Session ____

The WIMP51: A Simple Processor and Visualization Tool to Introduce Undergraduates to Computer Organization

David Sullins, Hardy Pottinger, Daryl Beetner
University of Missouri – Rolla

I. Introduction

The University of Missouri – Rolla offers a Junior-level lecture and laboratory course in hardware/software co-design[1]. The course focuses on the 8051 family of microcontrollers. Many students who take this course have no prior experience with processor architecture, so a short introduction to some basic concepts of computer organization is given in the first few weeks of the course.

In the past, the computer architecture portion of the course was taught using the Gnome processor, described in Van den Bout's *Practical Xilinx Designers Lab Book*[2]. The Gnome is a four-bit processor with eight-bit instructions described in VHDL and targeted for the Xilinx 4k series FPGA. While the Gnome is at an appropriate level of complexity for the course, it is quite different from the 8051 microcontroller. Past course evaluations indicate that many students feel their time was wasted learning the Gnome instruction set, only to be told to forget the Gnome instructions and learn a new instruction set three weeks into the course. Still, the Gnome is useful in the classroom, as the 8051 microcontroller is much too complex for a short introduction to computer architecture.

To resolve these problems a replacement processor based on the 8051 was designed. Called the WIMP51, it is a simple binary-compatible subset of the 8051, lacking internal memory, interrupts, peripherals, and many of the 8051 instructions. The WIMP51 was implemented in synthesizable VHDL and an interactive graphical simulator was developed for use in lab.

II. WIMP51 Design

The WIMP51 is a classic von Neumann processor. Every instruction lasts three clock cycles, one clock cycle for each phase: Fetch, Decode, and Execute. All WIMP51 instructions are assembly language and machine language compatible with the Intel 8051. Table 1 shows the WIMP51 instruction set.

Table 1: WIMP51 instruction set

Instruction	Machine Code	Function
MOV A, #D	`01110100 dddddddd`	A <= D
MOV A, Rn	`11101nnn`	A <= Rn
MOV Rn, A	`11111nnn`	Rn <= A
ADDC A, #D	`00110100 dddddddd`	C,A <= A + D + C
ADDC A, Rn	`00111nnn`	C,A <= A + Rn + C
ANL A, Rn	`01011nnn`	A <= A and Rn
ORL A, Rn	`01001nnn`	A <= A or Rn
XRL A, Rn	`01101nnn`	A <= A xor Rn
SWAP A	`11000100`	A <= A(3:0) & A(7:4)
CLR C	`11000011`	C <= 0
SETB C	`11010011`	C <= 1
SJMP rel	`10000000 rrrrrrrr`	PC <= PC + rel + 2
JZ rel	`01100000 rrrrrrrr`	PC <= PC + rel + 2 if A = 0

As shown in Figure 1, the WIMP51 contains four eight bit registers: an instruction register (IR), an accumulator (ACC), a program counter (PC), and an auxiliary register (AUX). In addition, the WIMP51 contains an eight by eight-bit register file, an arithmetic logic unit (ALU), a program counter ALU (PC ALU), and a control unit. The ALU contains a one-bit carry register used to store the carry from previous ADDC operations. The ALU also generates a control signal called Z, which is 1 if the contents of the accumulator are zero, and 0 otherwise. These elements were combined to form the WIMP51 datapath.

