Can the Advantages of RISC be Utilized in Real Time Systems?

A. Steininger and H. Schweinzer
Institut für Elektrische Messtechnik
Technical University of Vienna
Vienna, A-1040 Austria

Abstract

Within the last few years a lot of improvements have been made for processor architectures. Average and peak performance are increasing continuously. RISC, caching, pipelining and similar features have become indispensable to many systems. In critical real time systems, however, peak and average values are of minor interest. Because strict deadlines have to be met, only those values may be considered, that can be guaranteed even under adverse conditions. Features like reduced instruction set, caching, pipelining etc. are based upon statistical considerations and do not necessarily improve worst case performance. The question arises, whether new powerful processor architectures - especially the RISC architecture - are suitable for real time systems and if so, under which conditions. In this paper some of the most important hardware features are analyzed to find an answer to this question.

1 Introduction

The compatibility of RISC and real time is of central interest for every real time design that employs modern processors, especially RISCs. We are currently designing a high end single-board computer for real time applications with RISC processors and self-checking capability[1]. A survey of our discussion on the following items will be given subsequently:

a) Is the architecture of a RISC processor suitable for handling real time requirements and which type of processor should be chosen ?
b) Which environment (board architecture) must be chosen ?
c) How can the real time behaviour be modelled ?
d) Which special requirements arise for the compiler/ postprocessor ?

1) This work is supported by the Austrian Science Foundation under contract P7582-TEC

The answers are derived using the principles of the classical real time design by taking the following steps:

- Point out the specific requirements of a real time system
- Find out the characteristic features of a RISC architecture
- Analyze their influence on the real time behaviour.
- Draw the conclusions

2 Special Requirements in a Real Time System

In a non-real time system results are characterized by their values only. In a real time system, however, results are valid only if their values are correct *and* if they are computed in time [2].

Compared to a non-real time system the following additional requirements arise:

- An upper limit for the duration of each process must exist and be known. A real time behaviour that can be easily modelled is desirable.
- Peak or mean values of the performance are not relevant. Only that value of performance is of interest, that can be guaranteed even under adverse conditions.
- Interrupt latency must be as short as possible in order to allow efficient task switching and short response time to external stimuli.

The target of the RISC architecture is not a priori a real time system. Many features are based on statistical assumptions and make - with regard to the requirements stated above - the applicability for real time questionable. For this reason the main features of the RISC architecture will be pointed out in the following section.

0-8186-2212-1/91/0000/0030$01.00 © 1991 IEEE

Table 1
Comparison of the features of RISC and CISC

Type of Processor	RISC				CISC
	R3000 (MIPS)	CY7C600 (SPARC)	i860	MC88000	80386
Number of instr.	74	64	79	79	152
Parallel execution units	(Coproc.)	(Coproc.)	on chip	on chip	(Coproc.)
Pipelining	yes	yes	yes	yes	yes
General Purpose Registers	128Byte	544Byte	128Byte	128Byte	32Byte
Triadic register addressing	yes	yes	yes	yes	no
Caching directly supported	Data & Instr	Data & Instr	Data & Instr	Data & Instr	no
Harvard architecture	caches only	no	caches only	full possible	no
Block fetch directly supported	yes	yes	yes	yes	no

3 Characterization of RISC

There exists no general definition of RISC-specific features and no clear distinction between RISC and CISC. Only the combination of all features characterizes an architecture. The peculiarity of a RISC-architecture is made up by the sum of several features, which can separately be found in CISC Processors, too:

- reduced instruction set (hardwired instructions)
- parallelism
- pipeline-structure of function units
- great number of internal registers
- triadic register addressing (non destructive register operation)
- caching
- Harvard Architecture
- block fetch

4 Influence of the RISC-Features on the Real Time Behaviour

The function of a processor mainly consists of two basic tasks: Instructions/Data must be *accessed* and must be *processed.*

The timing of an access is typically determined by bus and memory bandwidth, the processing speed depends on the structure of the CPU. Since these two tasks are usually performed in parallel by independent function units, their durations can be investigated separately. The global real time behaviour results from their combination, as is shown in figure 1.

