



COMPUTER ENGINEERING PROGRAM

California Polytechnic State University

©Copyright: 2007 by Bryan Mealy



CPE 269 Experiment 8



USING A PICOBLAZE3 MICROCONTROLLER FOR BIT MANIPULATIONS

Objectives:

- To learn the steps required to construct a PicoBlaze microcontroller in the Xilinx ISE environment
- To write an assembly language program to implement various bit-level manipulations
- To use the PicoBlaze3 assembler to assemble your source code and test the results in hardware

Somewhat Meaningful Comments: The PicoBlaze3 microcontroller is defined by a collection of VHDL files. These files roughly consist of a set of core files that define the actual PicoBlaze microcontroller and a set of VHDL wrapper files that interface the PicoBlaze3 microcontroller with the Nexys development board. The wrapper files serve to custom-fit the PicoBlaze3 microcontroller to the Nexys board and in the process, hide some of the less important details from the PicoBlaze3 assembly language programmer.

The main goal of this experiment is to use the Xilinx tools to create and download the PicoBlaze microcontroller to the Nexys development board and have it execute a PicoBlaze assembly language program. When you prepare an assembly language program to be included as part of the PicoBlaze microcontroller description, you'll gain experience with SimPicoBlaze, the PicoBlaze assembler. This development process will be used in the every CPE 269 experiment that uses the PicoBlaze microcontroller on the Nexys board. The steps in this process are detailed in the procedure portion of this experiment and are described later.

PicoBlaze Programming Overview

With any luck, your CPE 229 instructor has already talked about PicoBlaze and presented some of the basics of programming it. Just in case they have not, this section provides the really fast overview of the basics involved in writing a PicoBlaze program.

Anytime you feel that special need to program a microcontroller (such as PicoBlaze), there is always two things you need to start with:

- 1) **Programming Model:** The programming model is the programmer's view of the machine: it shows what hardware resources are available for the programmer to use. These resources include registers and other types of memory. Another useful definition for the Programming Model is that it is the set of registers that can be manipulated by using instructions in the associated instruction set. The programming model for PicoBlaze is shown in
- 2) **Instruction Set:** The instruction set lists the set of operations that the hardware can perform under control of the programmer. The instruction set generally refers to the set of assembly language instructions that can be used on a given machine. The PicoBlaze instruction set can be found in the PicoBlaze documentation and a condensed form can be found on the course website.

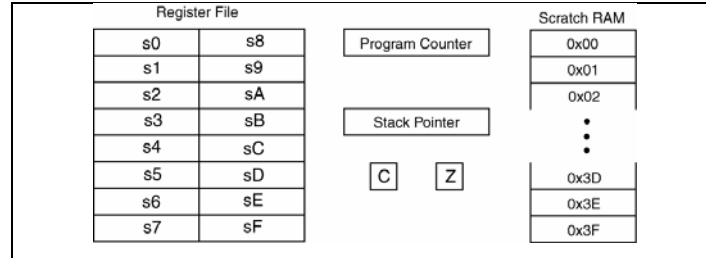


Figure 1: The PicoBlaze Programming Model.

A PicoBlaze assembly program consists of a set of assembly language instructions that direct the processing of data through the registers (s0 → sF) in the PicoBlaze hardware. The programmer decides what instructions are necessary to perform some task. The resulting assembly language program is then provided as input to a program known as an *assembler* which translates the assembly language instructions into 1's and 0's (machine code) that can be understood by the PicoBlaze hardware.

A complete PicoBlaze assembly language program appears in Figure 2. This program does not do anything but it nevertheless is a complete program. The OR instruction essentially OR's the value in register s0 with the immediate value 00 (interpreted as a hex value) and stores the result back in register s0. The JUMP instruction transfers program control from the JUMP instruction back to the instruction with the “main:” label.

```

;-----
main:    OR      s0,00      ; do nothing (nop)
         JUMP   main      ; go back to main loop
;-----

```

Figure 2: A basic PicoBlaze assembly language program.