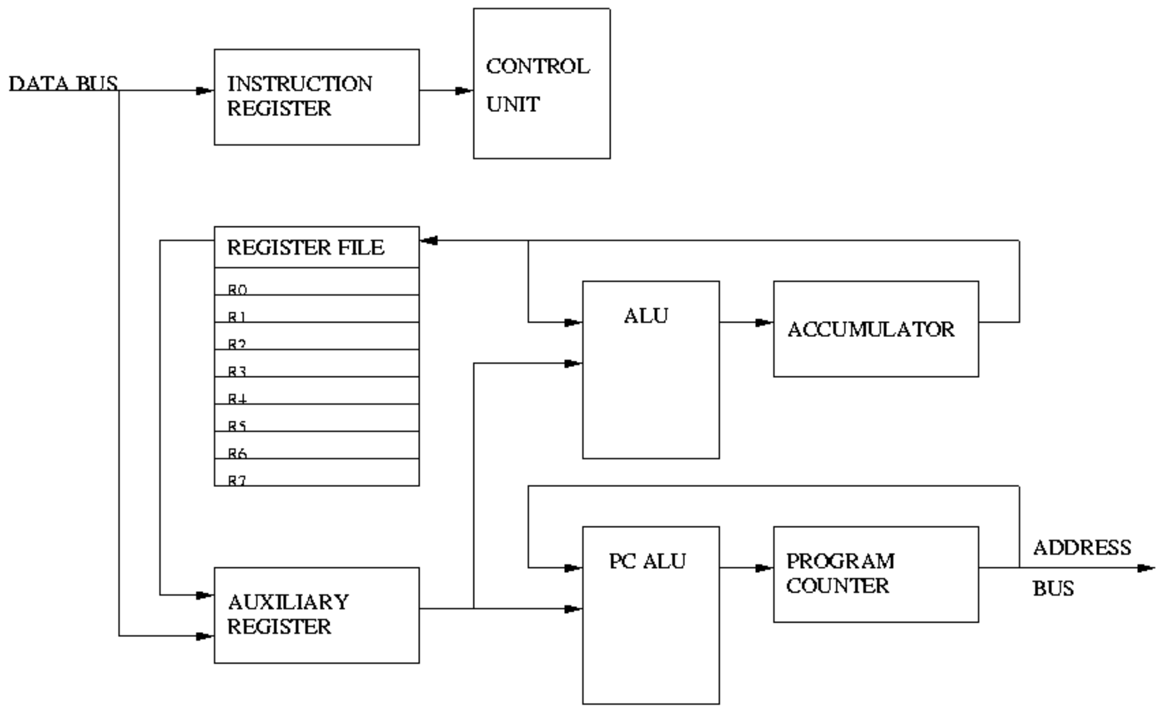


Figure 1: WIMP51 Datapath

In the first clock cycle, the Fetch cycle, the opcode for the next instruction is fetched from program memory into the instruction register through the data bus and the program counter is incremented by one. This cycle is the same for all instructions.

In the Decode cycle, the contents of the instruction register are examined. If the instruction requires an immediate operand, that operand is fetched from memory into the auxiliary register. If the instruction requires a register source operand, the contents of the appropriate register are fetched from the register file into the auxiliary register. The Decode cycle could also be called the operand fetch cycle.

In the third clock cycle, the Execute cycle, several things could happen. For branch instructions, the contents of the auxiliary register are added to the program counter, and the result is stored into the program counter. For the register destination instruction (MOV Rn, A) the contents of the accumulator are written to the appropriate register. For all other instructions, the ALU performs the necessary operation and the result is written into the accumulator.

The control unit is responsible for generating all the control signals necessary to make the processor work. It takes input from the instruction register and the Z flag, and outputs control signals to every component. See Figure 2 for a block diagram of the control flow.

