URISC: The Ultimate Reduced Instruction Set Computer

Farhad Mavaddat
and
Behrooz Parhami
Department of Computer Science

Research Report CS-87-36
June 1987

URISC: The Ultimate Reduced Instruction Set Computer

FARHAD MAVADDAT and BEHROOZ PARHAMI

Department of Computer Science
University of Waterloo

ABSTRACT

URISC is a single-instruction universal computing machine which appears to be ideal for the introduction of basic computer organization concepts to novice students. Even students not majoring in computer science or computer engineering consider URISC to be "fun" and easy to understand. In this paper, the specifications of URISC are given and complete microprogrammed and hardwired implementations of its control unit are presented. The authors' experiences with the use of URISC in classroom environment are discussed briefly.

Index Terms -- Computer architecture, Computer organization, Computer science education, Control unit design, Hardwired control, Microprogramming.

1. INTRODUCTION

The usual practice in introducing the students to the concepts of hardwired and microprogrammed control is to consider the sequence of steps needed for executing some simple instructions and to specify a partial design for the sequencing unit in terms of the needed logic circuits in AND-OR form or the bit patterns in horizontal microprogramming format. While most students are capable of extending the ideas to more complex instructions, they are usually bothered by the fact that many loose ends still remain and that they may not be able to carry out a complete design in practice. This is aggravated by the exceedingly complex instruction sets of most modern computer systems (e.g., VAX-11) with which the students become familiar.

The idea of using a toy computer with a very simple instruction set as an educational tool is not new. Many computer organization texts develop such designs from scratch [FOST85], [GORS86], [TANE84] or attempt to present a simplified model of some existing machine [WAKE81]. Development of the concept of reduced instruction set computers (RISCs) [BELL86], [PATT82], [PATT85], [RADI83], [STAL86], [WALL85] as an alternative to complex instruction set computers (CISCs) [COLW85] has been helpful for educators who now have at their disposal actual implemented systems which are not too complex to be explained to novices. Despite all these, the instructor of an introductory computer organization course may find it difficult to present the detailed design of a complete computer and still cover all the basic concepts of the field within the time limitation of a single quarter or semester.

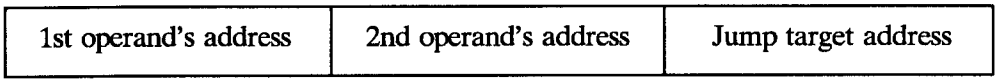
We believe that the URISC concept provides the educators with a powerful tool for teaching computer organization concepts in the framework of a complete design in such a way that the resulting design can even be implemented in a microprocessor or logic design laboratory if desired. The students derive a great deal of satisfaction from the fact that they are able to design an actual working system from scratch; not from microprocessor chips but from standard low-level components such as registers, counters, adders, and logic gates. In addition, the URISC concept may be applicable to the practical design of VLSI systems in cases where the technology being utilized does not support the level of integration required by single-chip CISC designs [MCNE87].

In the remainder of this paper, first the specifications of URISC are presented along with typical "coding" assignments to convince the students that URISC is in fact a universal machine (students with a background in automata theory may need less convincing). Then, in Section 3, a complete microprogrammed implementation is described. This is followed by a refreshingly simple hardwired design in Section 4. Finally, Section 5 contains our conclusions based on two years of actual use of the URISC concept in classroom environment.

2. URISC SPECIFICATIONS

Reduced instruction set computers (RISCs) are machines with a small number of simple, fixed-length, fixed-format instructions. The fact that such instructions are often executable in a single processor cycle (thus resulting in high performance in terms of instruction execution speed) has prompted some computer designers to advance RISCs as the most cost-effective alternative for high-speed computation. Here, we are not so much concerned with the speed of instruction execution. Rather, we are striving for simplicity in machine specification and design.

Consider a machine with a single 3-address instruction as the ultimate in RISC design (URISC). The format of the single URISC instruction is as follows:



To execute a URISC instruction, the CPU subtracts the 1st operand from the 2nd, returning the result to the 2nd operand's location. It then jumps to the specified "jump target address" if the subtraction result is negative; otherwise it proceeds with the execution of the next instruction in sequence. A jump to memory location 0 will stop the machine. Obviously, with a single instruction, there is no need for an opcode field in the instruction.

