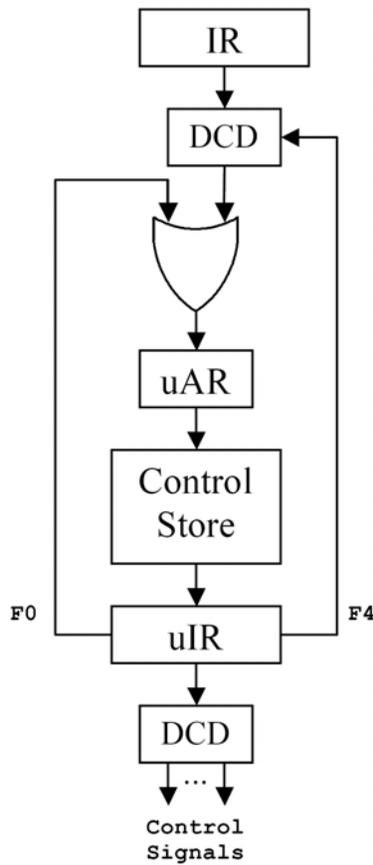


Figure 13: Microprogram controller for homework.

15 14 13 12 11 10		9 8 7		6 5 4		3 2 1 0	
F0				F1	F2	F3	F4

F1		F2		F3	
000	NOP	000	NOP	00	NOP
001	PC_En_Out	001	IR_En_In1	01	MAR_En_In
010	MDR_En_Out2	010	MDR_En_In2	10	Z_En_Out
011	Z_En_In	011	Y_En_In1	11	Y_En_In2
100	MEM_Write	100	IR_En_Out1		
101	MEM_Read	101	REGS_Sel		
110	REGS_Write	110	PC_Inc		
111	REGS_Read	111	Halt		

F0	F4	OR Fn
Addr of next instr	00 Next Addr	
	01 OR _{SRC}	00&SRC_MODE&00
	10 OR _{OP}	000&OPCODE
	11 OR _{DST}	000&DST_MODE&00

Figure 14: Microinstruction format.

VHDL Model

The existing VHDL models are supplemented with the addition of a ROM for the control store and decoders to produce the control signals. The control store is loaded with student generated microcode via the same generic construct used for the RAM. A sample of the ROM contents is shown in Figure 15.

Due to the implementation of the control logic in the control store, the VHDL controller process is simplified to the reading of the microinstruction from ROM and the determination of the address of the next microinstruction, as shown in Figure 16. The testbench and simulation results follow directly from the addressing mode homework, using the same assembly language program and test data.

```
Control_Store: ROM64x16 generic map ((
    X"04E4", -- 00  MAR <- [PC], PC <- [PC]+1
    X"0A80", -- 01  MDR <- [MEM([MAR])]
    X"0D10", -- 02  IR <- [MDR]
    X"4001", -- 03  Branch SRC_MODE

    X"83B2", -- 10  Y <- [REGS([SRC_REG])], Branch OPCODE

    others => X"0000"))
port map ( ADDRESS => uAR,
          Read      => ROM_Read,
          DATA_OUT => uIR );
```

Figure 15: Control store ROM.

```
uprog_cntl : process
begin
    wait until (CLK = '0' and not CLK'STABLE);
    ROM_READ <= '1';
    wait until (CLK = '1' and not CLK'STABLE);
    if Halt = '0' then
        case F4 is
            when "00" =>
                uAR <= F0 or "000000";
            when "01" =>
                uAR <= F0 or "00" & SRC_MODE & "00";
            when "10" =>
                uAR <= F0 or "000" & OPCODE;
            when "11" =>
                uAR <= F0 or "000" & DST_MODE & "00";
            when others => null;
        end case;
    end if;
end process;
```

Figure 16: VHDL microprogram controller.

Design Project

The capstone design project ties together many of the concepts from throughout the course to produce a model of a simple processor capable of fetching and executing a sample assembly language program. The processor is based on a multiple-bus architecture example from the text, shown in Figure 17. Only minor modifications are necessary to create the design project architecture, shown in Figure 18.