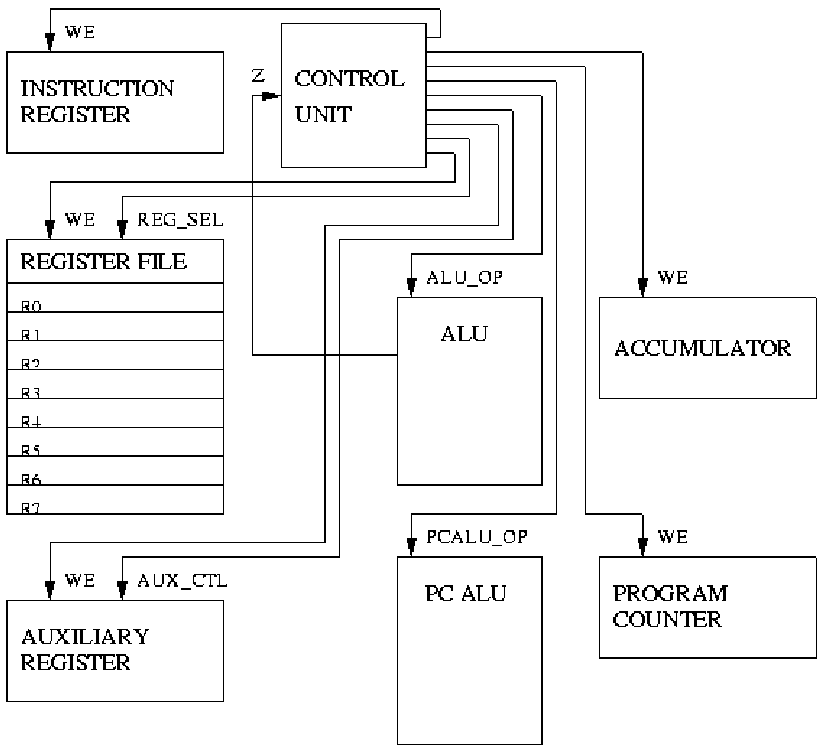


Figure 2: WIMP51 Control Path

Each register has a WE (write enable) input. The auxiliary register has an additional control input that selects whether data that is written comes from the data bus or the register file. The register file has a three-bit register select input that determines which register is read or written. The ALU and PC ALU each have a control input selecting which operation to perform. Finally, the PSEN_N output is the active low program memory read strobe, used to fetch data from program memory.

III. Visual Wimp

To add to the educational value of the Wimp51, an interactive Wimp51 simulator called the Visual Wimp was developed. The Visual Wimp shows an animated block diagram of the processor and allows the user to step through a program one clock cycle at a time. A screenshot of the Visual Wimp is shown in Figure 3.

The Visual Wimp was implemented in Tcl/Tk, a cross-platform interpreted language designed to be embedded into other applications[3]. Modelsim, the VHDL simulator used at UMR, includes an embedded Tcl/Tk interpreter. By using Modelsim to execute the Visual Wimp code, the graphical simulator is tied directly to the VHDL code, ensuring that the behavior of the simulator matches the behavior of the hardware. Modelsim is a cross-platform HDL simulator, so another benefit of implementing the Visual Wimp in this fashion is that it will run on any platform supported by Modelsim, including Microsoft Windows, Solaris, HP-UX, AIX, and Linux.

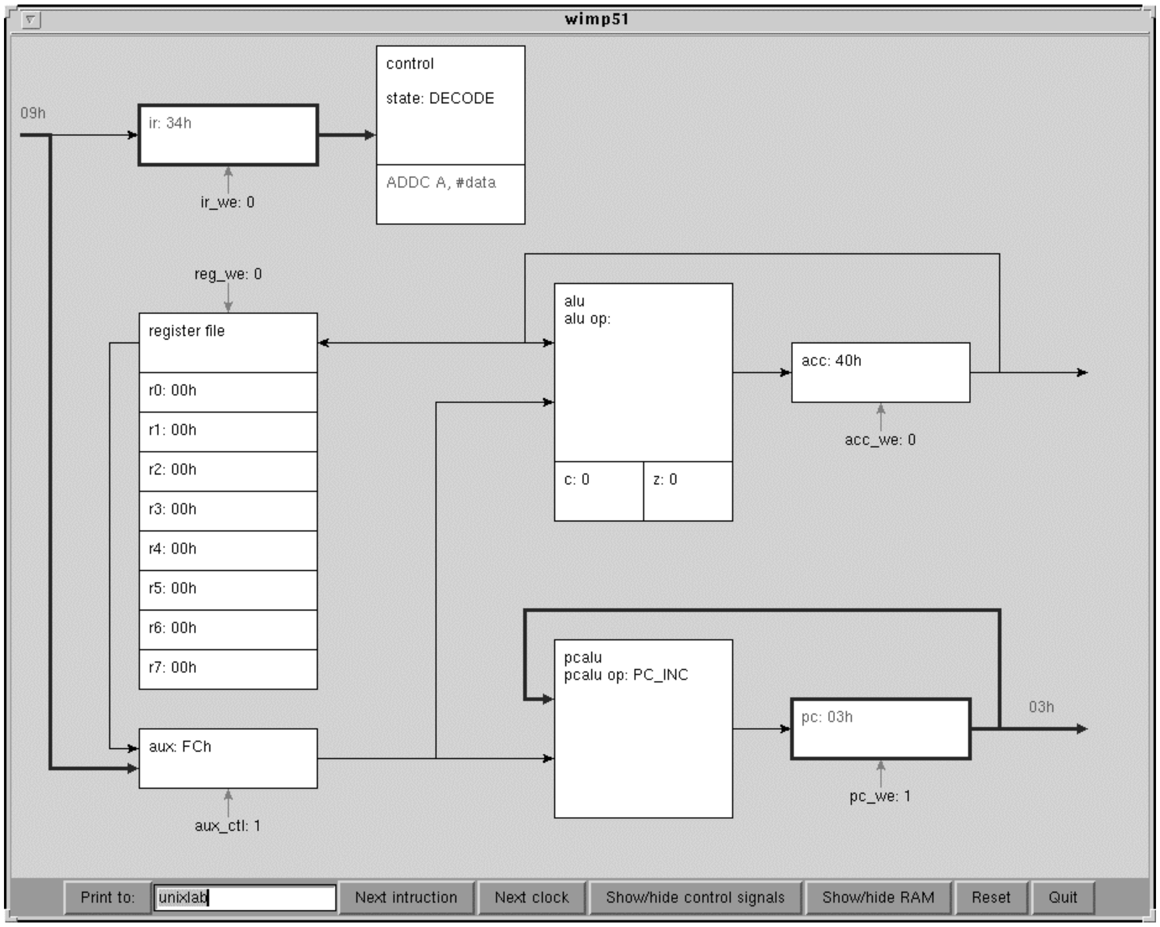
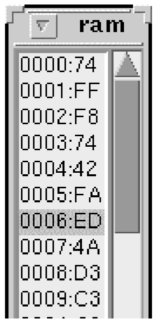


