EVOLUTION OF THE INSTRUCTIONAL PROCESSOR

Ronald J. Hayne
Department of Electrical and Computer Engineering
The Citadel

John I. Moore, Jr.
Department of Mathematics and Computer Science
The Citadel

Abstract

Most modern commercial microprocessors are too complex to be used as introductory examples. Many digital design courses and texts use hardware description language models of these processors, but they are often ad hoc. What is needed is a basic processor with sufficient complexity that can be modified, programmed, and tested.

An instructional processor has been developed for use as a design example in an Advanced Digital Systems course at The Citadel. The initial architecture provides sufficient complexity to demonstrate fundamental programming concepts and the entire system is modeled and simulated in VHDL. A collaborative project between the Department of Electrical and Computer Engineering and the Department of Mathematics and Computer Science has added new capabilities to the design, including expanded memory, additional instructions, and more addressing modes. Machine code from a compiler or assembler can also be uploaded without modification of the VHDL model.

The instructional processor is now in its third iteration with additional capabilities, an updated controller design, and a new memory model. Feedback is very positive that the VHDL model and FPGA implementation of the processor illustrate fundamental design concepts without unnecessary complexity. The expanded project continues to achieve its goal as a valuable instructional tool for Advanced Digital Systems with future utilization as an implementation platform for a Compiler Design course.

Introduction

Teaching Advanced Digital Systems involves use of many design examples including counters, registers, arithmetic logic units, and memory. The design of a computer processor combines these components into an integrated digital system. Most modern commercial microprocessors are too complex to be used as introductory examples of processor design. Hardware description language models of these processors exist, but are often ad hoc and don't divide the architecture into teachable subsets [1,2]. Other microprocessor designs are part of larger or follow-on courses in computer architecture [3,4]. What is needed is a basic processor with sufficient complexity to illustrate major design elements that can be modified, programmed, and tested.

An instructional processor has been developed for use as a design example in ELEC 418 Advanced Digital Systems at The Citadel [5]. The initial architecture provides sufficient complexity to demonstrate fundamental programming concepts. The entire system is modeled in VHDL and can be simulated to demonstrate operation of the processor. Since the original design was kept very simple, there are many opportunities for improvement of both the instruction set architecture and the programming environment.

A collaborative project between the Department of Electrical and Computer Engineering and the Department of Mathematics and Computer Science has added new capabilities to the instructional processor. Initially targeted to provide improved higher-level language support

for CSCI 412 Compiler Design, the project has resulted in the evolution of the processor architecture into a more capable design, while still retaining its core function as an instructional tool.

Instruction Set Architecture

The instruction set architecture of the example processor has been designed to illustrate multiple operations and basic addressing modes. It is based on a three bus organization of a 16-bit data path with a four-word register file (REGS) [6]. Key registers include: program counter (PC), instruction register (IR), memory data register (MDR), and memory address register (MAR). New to this version of the design are a subroutine STACK and a higher capacity, 4K word by 16-bit, memory (MEM). The complete data path is shown in Figure 1.

Addition of new instructions and addressing modes required modification of the processor instruction format, including redefining and resizing several fields. The new format provides four new conditional branch instructions with a larger, 10-bit, relative offset. An additional destination addressing mode allows for pointers for accessing the expanded memory, but at the expense of a smaller absolute address range. The resulting data and branch instruction formats contain fields for the opcode (OP), operand source (SRC), operand destination (DST), and branch mode (MD), as shown in Figures 2 and 3.

The expanded instruction set provides sufficient complexity to demonstrate fundamental programming concepts such as data transfer, counting, indexing, and looping. Addition of a subroutine capability allows for modularization of programs and support for top-down design. Library functions can be created for more complex operations not defined by the basic opcodes.

