A Visual Tool to Enhance Comprehension and Design in Micro-processing Systems

K. Ferens

Department of Electrical and Computer Engineering
University of Manitoba, Winnipeg, Manitoba R3T 5V6, Canada

Abstract

This paper reports on the design and development of custom animation software to enhance comprehension and design of micro-processing systems. The purpose of the custom software is to be used as a tool for teaching second/third year undergraduate computer/electrical engineering students the basic concepts surrounding micro-instruction based microprocessors and systems. The tool enhances comprehension through a visual depiction of the structure and operation of a basic micro-instruction based microprocessor with memory. The control vector and control memory are visualized along with graphical methods of visually displaying the internal control of every device within the micro-processor and attached memory. The tool animates the sequence of micro-instructions of a given instruction by showing address and data transmission and paths juxtaposed against an animated clock. Effective use of the “water flowing through pipes” analogy enhances comprehension and visualization. In addition the tool facilitates the design of micro-instruction based microprocessors by allowing students to create and/or modify micro-instructions and create and/or modify macro-instructions. The tool speeds student learning and allows for more complex topics to be taught in the same semester.

1 INTRODUCTION

Many electrical and computer engineering departments in universities across Canada teach and or perform research on digital system design and development. A set of courses offered in succession provide students with the fundamental theory behind digital systems as well as the knowledge for designing and developing such systems.

The department of Electrical and Computer Engineering at the University of Manitoba offers the following stream of digital system courses:

- Digital Logic
- Micro-processing Systems
- Micro-processing Interfacing
- Digital Design 1
- Digital Systems Implementation
- Digital Design 2

The Microprocessing Systems course provides students with the fundamental theory behind a basic microprocessor [1]. The course begins with a review of Binary and Hexadecimal number systems. This review helps students realize that a digital system understands only the Binary language, i.e. 0V and +5V. Also, a binary number wheel is introduced, which helps students visualize what the digital system is doing when it performs addition and subtraction of unsigned and signed numbers.

The course then goes on to introducing the struc-

ture and operation of a tiny operation set calculator (TOC). The TOC is wired up on a breadboard in the lab. Thus, students get hands on experience with the basic concepts of binary encoded instructions, the stored program concept, data paths, macro- and micro-instructions, control vectors, control memory, and interfacing a data processor with a memory. The TOC is used a stepping-stone to explain the difficult-to-visualize concepts of a micro-processor. After describing and working with the TOC, students are presented with the structure and operation of a basic 8-bit micro-processor, roughly modeled after Motorola's 6800 family [2]. After describing the structure and operation of the basic processor, students are presented with assembly language and the instruction set. The fundamentals of assembly language programming is supported with three labs, including multiple-precision unsigned arithmetic, square roots of integers, and bubble sorting, which also provides students with experience with writing a subroutine and passing data via the stack.

After dealing with assembly language, including computer decisions and subroutines, the course provides an introduction to physical and procedural interfacing of the basic micro-processor with memories and I/O controllers, such as the Peripheral Interface Adapter (PIA). This is then followed by I/O fundamentals, including polling, interrupts, and Direct Memory Access (DMA) using the DMA controller (DMAC).

Finally, a few design examples are given, including a speech recorder and player, a home security sys-

cessor. (Refer to Fig. 1).

Fig. 1. The animation software's main page, showing an 8-bit micro-processor connected to a memory.

Although micro-processors in today's market run at much higher speeds, have wider data paths, and access much larger memory spaces, the basic 8-bit processor has many remaining advantages. About 75% of the micro-controller applications currently deployed today do not need to run at more than a few million cycles per second. This is because most of the applications deal with Human interaction or slow mecha-tronics devices. Furthermore, these applications do not need more resolution than 8-bit. Moreover, most of these applications do not require a huge amount of memory. Typically, these controllers do not even fill up a 64K memory space.

The structural representation of the central processing unit (CPU) should allow a conceptual categorization of the different components. In particular, the control functionality, data paths, and address generation should be arranged in separate units, called computer control unit (CCU), data path unit (DPU), and the address generator unit (AGU).

Since the size of the machine is 8-bit, an 8-bit arithmetic logic unit (ALU) shall be used. The output of the ALU shall connect to a bus, call it the result bus (RBUS), capable of routing data to other components in the AGU, DPU, and the devices in the memory space. To facilitate data provision, two 8-bit data registers shall be used, one for each side of the ALU. However, one of these registers shall act as an accumulator to accumulate intermediate results, via the RBUS, while the other register needs to act as a buffer between the memory and the CPU; therefore, we call this register the memory data register (MDR). This MDR also needs to buffer data between the memory and the CCU, in order to provide the CCU the opcode of an instruction. While one accumulator may be sufficient for a very simple microprocessor, we need another accumulator to help in data manipulation. Therefore, we have two accumulators, ACCumulator A (ACCA) and ACCumulator B (ACCB). Finally, in order to route data along the differ-

This section states the requirements and provides rational for animation software for a basic micro-pro-

ent paths of the DPU, several data path switches, called multiplexors, are required.

