C-RISC
A C Language Reduced Instruction Set Computer

MIKE STARKEY
Department of Computer Science

ALI REZA FARHANG
Department of Electrical Engineering

UUCS-90-012

C-RISC

A C language Reduced Instruction Set Computer

UUCS-90-012

Mike Starkey
Department of Computer Science

Ali Reza Farhang
Department of Electrical Engineering

University of Utah

Abstract

This project is the implementation of a Reduced Instruction Set Computer (RISC) on a tiny chip. RISC technology is based on the idea that a small number of simple instructions can be used to create a fast, flexible computer. Our RISC uses this principle while staying within the confines of the tiny chip.

Size limitations directly affect the complexity in the number of possible instructions. Since the tiny chip is limited in size, a decision was required as to the set of instructions to include. Due to the implementers' knowledge of the C language, it was decided that the direction of the RISC instructions would follow the C language instruction set, hence the name C-RISC. The decision to direct the instruction set to the C language instructions does not restrict the flexibility of the C-RISC since many high level languages (HLLs) can be converted or cross-compiled into C code. The adoption of a given language direction allows easier comparison of quantitative results when deciding on instructions to be included in or discarded from the instruction set.

A number of ingenious ideas were incorporated into the design of the C-RISC. These were necessitated by the small chip size as well as the desired power of the processor. The final C-RISC chip was implemented in Path Programmable Logic (PPL) [1] using 2 micron CMOS technology.

This report placed third in the 1990 National Tiny Chip competition.

Background

The RISC technology works on the idea that a small set of relatively simple instructions will provide maximum flexibility and speed. Since implementing an instruction directly can greatly speed up an operation, the greatest increase in execution speed for a program can be realized if the most commonly executed instructions are reduced to one RISC instruction. The two most important instructions in any RISC are load and store due to the large utilization of registers. Only after these two instructions have been considered, can additional instructions be selected for inclusion in the instruction set. Figure 1 shows a breakdown of the remaining instructions. It can be seen that the majority of the most commonly executed instructions have been implemented on the C-RISC chip.

C Instruction		Further Breakdown				C-RISC Inst.	Usage
Assignments	42%	1-operand	66%	constants	20%	MOVI	5.54%
				scalars	55%	MOV	15.25%
				arrays	25%		6.93%
		2-operand	20%	constants	20%		1.68%
				scalars	55%	ADD	4.64%
				arrays	25%		2.10%
		Other	14%				5.88%
If	36%						
Loop	4%						
Branches		40%		unconditional	55%	BR	22.00%
				not equal zero	30%	BRLT	12.00%
				equal zero	15%	BREQ	6.00%
Call/Return		14%		3 arguments or less	95%	CALL	13.30%
				Other	5%		0.70%
Other		4%					4.00%

Figure 1. The usage of C language Instructions
Source: [3]

Architecture

Instructions: The instruction is defined as IR since it is present in the Instruction Register for decoding. The individual bits are specified as $i_7i_6i_5i_4i_3i_2i_1i_0$.

Opcode: The opcode has static size and will be referred to as $i_7i_6i_5$.

Operand 1: The first operand, bits i_4i_3, will indicate either a register offset into the current register bank (r_1r_0) or a sub opcode (s_1s_0). Bits (s_1s_2) are used for some opcodes to allow a number of instructions for the same opcode.

Operand 2: The second operand, bits $i_2i_1i_0$, indicates either a register R consisting of bits ($r_2r_1r_0$) or a number ($n_2n_1n_0$).

Bank Bit: A single bit prefixed onto the register offset (r_1r_0) to give the register number.

Condition code: The Condition Code consists of two bits, C_N and C_Z. These are used for indicating negative and zero conditions, respectively. The Condition Code is calculated from the result of the adder.

Function	Instruction	Definition
TEST	00000_R	TEST[R]
INCR	00001_R	[R] = [R] + 1
NOT	00010_R	[R] = ![R]
COMP	00011_R	[R] = -[R]
MOV	$001r_1r_0$_R	[R] = $[Br_1r_0]$
MOVI	$010r_1r_0n_2n_1n_0$	$[Br_1r_0] = 00000n_2n_1n_0$
CLEAR	$010r_1r_0000$	$[Br_1r_0] = 00000000$
ADD	$011r_1r_0$_R	$[Br_1r_0] = [Br_1r_0] + [R]$
RET	10000000	PC = [B00]; B = !B
STORE	$101r_1r_0$_R	M[R] = $[Br_1r_0]$
BR	11000_R	PC = [R]
BREQ	11001_R	if C_NC_Z = 01 then PC = [R]
BRLT	11010_R	if C_NC_Z = 10 then PC = [R]
CALL	11011_R	B = !B; [B00] = PC; PC = [R]
LOAD	$111r_1r_0$_R	$[Br_1r_0]$ = M[R]