The design project computer uses a hardwired controller, again based on a text example shown in Figure 19. The VHDL model combines

existing components to create the controller design shown in Figure 20.

The instruction format and addressing modes remain the same as previous homeworks, with the addition of instructions for ADD and Branch. The expanded opcode table is shown in Figure 21. This expanded instruction set is used to create the assembly language program shown in Figure 22. The program uses multiple addressing modes and a loop to add a sequence of numbers. This basic program is used through the course as a recurring example to illustrate various concepts.

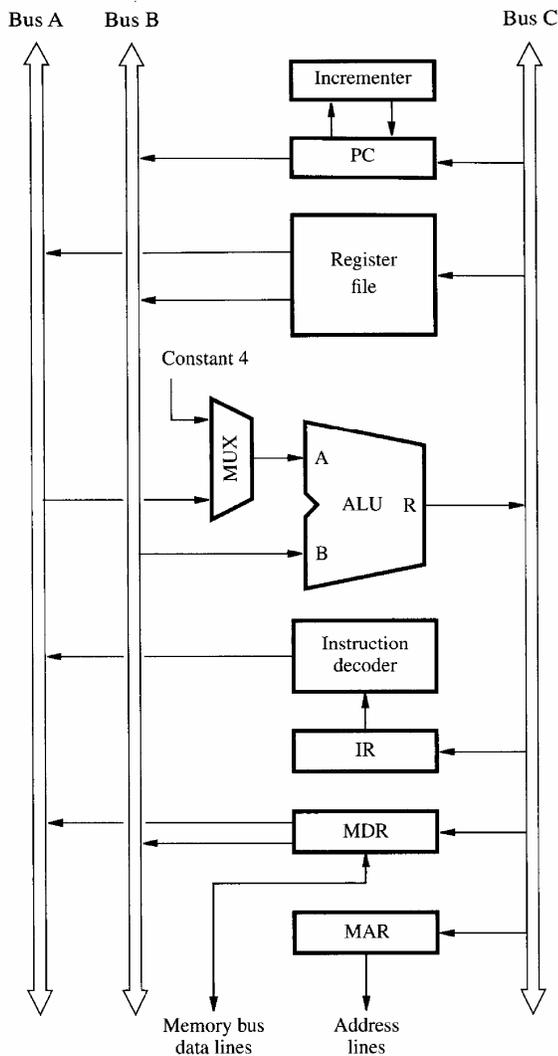


Figure 17: Multiple-bus architecture from text.