To allow for micro-programmability, the CCU shall provide a control memory to store the vectors for the micro-instructions. The CCU shall be equipped with a basic set of micro-instructions, to implement a small set of macro-instructions. These micro-instructions shall be modifiable, and the creation of new micro-instructions shall be possible. For each macro-instruction, the CCU shall contain a linked list of control vectors, each of which implements a micro-instruction. A linked-list is chosen so that no micro-instruction control vector needs to be duplicated. Since these control vectors will be stored in a control memory within the CCU, the opcode of the instruction shall be used as the key to the control memory. Furthermore, a micro-program counter shall be implemented to point to the next micro-instruction to be executed in the linked list.

The triggering of the machine shall be implemented using two clocks, identical except they shall be 180 degrees out of phase. By having two clocks, more edges and levels are obtained, and therefore, more concurrent operations may be performed. Furthermore, to minimize the cross-talk and inter symbol interference, the clocks shall be non-overlapping, so that when one clock is undergoing changes, the other clock is maintained level. Call these clocks, phase-1 and phase-2, or ϕ -1 and ϕ -2.

The AGU shall contain five registers, whose purpose shall be for generating addresses for memory and I/O controller's register access. Each of these registers shall be 16-bit, to implement a 64K memory space. A stack pointer (SP) shall be required for implementing a stack. The machine shall allow for any method of stack access, i.e., growing towards lower or higher memory locations, pushing and pulling and then decrement/increment or vice-versa. An index register (IX) shall be implemented to allow for the index addressing mode. There shall be a program counter (PC) register, whose purpose shall be to point to the next macro-instruction to be executed, as well as pointing to the next byte of an instruction during the micro-instruction execution. A temporary holding register (THR) is required for intermediate calculations of the effective address. Finally, a memory address register (MAR) shall be required to act as a port register between the other address generators and the devices in the memory space. All of the address generators shall have the capability of counting up or down (increment or decrement). The counting capability shall be triggered on an edge prior to the latch of data, to make possible concurrent operations. Furthermore, the address generators shall have the capability of sign extension, to allow for signed offsets in an addressing mode, as well as individual high byte of low byte access from/to the RBUS.

To facilitate address generation, a 16-bit adder/subtractor (16-bit AddSub) shall be contained within the

AGU. The inputs to the AddSub shall come from either the SP, IX, or PC, while the other input is fixed and comes from the THR. The output of the AddSub shall be connected to each 16-bit input of the AGU registers. Thus, an effective address may be calculated by simple addition or subtraction, and the result may be loaded into any one of the AGU registers.

To route the different address generators to the port address register, i.e., the MAR, a switch is required: call it the address generator multiplexor (AMUX). The AMUX shall have the capability of routing any AGU register to the input of the MAR or to one of the inputs of the AddSub.

To facilitate a visualization of instruction fetch and execute, a memory shall be present and connected to the micro-processor through the system bus, i.e., the address bus (ABUS), the data bus (DBUS), and control bus (CBUS).

Each component within the animation software's depiction of a basic 8-bit micro-processor contains control signals, which control the functionality of the device. For example, MUX2 needs the capability of either providing high impedance (choose nothing, via the MUX2_CN control signal) or routing data. When routing data, the MUX2 can either take input from the MDR or the RBUS and send the data to either one of the accumulators, ACCA or ACCB. In order to control the switching the data path within the MUX2, there are control signals MUX2_S1 and MUX2_S0, in which case MUX2_CN is false.

To configure a micro-instruction, the state of each control signal for every device within the system must be established. The set of control signals required to implement a micro-instruction is called a control vector. To facilitate control vector configuration, a table shall be provided in the *Configure μ OPs* tab, which shows all of the components of the control vector, with the name of the micro-instruction as the key. A list of control vectors is obtained. This list shall be stored in the CCU's control memory. Thus, a micro-instruction set may be established for the machine.

To configure a macro-instruction, a Build Instruction tab shall be provided. This tab shall provide a table and means to enter information about the instruction, such the opcode, the number of micro-instructions, and the name of each micro-instruction. This table shall be stored in the CCU's control memory.

The machine shall allow for using either the Big-Endian or Little-Endian word format. This will facilitate comprehension of these different data word formats.

4 EXPERIMENTS AND ANALYSIS

A number of experiments were conducted to show requirements verification of the animation software. One of the more interesting experiments is described in this paper.

Table 1: Improving loop efficiency.

