A RISC Processor with Extended Forwarding

Gert Markwardt* Günter Kemnitz** Reiner G. Spallek**

* Siemens AG

** Technische Universität Clausthal

*** Technische Universität Dresden

E-Mail: gkemnitz@informatik.tu-clausthal.de

Abstract

The paper examines a simple conceptual modification of the operation unit of a RISC processor. We propose to substitute a part of the conventional general purpose register file by a shift register for all operation results. The presented approach allows to reduce the instruction size for a great deal of instructions and so the instruction stream, and it is also a promising approach to make the processor architecture more regular.

1 Introduction

A basic idea of the RISC concept was to optimize functionality and simplify the computer structure. The result, a computer with less and simpler instructions could run a program as fast as a complex computer [1]. Now the descendants of the RISC prototype are even more complex. So, another RISC-like simplification would be nice. At the first glance, our proposed shift register file will not look like a simplification. But it has the potential of solving some difficult problems of modern superscalar processors in a simple way.

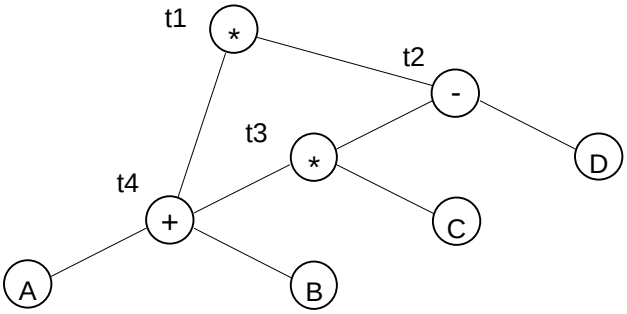
The idea originates from our research to reduce the instruction stream of RISC processors [2]. In this context, we could observe the following feature of RISC programs. Most results of calculations and load operations are used as operands only a few instructions after calculation. Storing them in a shift register instead of the conventional register file would save a register address in the instruction word, for the destination is always the shift register. Section 2 will illustrate that in RISC programs a lot of calculated values have a short life time. In section 3 a hardware concept is discussed and section 4 shows example instruction formats for code compaction.

Beyond the increased program code density, there are also other advantages. To put all results into a shift register and address them by the distance between producing and using is a substitute for the forwarding concept. It reduces the number of pipeline stages and it offers a very simple concept to handle speculative execution of instructions. These benefits will be discussed in section 5.

2 Temporary values

RISC processors achieve high performance by fast execution of simple machine instructions with short execution latencies. However, a lot of temporary values arise when a complex expression is executed as a long sequence of simple machine operations. Figure 1 shows an example for the execution of a high-level-language expression and the corresponding directed acyclic graph (DAG). The expression is calculated by four RISC instructions.

In the example, the leaves of the DAG are program variables. An interior node represents an operator and its children represent its operands. Only the result of the last multiplication, represented by t1 is of interest later in the program. The intermediate results t2, t3, t4 are temporary values with a short life time.




```
(1)   ADD R1, A, B      // R1=A+B
(2)   MUL R2, R1, C     // R2=R1*C
(3)   SUB R3, R2, D     // R3=R2+D
(4)   MUL R4, R1, R3    // R4=R1*R3
```

Figure 1: Directed acyclic graph and RISC program to calculate (A + B) * ((A + B) * C - D)

The next section will show that it is not necessary to assign register addresses to temporary results. This in turn allows to code a great deal of the instructions in smaller code words.

3 The proposed architecture

The idea is to use a shift register to collect the results. Each time a new result is produced, it will be put onto the top of the shift register file, the older data move down. The shift register must have random access for reading operands. The access address is equal to the number of shifts performed since it was produced. The compiler knows how often a value will be shifted before it is needed as source operand. Considering a sufficient length, all temporaries can thus be passed through the shift register without storing them into general purpose registers (see Figure 2).

However, in some cases it is not sufficient to store values only in the shift register. Some values have long life times within a procedure or a program (e.g. stack pointer, return addresses). They would get lost by shifting them out. There are also cases in which the compiler can not predict the position of a certain value within the shift register. That’s why we still need in addition to the shift register a conventional register file.