In the following discussion, we will assume the existence of an assembly-level notation for this machine in the form of

<L> : <A>, <B>, <P>

where <L> and <P> are the instruction label and jump target label, respectively, and <A> and <B> are mnemonic references to memory locations holding the 1st and 2nd operands, respectively. In addition we assume the availability of an assembly-type pseudoinstruction in the form of

<L> : WORD <C>

to initialize the location named <L> with the constant value <C>. By convention, all programs are loaded starting at memory location 0 and are executed starting at memory location 1. In assembly language, every program has

STOP : WORD 0

as its first instruction for proper program termination. A sample program fragment follows:

```
STOP  : WORD 0
START : DEST, DEST, ADD
ADD   : SRC,  TEMP, NEXT
NEXT  : TEMP, DEST, CONT
CONT  : ...
        ...
TEMP  : WORD 0
```

The above program fragment "moves" the contents of the memory location SRC to the memory location DEST.

That the following effects are also easily achievable by the single URISC instruction should be sufficient to convince most people that URISC is indeed a universal machine.

a. Adding the contents of M1 and M2, storing the sum in M3; i.e. M3 $\leftarrow$ (M1) + (M2).

b. Exchanging the contents of M2 and M1; i.e. (M1) $\longleftrightarrow$ (M2).

c. Jumping to L1 if the contents of M1 is not less than the contents of M2.

d. Jumping to L if the contents of M1 and M2 are equal.

Item (a) above shows that addition can be programmed using the machine's subtraction capability. Items (c) and (d) imply that other numerical comparisons (less than or equal to, greater than, etc.) can be easily programmed. As an interesting programming exercise, the students may be asked to write a bubble sort routine for URISC.

3. MICROPROGRAMMED URISC

Consider an implementation of URISC which employs a 64K×16 (64K words by 16 bits) memory. Therefore, all instructions are three words long, with each operand or destination address occupying a full 16-bit word. The micro-architecture of Figure 1 is used where all registers and buses are 16 bits wide and a 16-bit adder is permanently wired to form the binary sum of its two 16-bit inputs, with or without an input carry depending on the presence or absence of the C_{in} micro-order. One input of the adder is permanently connected to the left bus. The other input can receive either the logical complement of the contents of R (COMP micro-order) or the value 0 (no micro-order specified). The adder output is permanently connected to the right bus. If the Z_{in} or N_{in} micro-order is specified, the "condition" flip-flop Z or N will be set whenever the addition result is zero or negative, respectively.

In addition to the control signals shown in Figure 1, there are two conditional jump micro-orders: ZEND (zero end) resets the microprogram counter to zero if the Z condition flip-flop is set. NNEND (non-negative end) resets the microprogram counter if N is *not* set. Both ZEND and NNEND take effect upon the completion of other activities within the same microinstruction. For example, the following microinstruction causes all subsequent microinstructions to be ignored and control transferred to the first microinstruction if the sum of the complement of R and the contents of MDR is non-negative:

$$\text{COMP, MDR}_{out}\text{ , N}_{in}\text{ , NNEND}$$

All registers are made of master-slave flip-flops; therefore, they may output and input data within the same microinstruction. The memory is assumed to be sufficiently fast, so that a memory function initiated in one microinstruction is completed before the execution of the next microinstruction.

Here is the complete microprogram required for this URISC to continuously fetch and execute instructions, assuming that PC initially points to the first word of an instruction and that the microprogram counter counts modulo 9 (it is automatically reset after it reaches 8). This last assumption can easily be relaxed.