In case of the simplifying assumption that no internal queuing (buffering) of data/instructions is performed by the CPU, the duration of each phase is always determined by the slower function. (Internal queuing eliminates stalls that happen alternatively for accessing and processing. However, no improvement can be guaranteed under worst case conditions.)

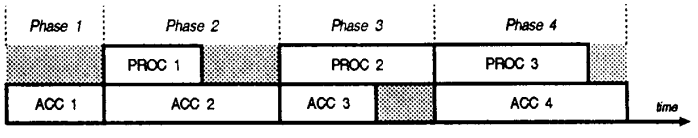

Fig.1 Principle of combination of access- and process-time

4.1 Influence of the RISC-Features on the Processing Speed

A general indicator for processing speed is the time a CPU needs to finish a predefined ("benchmark"-)computation. The following equation analyzes the determinants of computation time (for better understanding the fraction bar should be read as "per" or "for a"):

$$\frac{time}{computation} = \frac{instructions}{computation} * \frac{clock\text{-}cycles}{instruction} * \frac{time}{clock\text{-}cycle}$$

Example: RISC-Processor, 25 MHZ, the processor needs 250 instructions to perform the given computation, an instruction is finished every 1.2 clock-cycles (avg.):

*time per computation is: 250*1.2*1/25MHz=12ms*

One main idea of the RISC-concept is based on the fact, that within most computations a high percentage of very simple instructions is used. So, if only those simple instructions are provided and the rarely used complex ones are replaced with a sequence of simple ones by the compiler, term 1 on the right side of the above equation will not increase significantly.

On the other hand the reduction of the instruction set allows the omission of microcoding, which results in shorter instruction cycles improving term 2. Moreover, the internal architecture of the processor can be kept very simple, allowing high clock-rates (term 3). The resulting increase of processing speed is usable in real time systems without any restrictions and is one main reason for the attraction of the RISC architecture for real time applications.

Parallelism makes it possible to perform more than one computation at a time and thus multiplies the possible throughput (further improvement of term 2). As there is no guarantee that each function unit can always be kept busy, the model for the timing must include the observation of several conditions:

(1) Function units can only work in parallel, if there are no data dependencies between them (e.g. one unit needs the result of the other as input). This condition has to be taken into consideration when the timing behaviour is investigated.

(2) If a function unit that needs multiple clock cycles for the completion of one operation only accepts new data when the previous computation is finished, its availability must be considered carefully. By the use of an appropriate pipelining-strategy, however, it can be guaranteed that new data are accepted with each cycle [3], which greatly simplifies timing-analysis. This is managed by dividing the task of the function unit into sub-tasks that can all be performed in parallel and be finished within one clock-cycle.

Example: The MC88000 has a 3-stage data unit [5]. In stage 1 the address of the data to be transferred (e.g.read) is calculated, in stage 2 the resulting address is put on the bus, and at the same time the address for the next access is calculated by stage 1. In stage three data is read from the bus (see fig. 2).

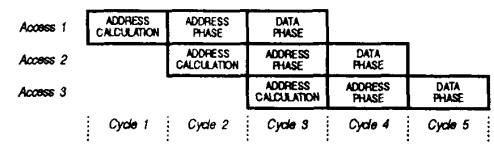

Fig. 2 Pipelined data access

The mechanisms described above enable the execution of one instruction per clock cycle and - by the use of parallelism - even more. With a high clock rate (e.g.25MHz) and such a powerful execution unit processing speed becomes - as described in fig. 1 - negligible in comparison to an access time of about 8 clock cycles to the main memory. Therefore special attention has to be paid to the access behaviour.

4.2 Influence of the RISC-Features on the Access-Behaviour

The optimization of access time is performed on three levels:

The first and simplest level is to minimize the number of external data access by providing a big number of registers and even to minimize the number of register loads by making all operations non destructive for the source registers (triadic register addressing).