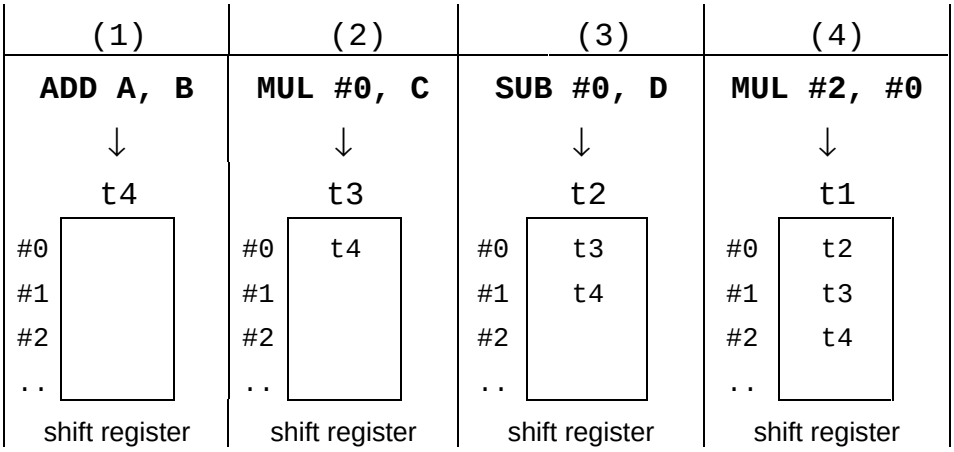


Figure 2: Passing temporaries via shift register

Usually, in RISC processors a bypass buffer for ALU results is implemented for result forwarding (Figure 3). We propose to replace the bypass buffer by a shift register. Figure 4 shows the modified operation unit. Each operation will now be performed in two (pipelined) steps. In the first step, the two operands are read either from shift register or general purpose register file. In the second step, the operation is performed. To obtain a regular structure, also the store and load operations should be performed in only two steps. This in turn does not allow the combination of address calculation and memory access.

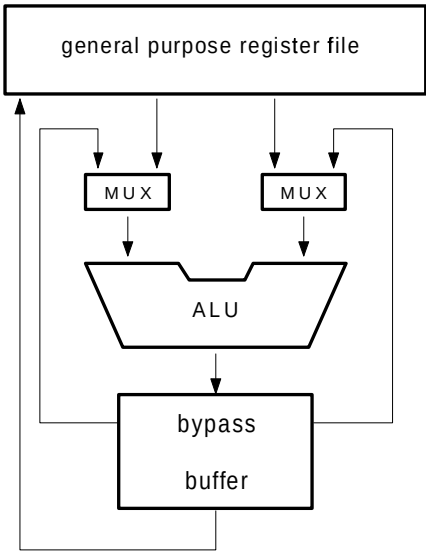


Figure 3: Conventional ALU with bypass buffer

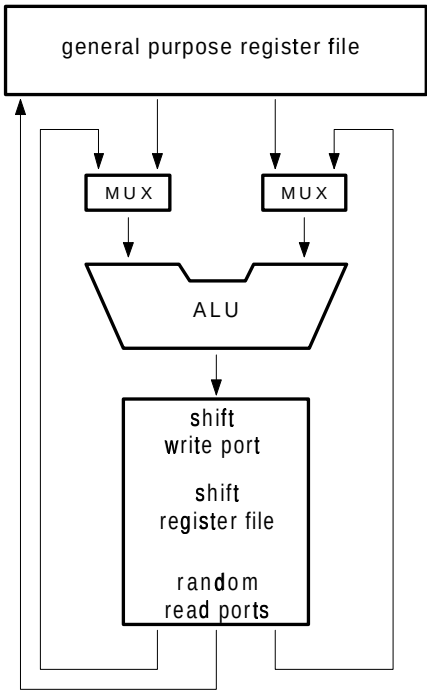


Figure 4: The modified operation unit

As Figure 5 shows, load and store operations do not perform address calculations. The address has to be calculated in a preceding instruction. The copy instruction is a peculiarity of the concept. Results that have to be available over a longer period must be moved to the register file before they are moved out of the shift register.

For the shift register, a part of the general purpose register file with some additional addressing logic could be used instead of a separate physical shift register.