PC - Program Counter

Figure 2. C-RISC Instruction Set

The opcodes are designed to provide as much information as possible to reduce the amount of decoding required. The i_7 bit is used as an ALU and CC bit. If this bit is '0'B, the condition codes are stored for the instruction and the instruction is known to require one clock cycle. The i_6 indicates the destination for storing the result from the instruction. A value of '0'B for i_6 indicates that operand 2 is the destination, and a value of '1'B for i_6 indicates that operand 1 is the destination. The instruction set is also designed so that the instruction '00000000'B (TEST) will have little effect on the chip and its surrounding environment since this instruction is the first instruction in the IR after resetting the chip. Figure 2 lists C-RISC's instruction set.

A process called Banking is implemented on the C-RISC. The eight registers in the register file are divided into two banks (Registers 0-3 are in bank one and registers 4-8 are in bank two). Since we have only two banks, only one bit is needed to keep track of which bank is currently being accessed at any time. This bit is called the Bank bit and is maintained internally. Refer to Figures 2 and 6 for more details on how the Bank bit is manipulated. Banking was implemented for two major reasons; 1) It allowed us to implement more instructions in our instruction set by giving us an extra bit to work with in the opcode. 2) It gives us a form of data protection during a procedure call. Having only two banks gives us the ability to have procedure calls up to one level deep. According to Dr. Katevenis [3], 96% of all procedure calls are four levels deep or less. This would indicate that two

banks are not satisfactory. However, procedure calls of any level can still be implemented with the C-RISC by using software programs (Part of the Operating System program) which save the contents of a given bank in main memory before giving write access for that bank to the processor.

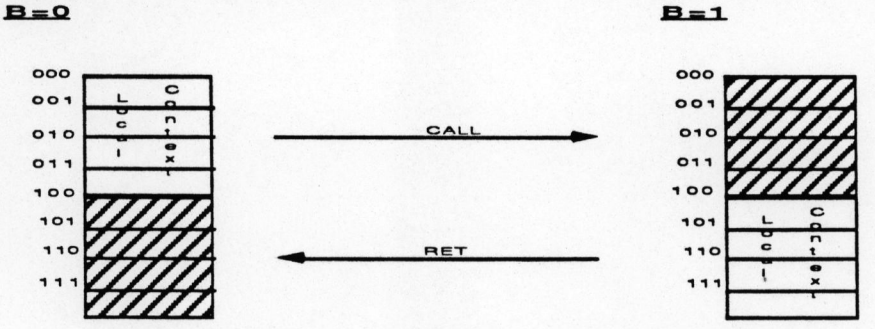

Figure 6. Bank Bit Manipulation showing Local Context

Single Clock Cycle Instructions

Single cycle instructions only require the first phase to get data from the registers and simultaneously present the new instruction address on the address bus, and store the result from the registers on the second phase cycle. These are all Arithmetic Logic Unit (ALU) instructions and therefore have the most significant bit (i_7) as a '0'B. The operation of these instructions is detailed in Figure 3.