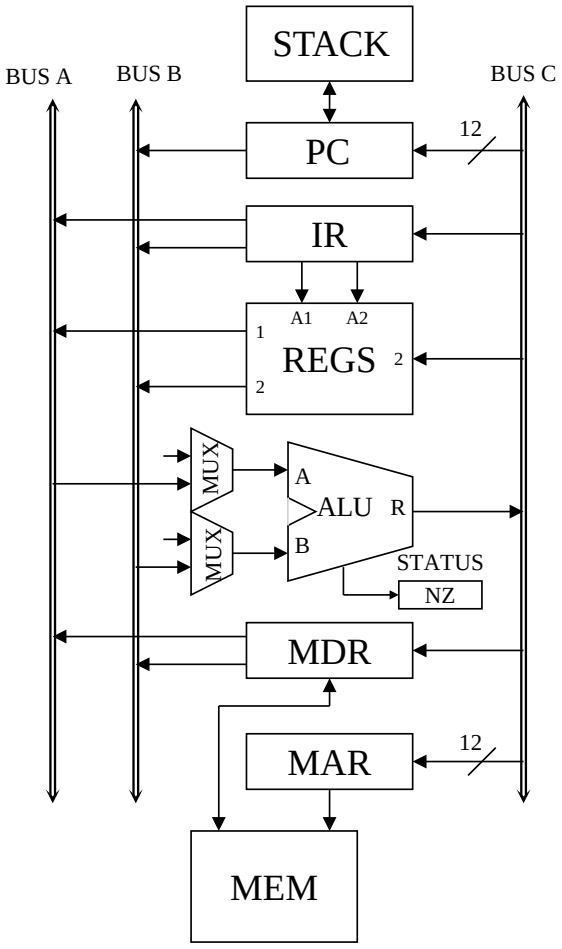
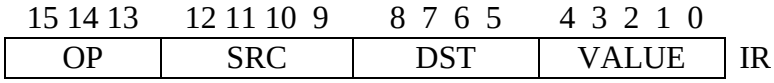


Figure 1: Data Path for Instructional Processor.

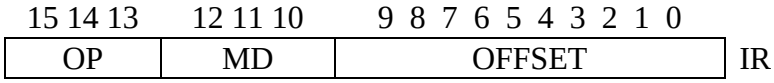
Key to the management of the increased memory capacity is a new memory model, shown in Figure 4, defining locations of memory-mapped input/output (I/O), data, and programs. Both the I/O and data segments can be accessed via the absolute addressing mode, while additional data locations can be created at the bottom of the memory space by using register indirect addressing for pointers. The location of the program segment determines the initial value of the PC for the beginning of the fetch sequence. This memory model provides the necessary structure while still maintaining enough flexibility for efficient resource utilization.



	Mode	REG #	Name	Syntax	Effective Address
SRC or DST	00	00-11	Register Direct	Rn	EA = Rn
	01	00-11	Register Indirect	[Rn]	EA = (Rn)
	10	vv	Absolute	[Value]	EA = Value
	11	vv	Immediate	Value	Operand = Value

OP	Fn	Assembly Language	Register Transfer Notation
000	MOVE	`MOVE SRC,DST`	DST ← SRC
001	ADD	`ADD SRC,DST`	DST ← SRC + DST
010	INV	`INV SRC,DST`	DST ← not SRC
011	AND	`AND SRC,DST`	DST ← SRC and DST
100	SHL	`SHL SRC,DST`	DST ← SHL(SRC)
101	ASHR	`ASHR SRC,DST`	DST ← ASHR(SRC)
110	…		

Figure 2: Data Instruction Format.



OP	MD	Fn	Assembly Language	Register Transfer Notation
111	000	BRA	`BRA Offset`	PC ← PC + Offset
	001	BZ	`BZ Offset`	PC ← PC + Offset (Z = 1)
	010	BNZ	`BNZ Offset`	PC ← PC + Offset (Z = 0)
	011	BN	`BN Offset`	PC ← PC + Offset (N = 1)
	100	BNN	`BNN Offset`	PC ← PC + Offset (N = 0)
	101	…		
	110	BSR	`BSR Offset`	STACK ← PC; PC ← PC + Offset
	111	RTN	`RTN`	PC ← STACK

Figure 3: Branch Instruction Format.

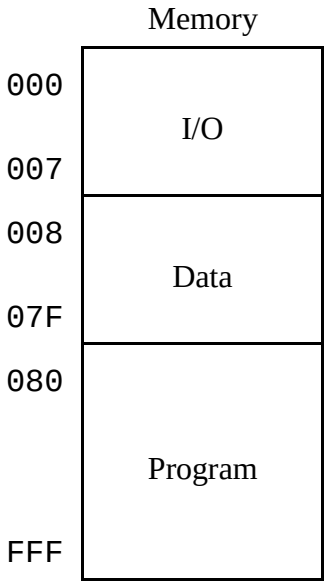


Figure 4: Processor Memory Model.

VHDL Model

Design of the instructional processor is taught in sections covering the instruction set architecture, followed by implementation of the data path, and finally the fetch, decode, and execute sequences for the control unit. Each component is modeled in VHDL and functionally verified using the Xilinx ISE [7].