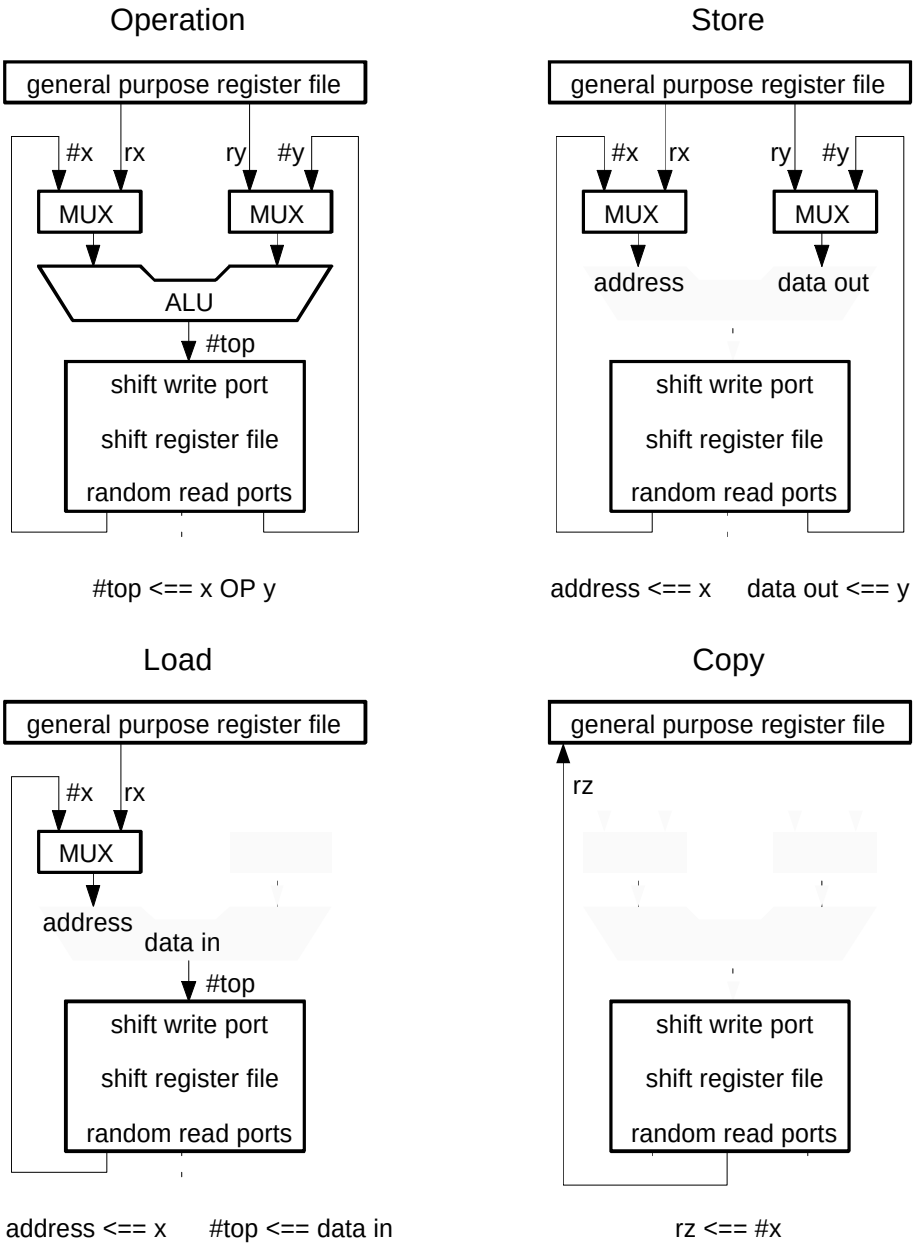


Figure 5: Data path for ALU operation, load, store, and copy

The machine with the proposed operation unit will have less but simpler instructions than a state of the art RISC processor, i.e. it is more RISC-like.

A critic of our concept may say that the simplification of the instructions will increase the number of instructions of a program and so its run time. Most instruction overhead arises from the splitting of conventional RISC load and store instructions into a separate address calculation and an instruction for memory access. A processor with only one unit for operation and data moving will be slower. For a processor with multiple units as shown in Figure 6 the situation is different. The address calculation can be performed in one of the ALUs, and in the next time slice data are moved from or to the memory. Simultaneously the next address calculation can be performed etc. But if no address calculation is needed, the ALU can also perform another operation. So, our shift register processor may even be faster then a conventional RISC processor, if it has the same number of arithmetic logic units.