Loop to Improve				Improved Loop			
Cycles	Label	Instruction		Cycles	Label	Instruction	
3		LDX	#start_block	3		LDX	#start_block
4	LOOP	CLR	\$00,X	4	LOOP	CLR	\$00,X+
2		INX					
4		CPX	#end_block	4		CPX	#end_block
3		BNE	LOOP	3		BNE	LOOP
Cycles through one loop=13				Cycles through one loop=11			

4.1 INDEX ADDRESSING WITH POST/PRE INC/DEC

Consider improving the simple loop shown on the left in Table 1. The purpose of this experiment is to design a more efficient addressing mode which mitigates loop overhead. and thus, increases the speed at which a loop is executed. This new addressing mode is the same as the INDEXED X addressing mode (IND X), except the new addressing mode (IND X+) increments X concurrently while an instruction using the IND X+ addressing mode performs its main function.

To test the IND X+ addressing mode, a loop was created to clear a block of memory. In the original method (left side of Table 1), the CLR operation uses the IND X addressing mode. This loop requires 13 machines cycles to execute. The right side of Table 1 shows the implementation of the IND X+ addressing mode, and the loop only requires 11 cycles to execute. This is because the INX instruction is not need when the IND X+ addressing mode is used in the CLR operation. The loop overhead function of incrementing the IX register is performed concurrently with the instruction CLR \$00,X+.

To implement and thus verify the IND X+ addressing mode, the CLR \$00,X+ instruction was created and animated (using three steps) in the animation software.

(Step 1) First, the macro-instruction was built and entered into the animation software via the Build Instruction tab (see Fig. 2).

Fig. 2. Macro-instruction configuration for the instruction utilizing the IND X+ addressing mode.

(Step 2) Next, the addressing mode IND X was analyzed and determine that the X register may be concurrently incremented only after the effective address has been calculated, because the present value of X cannot change during the calculation. In the instruction “CLR \$00,X+”, the last micro-instruction is the only micro-instruction in which X can be incremented. This is because the effective address has already been generated and stored in the THR, and the value of the X register may be changed without affecting the desired “CLR” operation of the macro-instruction. Therefore, we cre-

ated four micro-operations: “fetch”, “loadOffset”, “formEA”, and “clear (EA)”, as shown in Fig. 2. To complete the configuration of the instruction, the “Configure mOPS” tab was chosen, and four control vectors were created and entered into the table (Fig. 3).

Note the other interesting aspects of the animation. In particular, the “water flowing through pipes” animation method provides an element of visual learning that is simple and effective. The animation visually shows how the micro-processor controls memory devices by issuing an address, which is followed by data flow. In addition, by juxtaposing the animation of the clock with the “water flowing through pipes” animation, graphically shows that a time-ordered sequence of events are required to execute a macro-instruction. The animation software makes it possible to visually correlate address and data flow with certain clock edges and levels.

5 SUMMARY AND FUTURE WORK

This paper describes an animation tool for enhancing design and comprehension of a micro-program based micro-processor. The animation tool provides the ability to create and modify macro-instructions, micro-instructions, and control vectors. The “water flowing through pipes” animation method effectively demonstrates concepts otherwise difficult to communicate using standard teaching modalities. Future work includes addition of new features, such as the animation of instruction and data caches, pipelines, and branch prediction.

ACKNOWLEDGMENTS

This paper has its foundations built from Dr. Kin-sner’s lectures on computer architecture in the Department of Electrical and Computer Engineering, University of Manitoba. Dr. Poskar assisted in the development of the animation software.

REFERENCES

- [1] K. Ferens, “Micro-Processing Systems,” Micro-processing Systems (ECE 3610) Course Notes, Winnipeg, MB: Department of Electrical and Computer Engineering, University of Manitoba, 1994.
- [2] Motorola 6800, Wikipedia, the free encyclopedia; http://en.wikipedia.org/wiki/Motorola_6800.
- [3] Freescale Semiconductor, “University Programs”, <http://www.freescale.com/>.
- [4] Freescale Semiconductor Technical Staff, “M68EVB912B32 Evaluation Board: M68EVB912B32/D Rev. 1”, Freescale Semiconductor, Inc., 1999.
- [5] Bilderbeek, W., Broeders, H., and van Rooijen, A., “The THRSim11 68HC11 Simulator,” Dr. Dobb’s Journal, July 22, 2001.
- [6] Wolffe, G., Yurcik, W., Osborne, H., and Holliday, M., “Teaching computer organization/architecture with limited resources using simulators,” Proceedings of the 33rd Technical Symposium of Computer Science Education (SIGCSE 2002), ACM Press, Northern Kentucky USA, Feb./March 2002.
- [7] Yehezkel, C., Yurcik, W., and Pearson, M.

“Teaching computer architecture with a computer-aided learning environment: state of the art simulators,” Intl. Conference on Simulation and Multimedia in Engineering Education (ICSEE), Society for Computer Simulation Press, Phoenix AZ USA, January 2001.

[8] Yurcik, W., Wolffe, G., and Holliday, M. “A Survey of simulators used in computer organization/architecture courses,” Proceedings of the Summer Computer Simulation Conference (SCSC), Orlando FL USA, July 2001.

[9] Yurcik, W. “The Simulation Education Homepage,” SIMULATION: Journal of the Society for Modeling and Simulation Intl., April 2001, pp. 202-206.