The VHDL model is created in phases with new capabilities added in each phase. Phase 1, for example, includes the components of the data path, which have been developed throughout the course. These include the memory, registers, multiplexers, and ALU shown in Figure 1. Implementation of the controller follows the state machine design techniques used in many previous examples. A sample of the register transfer notation and control signals for the instruction fetch sequence is shown in Figure 5, and the resulting VHDL is shown in Figure 6.

Programming Environment

In previous versions of the instructional processor, programs were written in hexadecimal machine code and loaded into memory by modification of the VHDL component model. The collaborative project with the Department of Mathematics and Computer Science has also produced a programming environment that includes an assembler for generation of machine code from user programs. This binary machine code can now be loaded into memory, via file input, without modification of the VHDL model.

Step	Register Transfer Notation	Control Signals

T0	MAR ← PC, PC ← PC + 1	`BUS_B <= PC`
		`ALU_OP <= Pass_B`
		`Load_MAR <= '1'`
		`Inc_PC <= '1'`

T1	MDR ← MEM(MAR)	`MEM_Read <= '1'` `Load_MDR <= '1'`

T2	IR ← MDR	`BUS_B <= MDR`
		`ALU_OP <= Pass_B`
		`Load_IR <= '1'`

Figure 5: Instruction Fetch Sequence.

```
Control : process(STEP, IR, STATUS)
begin
  case STEP is -- Fetch
    when T0 =>
      BUS_B <= "0000" & PC;
      ALU_OP <= Pass_B;
      Load_MAR <= '1';
      Inc_PC <= '1';
    when T1 =>
      MEM_Read <= '1';
      Load_MDR <= '1';
    when T2 =>
      BUS_B <= MDR;
      ALU_OP <= Pass_B;
      Load_IR <= '1';
```

Figure 6. VHDL Model for Control Unit.

The assembler serves as an intermediate step for a high-level language compiler to be taught in a future Compiler Design course. It supports all the syntax defined in the instruction formats in Figures 2 and 3, as well as basic assembler directives for program/data labels and macros. A demonstration program, which illustrates memory-mapped I/O, conditional branching, looping, and a subroutine, is shown in Figure 7.

Test programs can be simulated using Xilinx ISim [7] to verify their operation. A sample simulation trace is shown in Figure 8. The VHDL model, including the test program, can also be synthesized for a target Xilinx Spartan 3e FPGA and loaded onto a Digilent BASYS 2 board [8]. The SWITCH port is mapped to eight input switches on the board, while the LED port is mapped to eight LEDs. The final hardware implementation, shown in Figure 9, utilizes approximately 50% of the FPGA resources, while fully exploiting the available 4K of block RAM.

Student homework assignments involve expansion of the instructional processor by adding new addressing mode and opcode combinations. For example the assembly language instruction `MOVE [Rs],Rd` uses register indirect addressing for the source and register direct addressing for the destination.

```
.define SWITCH [0]
.define LED    [1]
.macro  SUB    arg1 arg2
        INV    arg1, arg1
        ADD    1, arg1
        ADD    arg1, arg2
.endm

.code
START:  MOVE   0x67, R1
REPEAT: MOVE   SWITCH, R0
        MOVE   R0, R3
        AND    0x01, R0
        BNZ    ON
        INV    R1, R1
        BRA    OUTLED
ON:     ASHR   R1, R1
OUTLED: MOVE   R1, LED
        BSR    DELAY
        BRA    REPEAT
DELAY:  MOVE   0, R2
LOOP:   ADD    1, R2
        BNZ    LOOP
        MOVE   1, R0
        SUB    R0, R3
        BNZ    DELAY
        RTN
```

Figure 7. Assembly Language Program.