The second way to improve access time is the use of caches to avoid unnecessary wait states resulting from an access to the main memory. In a real time system a statistical information about the cache hitrate is not sufficient. The time analysis tool must determine a minimum value that can be guaranteed under all circumstances. By this the complexity of the timing model grows significantly. The

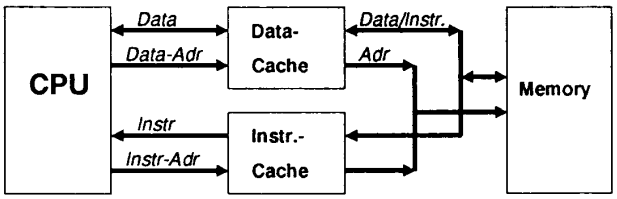

Fig.4 Single stage Harvard Architecture

alternative is omitting the caches, which means loosing a major part of the advantages of the architecture.

Example: An 88000 with 25MHz can achieve a performance of 21VAX-MIPS[2] by the use of caching. Without caching each instruction must be directly fetched from the main memory (e.g. 70ns-DRAM), which results in an upper performance-limit of 1/(70ns)=14 MIPS given by the possible speed of data transfer only.

The third way of optimizing access time is the use of block fetch in combination with caching. An adverse case can easily be imagined, where only one word of each fetched block is used. Nevertheless a closer investigation shows, that, due to the locality of data, in most cases the increase of the cache hitrate will be a compensation for the additional access delay.

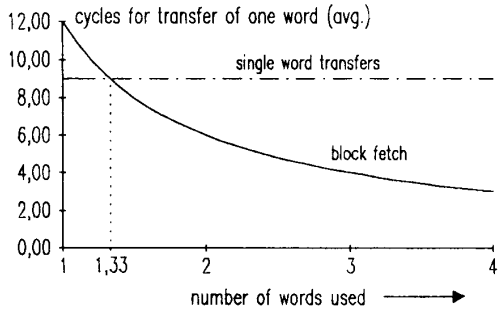

Fig.3 Access time with block fetch

Example: For an access to a single word in main memory the 88000 needs 9 clock-cycles (translation table lookup, arbitration-cycle, actual access with 2 wait-states, cache update; for the sake of simplicity translation table miss, copyback and bus contention are neglected). In block fetch mode with every cache miss 4 contiguous words can be fetched within 12 clock-cycles using the DRAM nibble mode [6]. As is shown in fig. 3, block fetch is advantageous, if the probability that any of the three additionally fetched words is used before the cache line is replaced is more than 33%.

The corresponding timing model has to take into account two facts:

- the cache hitrate changes with the use of block fetch
- the access time is increased by block fetch

Another aspect of optimizing access behaviour is to avoid bus contention and arbitration phases. For single master systems this can be managed by using Harvard Architecture. In combination with caching there are two possible ways of implementation:

- Full Harvard Architecture: Completely separate busses, separate caches and separate memories for data and instructions guarantee that there is no bus contention at any time.
- Single stage Harvard Architecture: While caches are separate, there is only one memory for data and instructions (fig.4). The possibility of simultaneous cache misses for data and instructions makes arbitration necessary.

From the view of real time systems the full Harvard Architecture is most desirable because it is easier to be handled. The only advantage of the single stage solution is reduced cost. Nevertheless, only one of all available high end RISC processors supports the full Harvard Architecture (see table 1). This type, the Motorola MC88000, has been taken as an example for a comparison since it allows both approaches.