0. PC_{out} , Z_{in} , MAR_{in} , READ, ZEND
1. MDR_{out} , MAR_{in} , READ
2. MDR_{out} , R_{in}
3. PC_{out} , C_{in} , PC_{in} , MAR_{in} , READ
4. MDR_{out} , MAR_{in} , READ
5. MDR_{out} , COMP, C_{in} , N_{in} , MDR_{in} , WRITE
6. PC_{out} , C_{in} , PC_{in} , MAR_{in} , READ
7. PC_{out} , C_{in} , PC_{in} , NNEND
8. MDR_{out} , PC_{in}

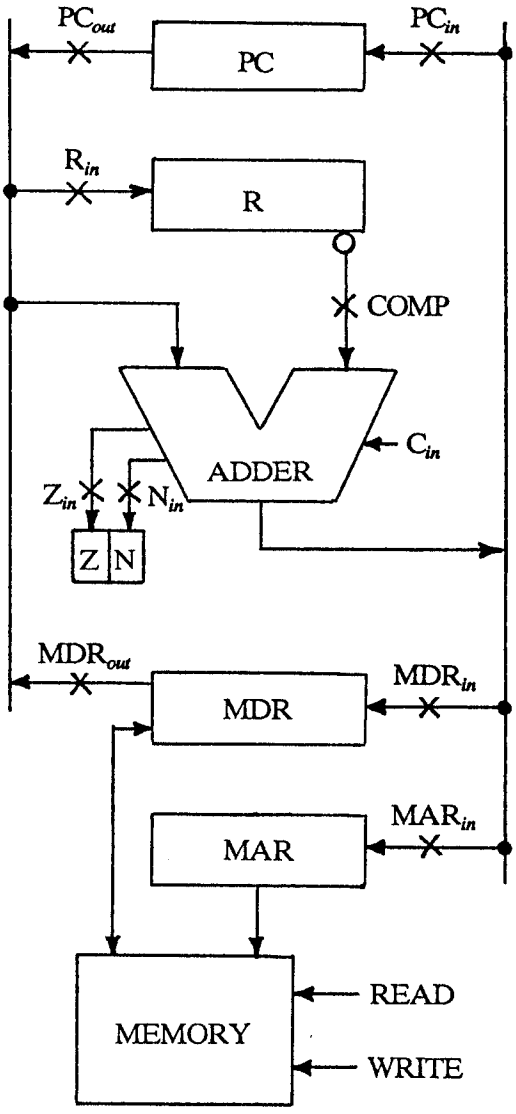


Figure 1. Organization of a Microprogrammed URISC.

4. HARDWIRED URISC

The set of Boolean equations specifying each control signal in a hardwired implementation of URISC (as specified in Section 3) can be written directly from the preceding microprogram. Assuming that a modulo-9 sequence counter produces the timing signals ϕ_0 through ϕ_8 corresponding to the 9 microinstructions (execution steps), we have:

$$
\begin{aligned}
PC_{in} &= \phi_3 + \phi_6 + \phi_7 + \phi_8 \\
PC_{out} &= \phi_0 + \phi_3 + \phi_6 + \phi_7 \\
R_{in} &= \phi_2 \\
COMP &= \phi_5 \\
C_{in} &= \phi_3 + \phi_5 + \phi_6 + \phi_7 \\
Z_{in} &= \phi_0 \\
N_{in} &= \phi_5 \\
MDR_{in} &= \phi_5 \\
MDR_{out} &= \phi_1 + \phi_2 + \phi_4 + \phi_5 + \phi_8 \\
MAR_{in} &= \phi_0 + \phi_1 + \phi_3 + \phi_4 + \phi_6 \\
READ &= \phi_0 + \phi_1 + \phi_3 + \phi_4 + \phi_6 \\
WRITE &= \phi_5 \\
ZEND &= \phi_0 Z \\
NNEND &= \phi_7 \overline{N}
\end{aligned}
$$

Figure 2 shows a complete design for URISC's hardwired controller. A four-bit hex or decade counter and a 3-to-8 decoder are used to provide the timing signals ϕ_0 through ϕ_8.

The above equations are also useful in the design of an "optimized" microprogrammed version since they help the students to realize the following equivalences among the micro-orders:

$$
\begin{aligned}
&COMP = N_{in} = MDR_{in} = WRITE \\
&MAR_{in} = READ \\
&MDR_{out} = \overline{PC}_{out}
\end{aligned}
$$

Thus, only 9 bits are required in each microinstruction rather than 14 (as implied by the 12 control signals shown in Figure 1 plus the two micro-orders ZEND and NNEND). The length of microinstructions is further reduced to 8 bits by noting that the ZEND micro-order can be made redundant if the output of the Z flip-flop is connected directly to the "clear" input of the microinstruction counter, since Z can be 1 only during ϕ_0. This property is utilized in the design depicted in Figure 2, which in addition uses only NOT and two-input OR gates to facilitate the implementation.