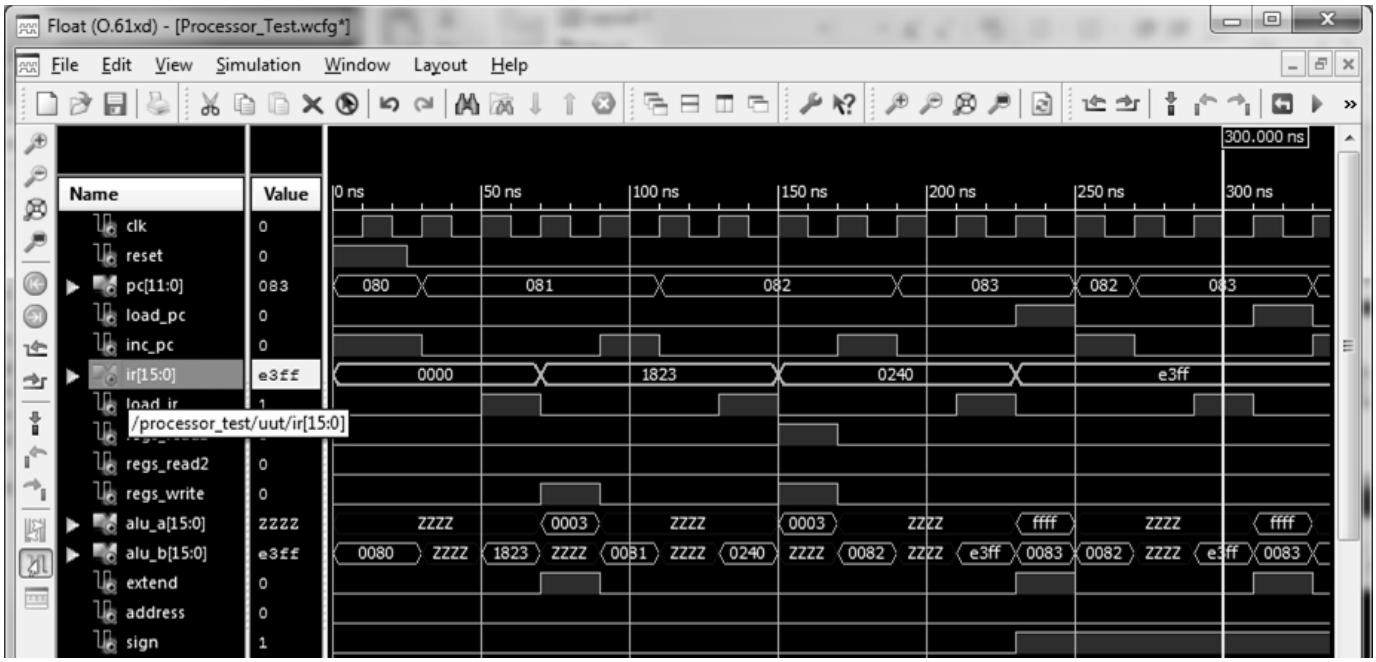


Figure 8: VHDL Simulation Waveform.