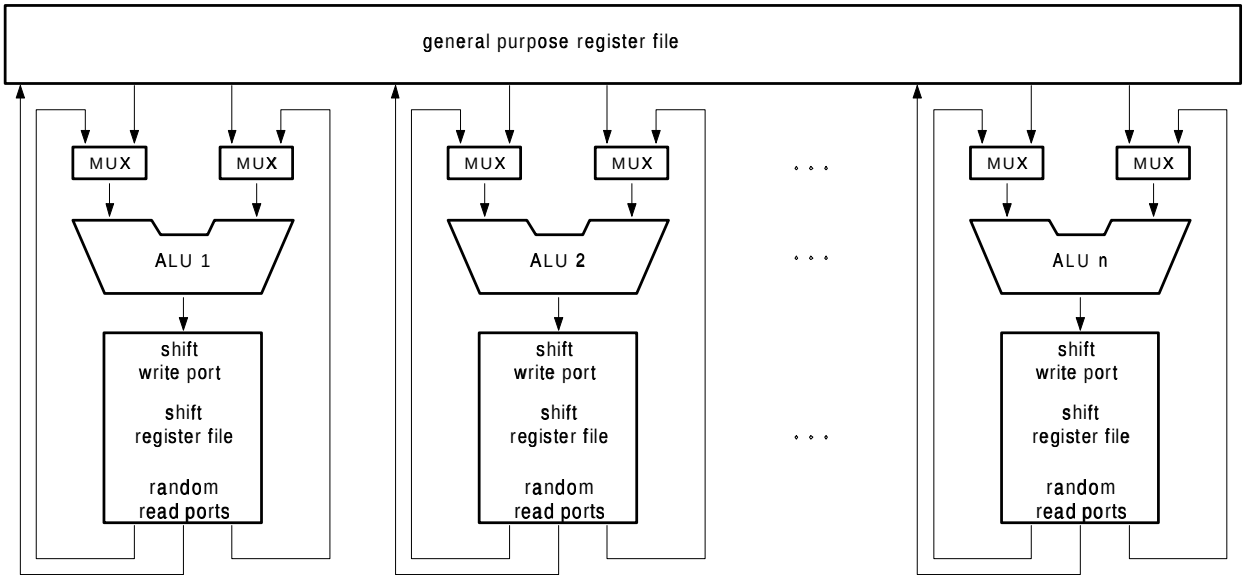
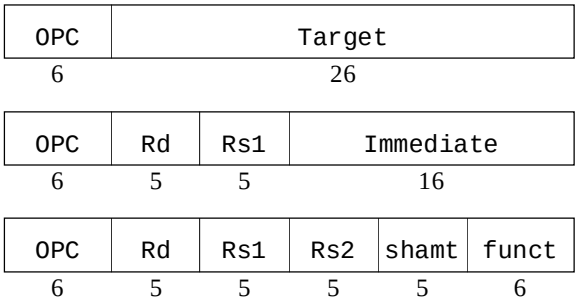


Figure 6: Data path of a VLIW- or a superscalar processor using shift registers for the results

4 Compact Instruction Encoding

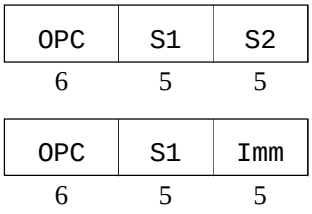
Our research project focuses on methods to increase the program code density. RISC processors use a small number of different 32 bit instruction formats. As an example, Figure 7 shows the formats of MIPS processors.



(1) OPC: opcode
funct: extended opcode (function)
Target: jump target address
Rd: destination register address
(2) Rs1, Rs2: source register addresses
Immediate: immediate value (16 bits)
shamt: shift amount
(3)

Figure 7: Instruction formats of the MIPS processor [3]

The proposed shift register technique with implicitly directed results gives us the ability to introduce smaller formats for frequently used instructions.



(4) S1, S2: general purpose register or shift register addresses

Imm: short immediate value (5 bits)

(5)

Figure 8: Compact instruction formats without destination register

Due to the limited opcode field size, it is important to find a useful balance between standard and compact encoding in the instruction set design. Experiments and instruction usage measurements have shown that even in RISC processors the greatest deal of the instruction stream is caused by a small subset of the instruction set (for one example see appendix). We currently evaluate this approach in experiments with our configurable processor simulator [4] and a special code generator [5] developed as back end for the SUIF C compiler [6]. Our experiments are still in progress but it turns out that compacting instructions in the described way would significantly increase the program code density and reduce the instruction stream.