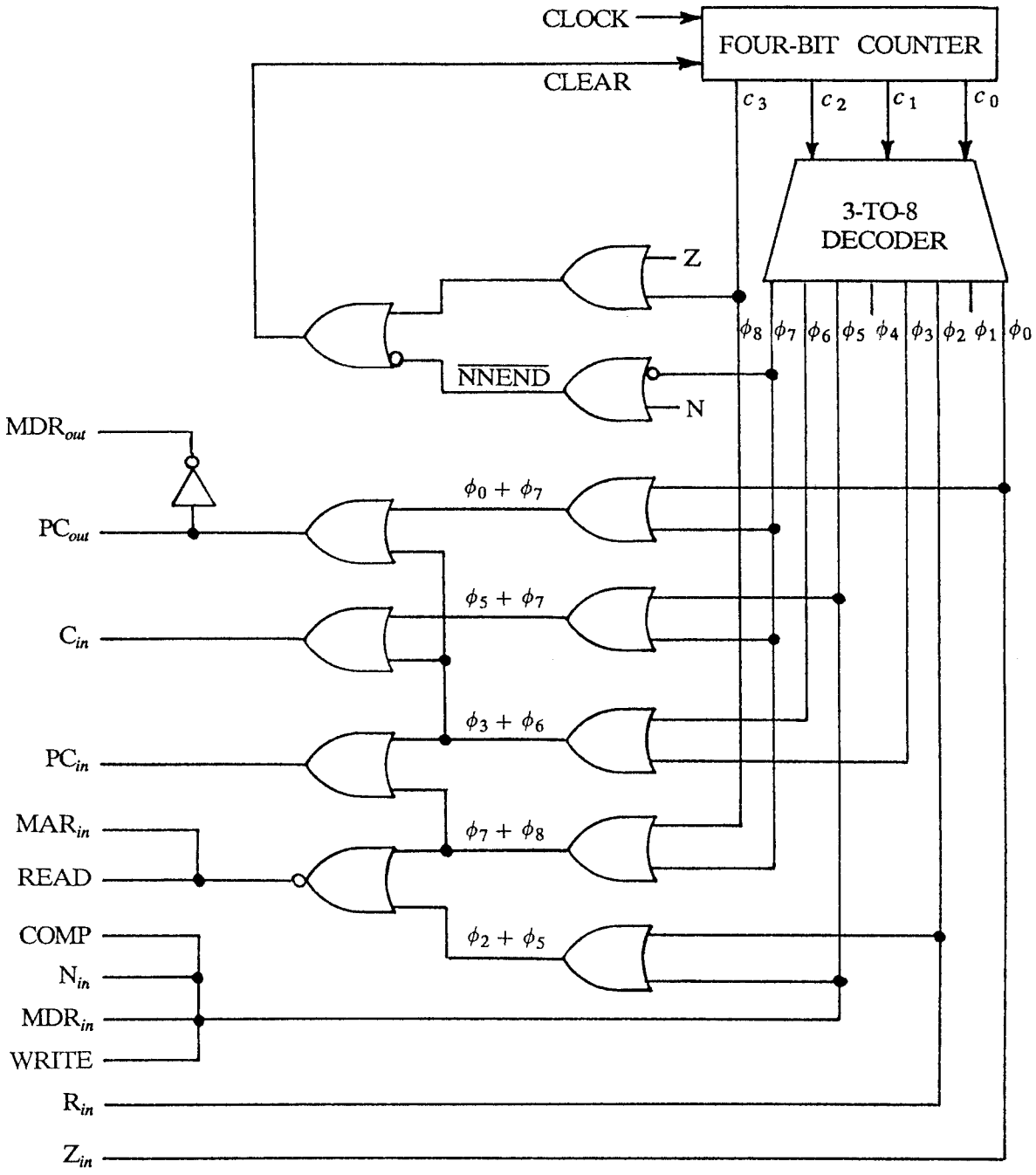


Figure 2. Hardwired Controller Design for URISC.