Figure 3: Visual Wimp window

Users can step forward one clock cycle with the "Next clock" button, or one machine cycle with the "Next instruction" button. The "Show/hide control signals" button will add or remove arrows on the display indicating the values of control inputs to each register. In the screenshot above, the control signal feature has been turned on. The "Show/Hide RAM" button will open or close a memory window, shown in Figure 4, that displays the contents of program memory.



```
0000:74
0001:FF
0002:F8
0003:74
0004:42
0005:FA
0006:ED
0007:4A
0008:D3
0009:C3
```

Figure 4: Memory window

During the Fetch cycle, the Visual Wimp highlights the address and data bus and shows what data is present on each bus. The current address in code memory is also highlighted in the memory window, if that window is active.

During the Decode cycle, the new value of the instruction register is highlighted in red. The arrow from the instruction register to the control unit is highlighted to indicate the dataflow. The decoded instruction is also shown inside the control unit. If the decoded instruction is a register-source instruction or an immediate-source instruction, the appropriate dataflow will also be highlighted in this cycle. Figure 3 was captured during a Decode cycle.

During the Execute cycle, the appropriate dataflow is highlighted. For a branch instruction, the inputs to the PC ALU will be highlighted. For the "MOV Rn, A" instruction, the path from the accumulator to the register file will be highlighted. For other instructions, the inputs to the ALU will be highlighted.

Control inputs to the ALU and the PC ALU are indicated in symbolic form inside the respective logic units. The register select control input to the register file is implicitly indicated by highlighting the appropriate register in red. Other control signals are optionally shown by toggling the "Show/hide control signals" button.

IV. Laboratory Use

The WIMP51 has been used in class for a semester at the University of Missouri – Rolla. Two laboratory experiments have been designed around the WIMP51, modified from previous experiments involving the Gnome processor.

The first laboratory experiment requires the students to write a simple program for the WIMP51. After assembling the program into a file in Intel-HEX format, the students simulate their program running on the VHDL model of the WIMP51 using Quicksim Pro. The students use the Visual Wimp interface to step through their program one clock cycle at a time. The students must verify their program works correctly, then answer questions about the operation of the hardware at various stages of their program.

The second laboratory experiment uses a hardware implementation of WIMP51. The students test the program from the previous experiment with an FPGA implementation of the WIMP51. The hardware implementation of the WIMP51 given to the students has a bug, however. The logic to produce the Z signal (indicating when the accumulator is 0) has been reversed. The students must discover why their program does not work by examining the signals from the accumulator, data bus, and address bus throughout the execution of their program. When they discover the problem, they must write software to work around the bug and describe how they would fix the problem in hardware if they had access to the faulty hardware design.