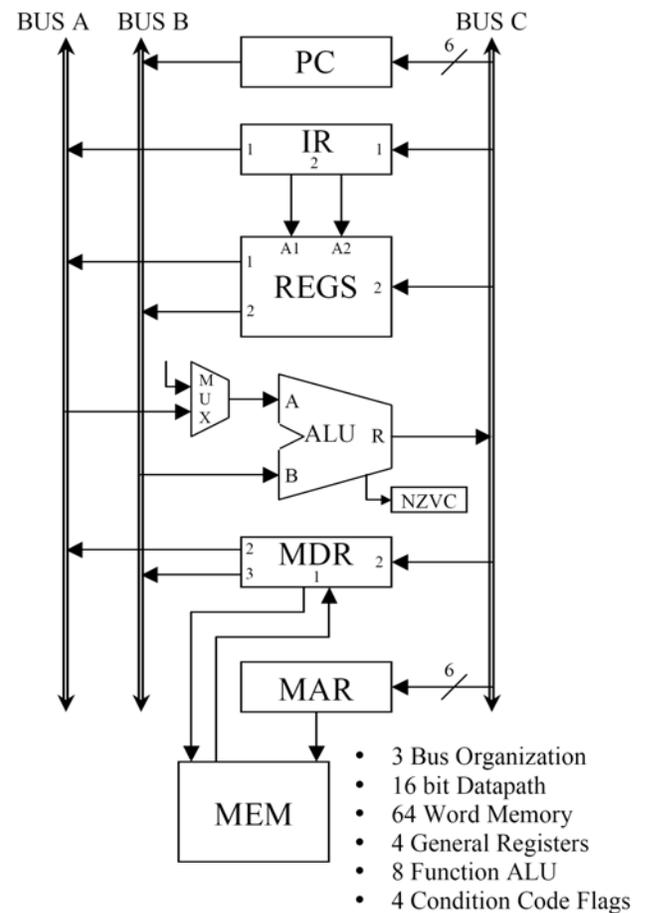


Figure 18: Design project architecture.

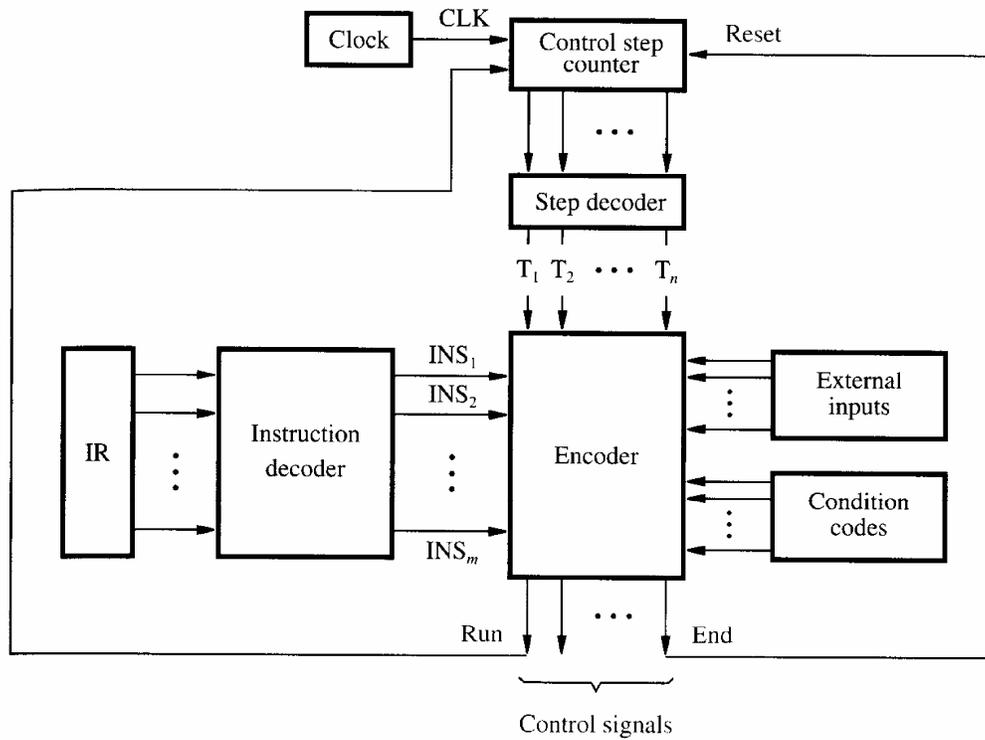


Figure 19: Hardwired controller from text.