5. CONCLUSIONS

We have described URISC as a powerful tool for teaching fundamental concepts of control unit design and microprogramming. The simple version of URISC discussed in Sections 3 and 4 can be extended in several different ways in order to introduce more advanced concepts. For example, the "ultrafast memory" assumption can be relaxed by introducing the MFC (memory function complete) signal and the WMFC (wait for MFC) micro-order which will be appended to a microinstruction before each memory-referencing microinstruction. As another example, the ZEND micro-order can be replaced by the ZHALT micro-order which causes microinstruction fetch and execution to stop completely when PC contains zero and to restart when a "RUN" button on the machine's control console is pressed. Elementary I/O facilities can be added by the provision of switches and lights on the control console and the addition of appropriate control circuits or micro-routines.

Several versions of URISC have been used by the authors in computer systems courses at the University of Waterloo. At one instance, microprogramming of an earlier version of URISC was given as a final exam question constituting 10% of total marks in a 3-hour session. Most students did well in coming up with a correct and efficient microprogram, considering the fact that they were facing the URISC concept for the first time. The average mark for the 102 students was about 70% (75% if blank answers were ignored). A common mistake was to treat the first and second parts of an instruction as operands rather than addresses. In another case, the design of URISC was given as an assignment problem for non-computer-science majors. Again, the performance of the 90 students who handed in the assignment was quite encouraging. In all cases, the students were amused by the idea of URISC and found it an interesting and useful exercise.

Aside from the obvious educational value for computer organization courses, the concept of URISC is interesting for at least two other reasons. First, it serves to show that very simple arithmetic and conditional jump capabilities are sufficient for the universality of a computing device. Even though such an observation is readily made in automata theory courses, it becomes much easier to understand when derived from an actual computer system with a familiar programming model. Second, VLSI implementation of a hardwired URISC requires a very small area and can be made quite fast. It is conceivable that a large collection of such simple processors can be gainfully employed in certain large computing problems which lend themselves to cooperative solution. For this purpose, a simple communication mechanism must be added to the minimal URISC design presented in this paper.

ACKNOWLEDGEMENT

The work reported here was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A5515.

REFERENCES

[BELL86] Bell, C.G., "RISC: Back to the Future," *Datamation,* Vol. 32, No. 11, pp. 96-108, 1 June 1986.

[COLW85] Colwell, R.P. et al, "Computers, Complexity, and Controversy," *Computer,* Vol. 18, No. 9, pp. 8-19, Sep. 1985.

[FOST85] Foster, C.C. and T. Iberall, *Computer Architecture,* Van Nostrand Reinhold, New York, 3rd Edition, 1985.

[GORS86] Gorsline, G.W., *Computer Organization: Hardware/Software,* Prentice-Hall, Englewood Cliffs, NJ, 2nd Edition, 1986.

[MCNE87] McNeley, K.J., "Emulating a Complex Instruction Set Computer with a Reduced Instruction Set Computer," *IEEE Micro,* Vol. 7, No. 1, pp. 60-72, Feb. 1987.

[PATT82] Patterson, D.A. and C.H. Sequin, "A VLSI RISC," *Computer,* Vol. 15, No. 9, pp. 8-18, Sep. 1982.

[PATT85] Patterson, D.A., "Reduced Instruction Set Computers," *Communications of the ACM,* Vol. 28, No. 1, pp. 8-21, Jan. 1985.

[RADI83] Radin, G., "The 801 Minicomputer," *IBM Journal of Research and Development,* Vol. 27, No. 3, pp. 237-246, May 1983.

[STAL86] Stallings, W. (Editor), *Reduced Instruction Set Computers,* IEEE Computer Society Press, 1986; The annotated bibliography from this volume is reprinted in *Computer Architecture News,* ACM SIGARCH Publication, Vol. 14, No. 5, pp. 13-19, Dec. 1986.

[TANE84] Tanenbaum, A.S., *Structured Computer Organization,* Prentice-Hall, Englewood Cliffs, NJ, 2nd Edition, 1984.

[WAKE81] Wakerly, J.F., *Microcomputer Architecture and Programming,* Wiley, New York, 1981.

[WALL85] Wallich, P., "Toward Simpler, Faster Computers," *IEEE Spectrum,* Vol. 22, No. 8, pp. 38-45, Aug. 1985.