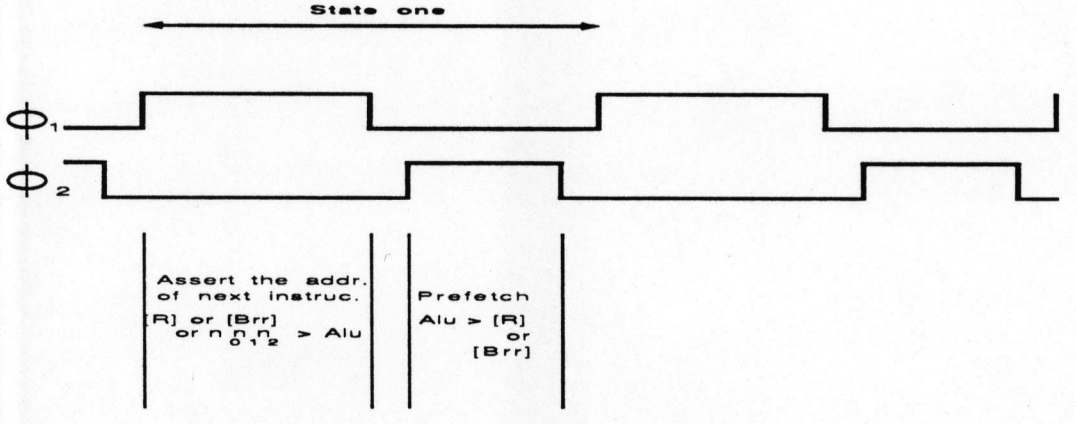

Figure 3. Single Cycle Instruction Clocking

The most important instruction in this class of instructions is the complement (COMP) instruction which gives the C-RISC the ability to subtract. However, while implementing COMP, it was found that it is easy to also include some other useful instructions. The increment (INCR) instruction is very useful in the C Language since incrementing with commands such as i++ is very

commonly used. The test (TEST) instruction is also very useful for setting the Condition Codes for a value in a register. The not (NOT) instruction, which inverts all bits, falls out as a side effect and may be useful. These instructions are implemented by EXORing the bits of the operand with i_4 and then adding i_3 to the result.

Double Clock Cycle Instructions

Double clock cycle instructions require two complete clock cycles to complete. These instructions either change the address of the next instruction to be executed (Branching instructions or procedure call instructions) or require the address bus to transfer data during the first clock cycle (Instructions that require loading to and from memory). In both cases, a second clock cycle is required to prefetch the next instruction. Figure 4 describes the operation of these instructions.

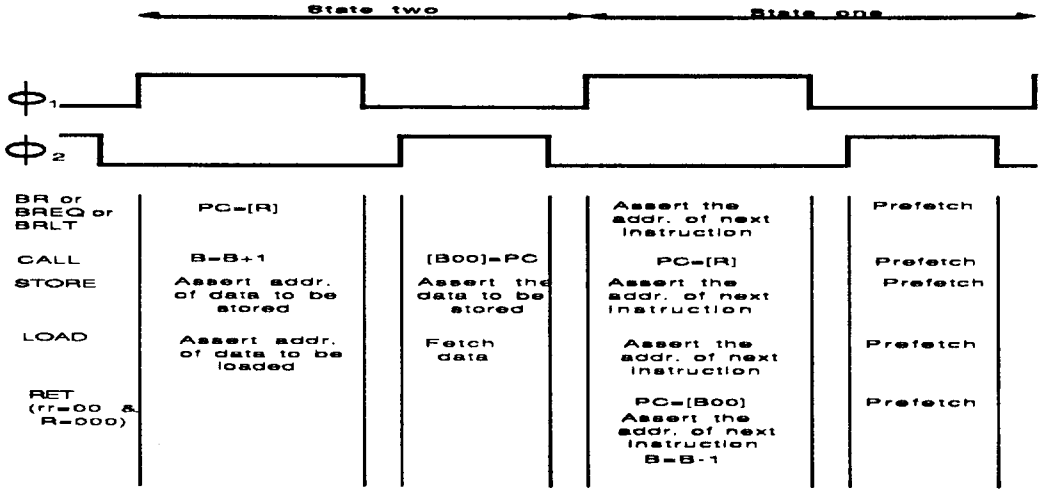

Figure 4. Double Cycle Instruction Clocking

Detailed Design

All major blocks of the C-RISC (ALU, PC, CC, register file, etc.), have been designed in such a way as to control themselves. This allows us to eliminate the requirement of a large complicated controller. The outputs of the IR (This register contains the eight bit Instruction) are carried up along the left side of the chip. This gives all the major blocks access to the entire Instruction which each uses for internal decoding. Figure 5 shows the block diagram of the C-RISC chip and also shows where the major blocks are positioned in the tiny chip.

As previously mentioned, the instruction set has been very carefully designed in a way to minimize the amount of decoding necessary for each instruction in the instruction set. It can be seen

from Figure 2 that all the one cycle instructions (ALU instructions) have Instruction Codes beginning with a zero (i_7=0) and all the two cycle instructions have Instruction Codes beginning with a one (i_7=1). This simple design makes the decoding performed by the (two state) state machine trivial. Another simple and unique design used in the instruction set is the fact that conditional branching is only performed when the i_3 and i_4 bit of the Instruction Code are equal to the C_N and C_Z bit of the Condition Code Register respectively. This simplifies the amount of decoding performed by the PC. These and other designs integrated into the design of the C-RISC instruction set have reduced the area needed by the processor for decoding.