Together, these two laboratory experiments stress the importance of software simulation before hardware testing and the importance of co-simulation before implementation. After performing the first experiment, students are sure that their program is correct. They know that the problem must be due to a hardware design error, not a bug in their software. Additionally, the simulation helps the students get familiar with the operation of the hardware, making it easier to know what to look for once they discover there is a problem.

A third WIMP51-based experiment is under development. In this experiment, students will extend the instruction set of the WIMP51 with an additional instruction by adding hardware to the WIMP51 processor and modifying the control unit, simulate their altered hardware to verify that it works, and then test their design in an FPGA. This laboratory should give the students a deeper understanding of computer organization.

V. Results

Currently the WIMP51 has been in use for one semester at the University of Missouri – Rolla, both in a lecture course and in the associated laboratory course. Instructors for the lecture course indicate that the introduction of the 8051 has gone more smoothly than in previous semesters, due to the students' experience with the WIMP51.

Instructors for the laboratory course have seen that students have a better understanding of the operation of the hardware with the WIMP51 than with the Gnome processor. As there was no tool like the Visual Wimp for the Gnome processor, it is easier for the students to grasp what happens inside the Wimp processor during simulation. In a similar Gnome-based hardware debugging laboratory experiment from earlier semesters, students took longer to identify the problem and its cause. This is likely due to less understanding of the internal operation of the Gnome processor. Although the architecture of the Gnome is no more complicated than the architecture of the WIMP51, students needed more instructor intervention to know what to look for to uncover the source of the problem. This would imply that the simulator helped better prepare students for hardware debugging by helping them more completely understand how the processor works.

VI. Conclusions

The WIMP51 has proven to be a successful teaching tool for introducing computer architecture. It has the benefit of being machine language and assembly language compatible with the 8051 microcontroller, allowing WIMP51 code to be assembled by any standard 8051 assembler.

The addition of the Visual Wimp to the laboratory simulation environment reinforces concepts learned in lecture, shows students the dataflow through the processor, and helps them debug their programs. Student performance in lab shows that students understand the operation of the processorbetter after using the Visual Wimp.

The WIMP51 is an ongoing project. In the future, the Visual Wimp may also be used in the lecture course to demonstrate the operation of the processor when the concepts are first introduced. Future lab experiments will ask students to change the hardware of the WIMP51 to extend the instruction set. The WIMP51 and the Visual Wimp have already been shown to be effective teaching tools and will continue to improve instruction as new course material is developed around them.

References

1. D.G. Beetner, H.J. Pottinger, and K. Mitchel, "Laboratories Teaching Concepts in Microcontrollers and Hardware-Software Co-Design," *30th ASEE/IEEE Frontiers in Education Conference*, pp. S1C/1-5, 2000
2. *The Practical Xilinx Designers Lab Book* by D. Van den Bout, Prentice Hall, 1999
3. Tcl Developer Xchange website, http://www.scriptics.com

DAVID SULLINS
David Sullins is a graduate student working on his MS in Computer Engineering at the University of Missouri – Rolla. He received his BS in Computer Engineering from the University of Missouri – Rolla in 2000. He has taught a laboratory course in hardware/software co-design for the past three semesters. He may be contacted at dsulli@umr.edu.

DARYL BEETNER
Dr Daryl Beetner is an Assistant Professor at the University of Missouri-Rolla. He received his BS degree in Electrical Engineering from Southern Illinois University at Edwardsville in 1990. He received an MS and DsC degree in Electrical Engineering from Washington University in St Louis in 1994 and 1997, respectively. Questions or comments regarding the paper may be sent through email to daryl@umr.edu.

HARDY POTTINGER
Dr. Pottinger is an Associate Professor at the University of Missouri – Rolla. He received his BS, MS, and PhD degrees from the University of Missouri – Rolla in 1966, 1968, and 1973 respectively. He has taught in the Department of Electrical and Computer Engineering at the University of Missouri – Rolla since 1979 and is currently Director of Computer Engineering for the department. He may be contacted at h.pottinger@ieee.org.