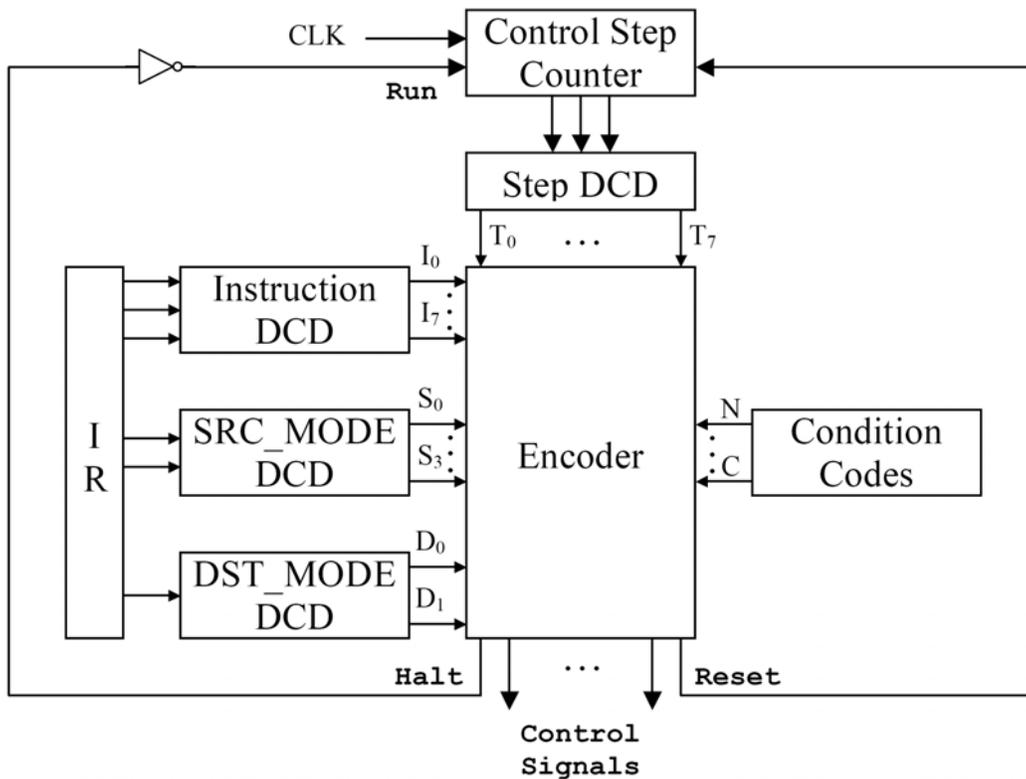


Figure 20: Controller for design project.

OP	Fn	Assembly Language	RTN
000	MOVE	MOVE SRC ,DST	DST <- [SRC]
001	ADD	ADD SRC ,DST	DST <- [SRC] + [DST]
110	BGTZ	BGTZ DISP	PC <- [PC] + DISP (CC > 0)
111	HALT	HALT	Halt <- 1

Figure 21: Expanded opcodes for design project.

```

        MOVE N,R1
        MOVE #NUM1,R2
        MOVE #0,R0
    LOOP ADD (R2),R0
        ADD #1,R2
        ADD #-1,R1
        BGTZ LOOP
        MOVE R0,SUM
        HALT

```

Figure 22: Assembly language program for design project.

VHDL Model

The VHDL for the design project is again a structural model of the interconnection of the components from Figure 18. The main exercise for the students is the determination of the control signal logic using the method detailed in the text and class notes. The resulting VHDL controller structure is shown in Figure 23.

VHDL testbench and simulation again follow directly from previous homeworks. Students can verify their controller design by examining simulation waveforms showing signal values and via register and memory dumps showing their final contents. A sample simulation waveform is shown in Figure 24.

```

Instruction_DCD : DCD3to8 port map ( );
Step_DCD : DCD3to8 port map ( );
Source_DCD : DCD2to4 port map ( );
Dest_DCD : DCD1to2 port map ( );

encoder : process(T)
begin
    PC_Inc <= T(0);
    PC_En_Out <= T(0)
                or (T(3) and I(6));

    REGS_Read1 <= (T(3) and I(0) and S(0) and D(0))
                  or (T(3) and I(0) and S(0) and D(1));

```

Figure 23: VHDL controller for design project.