5 Some other advantages

With the increasing number of transistors, the processor design must become more regular. Our idea with the shift register would allow to design more regular processors. The following functions could be simplified:

- Forwarding: In a plain pipeline implementation of a RISC processor results are not available to the next instruction. However, as shown in Figure 1, temporary results are often needed in the following instruction. To avoid NOPs, RISC processors have a special forwarding logic. It comprises comparators to detect if one of the source registers is the destination of the preceding operation, and a special bypass logic. In our concept, an operation can directly use the result of its predecessor without special logic simply by addressing shift register zero.
- Fewer pipeline stages: The proposed operation unit needs only two pipeline stages: Select operands and execute. The third, writing the result back into the register file, is saved. A shorter pipeline makes the handling of interrupts and traps easier.
- A simple concept for the speculative execution of instructions: Speculative execution is used in processors with multiple processing units for a better exploitation of fine grain parallelism. In our concept the speculative result will be put in the shift register as any other result. If the program branches in the predicted way, the value can be addressed by the number of instructions between calculation and usage. If the point of usage will not be reached, the data, which are idle in this case, will drop out of the shift register after a certain time. Summing up, speculative execution does not need any additional hardware.

6 Summary

Substituting a shift register for a part of the general purpose register file of a RISC processor allows to encode instructions without explicit destination register address. This can be used to

reduce the instruction stream. On the other hand, the shift register concept offers interesting simplifications of the processor architecture.

Appendix

The following table should illustrate that a RISC program in general uses mainly a small subset of the processors instruction set. The table shows the frequency of executed instructions for the programs *gzip* and *ghostscript* on a MIPS processor. More than 90% or 75% respectively of the instruction stream consist of only 18 different instructions. These instructions are candidates for encoding with compact instruction formats.

gzip				ghostscript			
instr.	count	%	sum	instr.	count	%	sum
addu	6980621	16.21%	16.2%	lw	4606215	14.09%	14.1%
lnop	3950888	9.18%	25.4%	addu	3352641	10.26%	24.3%
addiu	3810672	8.85%	34.2%	sw	3278786	10.03%	34.4%
lbu	3097410	7.19%	41.4%	addiu	2747057	8.40%	42.8%
andi	2856325	6.63%	48.1%	bnop	1619402	4.95%	47.7%
bnez	2460465	5.71%	53.8%	lnop	1491952	4.56%	52.3%
sll	2419990	5.62%	59.4%	beqz	1423013	4.35%	56.7%
bne	2393737	5.56%	65.0%	bnez	1028638	3.15%	59.8%
lhu	2206812	5.13%	70.1%	bne	863465	2.64%	62.4%
lw	2201217	5.11%	75.2%	li	842103	2.58%	65.0%
beqz	2164596	5.03%	80.2%	sll	807479	2.47%	67.5%
sltu	2002593	4.65%	84.9%	lbu	702178	2.15%	69.6%
sw	1290948	3.00%	87.9%	sltu	606081	1.85%	71.5%
sh	664991	1.54%	89.4%	b	583343	1.78%	73.3%
bnop	523100	1.21%	90.6%	subu	557459	1.71%	75.0%
	...				...		

References

[1] Hennessy, J. L.; Patterson, D. A.: Computer Architecture - A Quantitative Approach Morgan Kaufmann Publishers, Inc., San Mateo, 1990.

[2] Kemnitz, G.; Markwardt, G.; Schulz, P., Sawitzki, S.; Spallek, R. G.: Design of Memory Space Optimized and Instruction Stream Reduced RISC Processors, Tech. Rep. FI/95/15, Fakultät Informatik,TU Dresden, November 1995.

[3] MIPS Computer Systems, Inc.: MIPS R4000 User's Manual, 1991.

[4] Markwardt, G.; Kemnitz, G.; Schulz, P; S.; Spallek, R. G.: PROSIM: A Tool for Instruction Level Simulation of RISC Processors, Tech. Rep. FI/95/16, Fakultät Informatik,TU Dresden, November 1995.

[5] Schulz, P., Design and Implementation of an Object Code Generator for SUIF Tech. Rep. FI/95/17, Fakultät Informatik,TU Dresden, November 1995.

[6] Stanford SUIF Compiler Group, SUIF: A parallelizing & optimizing research compiler, Tech. Rep. CSL-TR-94-620, Computer Systems Lab, Stanford University, May 1994.