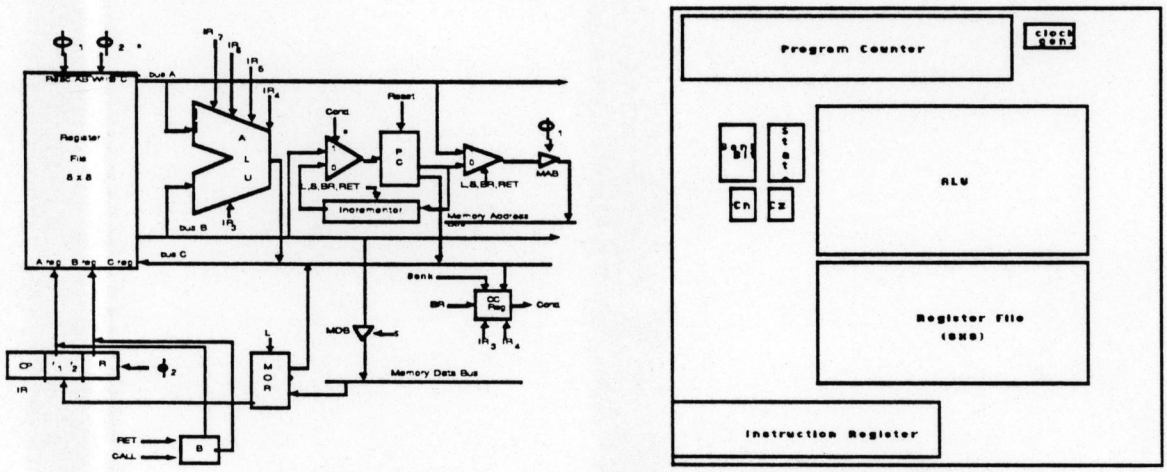

Figure 5. Block Diagram and a Major Block Map of C-RISC

Testing

The entire C-RISC PPL layout is presented in Figure 6. The PPL layout was then extracted and 3250 transistors were generated. Testing was performed on the extracted chip using SIMPPL[1]. A number of programs were written in C-RISC machine code which were converted into simulation input. Five programs were written which tested features incrementally. These programs included almost 100% coverage of all instructions. All instructions executed properly.

Four chips were fabricated through MOSIS. The estimated speed of the chips is 25 MHz, however a fast tester was not available and this speed could not be verified. The chips were tested using a CADEC tester at 1 KHz. This speed was fast enough to test some instructions however none of the programs would execute in their entirety since the dynamic cells used in C-RISC require a faster clock frequency.

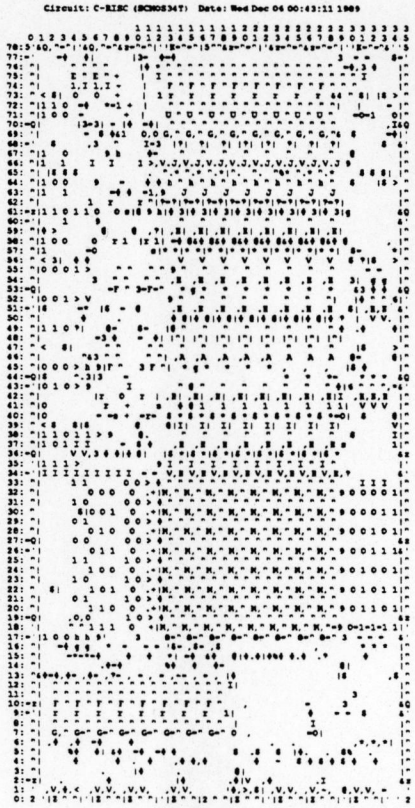

Figure 6. PPL Layout of C-RISC

Conclusions

The C-RISC chip is an interesting implementation of RISC technology. It uses some of the concepts developed by the Berkeley RISC group while being constrained to a fairly small chip size. The deviation from the PRISC architecture was a good learning experience and caused greater appreciation for the design considerations of processors and instruction sets.

After programming the chip, it was realized that some additional instructions would have been useful. A Load Immediate instruction which loaded the next byte after the current Program Counter address would have eased programming considerably.

References

1. T. Carter, "Course Notes for Computer Science 539/Electrical Engineering 560", Kinko's Professor Publishing, 1989.
2. R. Ginosar, A. Harsat, "PRISC: A PPL-Based Reduced Instruction Set Computer", Technical Report, Department of Electrical Engineering, The Technion - Israel Institute of Technology.
3. M. Katevenis, "Reduced Instruction Set Computer Architectures for VLSI", The MIT Press, 1985.