The thing that makes computers useful is their ability to communicate with the outside world. And since PicoBlaze is arguably a useful computer, there must be some instructions that handle details such as I/O (input/output). Alas, the INPUT and OUTPUT instructions do just that. Note the use of these instructions in the program shown in Figure 3. This program contains two assembler directives: SWITCHES and LEDS. The assembler directives are not instructions; they are actually messages to the assembler to replace the alphabetic terms with the associated numerical terms in any assembly instruction in which the alphanumeric term appears. The INPUT command reads data from the outside world (from the switches) and places the data into register s2. The data is then complimented (XORed with FF with the result being placed in register s2). The resulting data in register s2 is the written to the LED port by using the OUTPUT instruction. The program then once again JUMPs back to the instruction that contains the “main:” label. The program essentially operates in an endless loop.

```

;-----
;- Assembler Directives
;-----
CONSTANT SWITCHES, 30      ; switches address = port 30 (input)
CONSTANT LEDS, 0C         ; switches address = port 0C (output)
;
;-----
main:    INPUT   s2, SWITCHES ; put switch status into register s2
         XOR    s2, FF      ; complement switch data (trick)
         OUTPUT  s2, LEDS   ; write data to LEDs
         JUMP   main      ; main loop (do it again)
;-----

```

Figure 3: A slightly more complex PicoBlaze assembly language program.

Programming Assignment:

Write a PicoBlaze assembly language program that will:

- Read the status of the eight slide switches on the Nexys board.
- Read the status of BTN2 on the Nexys board.
- Depending on the status of BTN2 (refer to the Nexys reference manual for button logic levels), the following actions should occur on the Nexys board:
 - If BTN2 is not being pressed, display the number of switches that are in the on position by turning on the left-most LEDs and working towards the right. In other words, if n switches are on, the n left-most LEDs will be on (see examples below).

Switch Data	LEDs (dark = on)
1001 0110	● ● ● ● ○ ○ ○ ○
0010 0001	● ● ○ ○ ○ ○ ○ ○

- If BTN2 is being pressed, swap the order of the bits (again, consider the eight slide switches to represent the input bits). For example, the status of SW1 will be displayed on LD8; the status of SW2 will be displayed on LD7 etc.
 - Output the bits that were swapped in the previous step to the eight LEDs on the DIO1 board.
 - Repeat the previous steps continuously.

NOTE: Instructions detailing how to access the various input and output devices on the Nexys board are located in the header of the provided `prog_rom.vhd` file.

Procedures1) PicoBlaze Environment Set-up:

- a. Grab the zipped file listed with Experiment 7 on the CPE-269 website. Extract the files to a folder that is not in the Xilinx path and does not have spaces in the pathname. This set of files contains everything you need in order to write and execute PicoBlaze3 programs on the Nexys board.

2) Writing the assembly code:

- a. In extracted zip file, there is a file named "`prog_rom.psm`". This is the file where you'll write your PicoBlaze3 assembly language source code. Open this file from the Xilinx environment and do your editing there (the Xilinx editor ain't too bad!). Save the file when you're done editing.

3) Assembling the assembly language source code:

- a. From the start menu in Windows, select the run option.
- b. Navigate to the directory where you `prog_rom.psm` file lives.
- c. Assemble your source code by entering "`kcpsm3 prog_rom.psm`" at the command prompt. Fix any errors that you may have before continuing. If your code successfully assembles, then the assembler has generated a VHDL ROM object containing the machine code for your assembly language program. This file replaces the `prog_rom.vhd` file that was already in your project from the previous time you assembled your code. Keep in mind that you know you did something correct when your green check change over to orange question marks after you assemble.

4) Testing the assembly code in hardware:

- a. Download your design onto the Nexys board like you've been doing all quarter.

Deliverables:

1. Demonstrate your working circuit to the lab instructor or Teaching Assistant.
2. Provide a listing of your assembly source code with your lab write up. Make sure you take a look at the PicoBlaze assembly language style file before you submit your source code.
3. Answers to the questions below.

Questions:

1. The structure of your program could be characterized as an endless loop (code that repeated itself over and over again). The system clock on the Nexys board runs at 50MHz. Using this value and your knowledge of the PicoBlaze architecture, calculate the time (in seconds) required to complete each of the two iterations of the code contained in the endless loop. State your assumptions and show your work for this calculation.
2. The endless loop mentioned in the previous problem is typical of embedded system applications. Using only the instructions available in the PicoBlaze instruction set, would it be possible to stop PicoBlaze from executing instructions at some point in the code? Explain your answer.
3. What was the main function provided by the VHDL "wrapper" code for the PicoBlaze implementation?
4. There are actually two forms of the PicoBlaze INPUT and OUTPUT instructions. Briefly discuss the differences and describe a situation where form that uses parenthesis would be useful.
5. I suddenly had this good idea... why not provided each PicoBlaze register with its very own carry and zero flag. Briefly discuss why this would not be good idea.