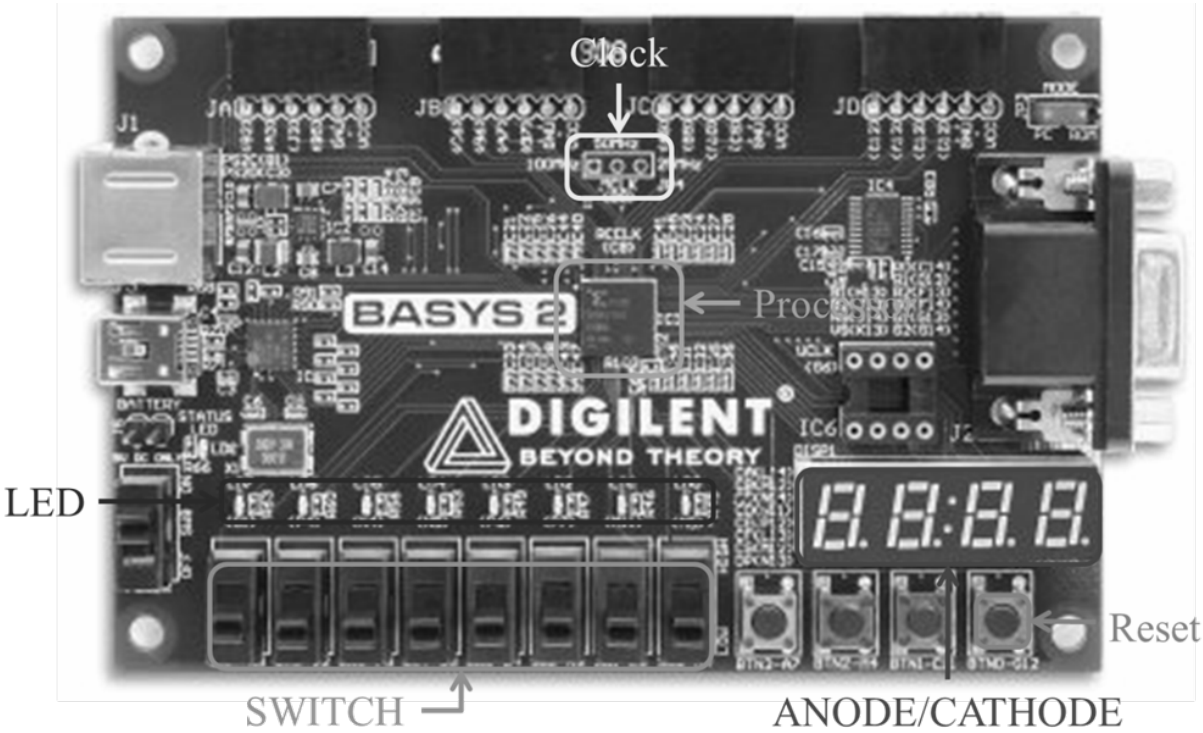


Figure 9. FPGA Implementation.

Students first determine the register transfer notation sequence necessary to execute the instruction, then they translate that into the appropriate VHDL control signals for the data path. A test program can then be written, assembled, loaded, and simulated to verify the correct function of the new instruction.

Results and Conclusions

The instructional processor is now in its third iteration with additional capabilities, an updated controller design, and a new memory model. The processor is used as a design example to replace the existing MIPS microprocessor in the current course text [1]. Because the new design uses examples throughout the semester, it integrates directly into the flow of the course. Development of the model in phases allows separate coverage of the data path and the sequential controller.

Students already have familiarity with assembly language programming from the prerequisite ELEC 330 Digital Systems Engineering course, allowing straightforward integration of the new programming environment. Realization of only a subset of the processor instructions provides sufficient capabilities to demonstrate fundamental programming concepts. Additional instructions, implemented as student homework assignments, allow direct application of the design techniques taught in class.

Student feedback is very positive that the VHDL model and FPGA implementation of the processor illustrate fundamental design concepts without unnecessary complexity. Responses from the end of course Student Evaluation of Learning indicate that 86% of the students either agreed or strongly agreed that "My ability to design a system to meet a specified requirement improved as a result of this course."

Results from homework assignments demonstrate that students can successfully design modifications to the processor and test them via program simulation. The average score for the processor homework and corresponding final exam question was 81%. When asked "What did you like most about this course?" multiple responses specifically cited the processor design and VHDL projects.

The collaborative project between the Department of Electrical and Computer Engineering and the Department of Mathematics and Computer Science at The Citadel has resulted in the evolution of the instructional processor into a more capable design with an improved programming environment. The expanded project continues to achieve its goal as a valuable instructional tool for Advanced Digital Systems with future utilization as an implementation platform for a Compiler Design course.

Bibliography

1. C. H. Roth, and L. K. John, *Digital Systems Design Using VHDL,* 2nd ed., Thompson, Toronto, Canada, 2008.

2. S. Lee, *Advanced Digital Logic Design: Using VHDL, State Machines, and Synthesis for FPGAs,* Thompson, Toronto, Canada, 2007.

3. D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture,* Morgan Kaufmann, San Francisco, CA, 2007.

4. E. O. Hwang, *Digital Logic and Microprocessor Design with VHDL,* Thompson, Toronto, Canada, 2006.

5. R. J. Hayne, "An Instructional Processor Design using VHDL and an FPGA," *Computers in Education Journal, ASEE,* Vol. 3 No. 2, April - June 2012.

6. R. J. Hayne, "VHDL Projects to Reinforce Computer Architecture Classroom Instruction," *Computers in Education Journal, ASEE,* Vol. XVIII No. 2, April - June 2008.

7. Xilinx ISE 13.1i Software Manuals, Xilinx, Inc., 2011.

8. Digilent Basys2 Board Reference Manual, Digilent Inc., 2010.

Biographical Information

Ronald J. Hayne is an Associate 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, his M.S. in Electrical Engineering from the University of Arizona, and his Ph.D. in Electrical Engineering from the University of Virginia. 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.

John I. Moore, Jr. is a Professor in the Department of Mathematics and Computer Science at The Citadel. He received his B.S. in Mathematics from The Citadel, his M.S. in Computer Science from Georgia Institute of Technology, and his Ph.D. in Mathematics from the University of South Carolina. Dr. Moore has a wide range of experience in both industry and academia, with specific expertise in the areas of object-oriented technology, mobile applications, programming language translators, graph theory, and e-commerce.