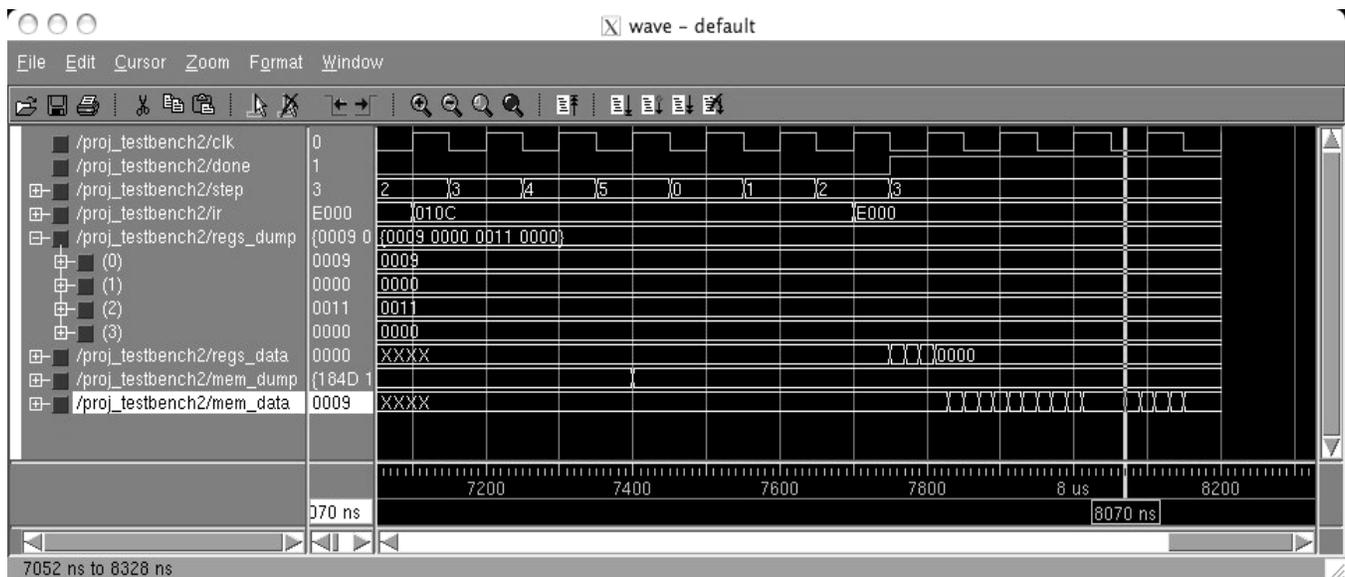


Figure 24: Design project simulation waveform.

Results and Conclusions

The VHDL homeworks and design project described in this paper have been successfully used to augment the classroom instruction for a pilot section of a computer architecture course. Student feedback has been very positive that the exercises provided direct reinforcement of classroom concepts and allowed them to visualize results via simulation. Approximately three-quarters of the students were able to successfully complete all homework assignments and over two-thirds were able to implement a completely functional design project.

The exercise design methodology was to combine/modify instructor provided VHDL models, rather than turning the course into a “programming” class. Student effort was focused on demonstration and understanding of classroom concepts by concentrating on the control signals and logic, rather than becoming engrossed in the VHDL. All architectural designs were derived directly from textbook examples providing reinforcement of concepts, as opposed to introduction of additional disparate material.

Again, the existing course text contains no coverage of VHDL. Hardware and architectural

constructs were described in sufficient detail to allow easy modeling in VHDL by focusing on the “hardware description” characteristics of the language. The resulting combination of textbook and classroom instruction with VHDL modeling and simulation exercises provided students with a more robust learning experience and exposure to state-of-the-art design tools.

Bibliography

1. C. Hamacher, Z. Vranesic, and S. Zaky, *Computer Organization*, 5th ed., McGraw Hill, New York, 2002.

Biographical Information

Ronald J. Hayne is an Assistant Professor in the Department of Electrical and Computer Engineering at The Citadel. He received his B.S. in Computer Science from the United States Military Academy in 1980, his M.S. in Electrical Engineering from the University of Arizona in 1987, and his Ph.D. in Electrical Engineering from the University of Virginia in 1999. Dr. Hayne's professional areas of interest include digital systems design and hardware description languages. He is a retired Army Colonel with experience in academics and Defense laboratories.