2) Value given by the manufacturer

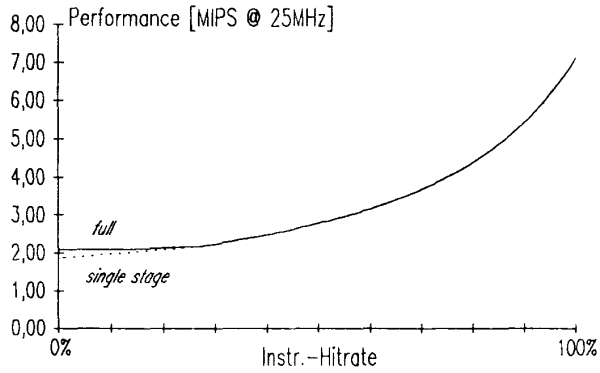

Fig.5 Comparison of worst case performance

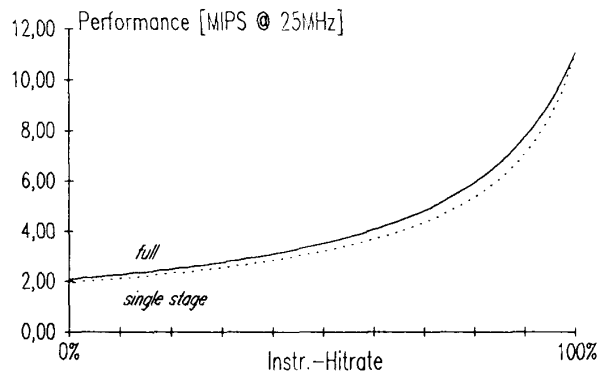

Fig.6 Comparison of average performance

4.3 Comparison of Full and Single Stage Harvard Architecture for the MC88000

The complete timing behaviour has been modelled on the base of a given minimum for the hit rates of data and instruction caches as demanded above. Two cases are to be distinguished:

a) The distribution of data misses in respect to instruction misses is as adverse as possible (worst case). This case corresponds to a timing model unable to predict the real distribution.

b) The distribution of data misses and instruction misses is arbitrary (mean values). The corresponding model is able to predict the reality ideally.

Figures 5 and 6 illustrate the worst case performance and the average performance using the following parameters:

- data hitrate = 40%,
- data access rate (= percentage of data transfer instructions refered to total number of instructions) = 20% Rd+15% Wr,
- block fetch,
- write through,
- 25MHz clock

The model assumes optimum processing speed (1 instr/ cycle) and only considers access behaviour. The influence of decreased processing speed can be computed separately. For a simple approximation of the worst case behaviour the

effects of processing speed and access behaviour can be superposed, as is described at the beginning of section 4.

Figures 5 and 6 show, that the full Harvard Architecture does not improve the performance substantially, even though - as is indicated by the low performance values attained - rather adverse conditions for data hitrate and transfer rate have been assumed [4]. This especially concerns the worst case performance in combination with instruction cache hitrates above 20%. The reason for this may be the fact, that for high cache hitrates the corresponding bus traffic at the main memory becomes low and arbitrations are rarely performed.

4.4 Influence on the Interrupt Latency

Most of the instructions are executed within one clock cycle, so the interrupt latency is very short. For real time applications two major problems have to be considered:

- A great number of registers must be saved before a task switch. This can be avoided by restrictions in the usage of registers within interrupt service routines.
- The interrupt service routine is very likely not in the instruction cache. So the access to main memory must be waited for. It would be advantageous for some applications, if the cache controller supported the locking in of information (e.g. a very important interrupt service routine) in the cache.

5 Results

The questions in section 1 can be answered now as follows:

a) The RISC Architecture is able to handle real time requirements. As there is no need of using full Harvard Architecture, all processors are basically suitable. For the research project the MC88000 was chosen because of its superior support of fault tolerance.

b) Single stage Harvard Architecture is sufficient, but makes timing analysis more complicated.

c) The classical model for timing analysis must be extended by:

- the consideration of data dependencies between the function units
- a determination of a lower limit for the cache hitrate (including the effect of block fetch).
- the consideration of the distribution of data misses in respect to instruction misses

d) the compiler or a postprocessor must

- support the time analysis tool with appropriate information and
- consider additional parameters for optimization (e.g. optimal usage of caches, registers [1], pipelines and parallel function units).
  This last demand is usually met by RISC compilers, whereas the informations demanded in the first item must be delivered by a postprocessor.

References

[1] Chi-Hung Chi, H. Dietz. Unified Management of Registers and Cache Using Liveness and Cache Bypass. *Sigplan Conference on Programming Language Design and Implementation*, acm Press, Sigplan Notices Vol.24, No.7, pp 344-355, July 1989.

[2] H. Eichele. *Multiprozessorsysteme*. Teubner, Stuttgart 1990. (German Language)

[3] J. L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc. 1990

[4] Mark D. Hill and Alan Jay Smith. Evaluating Associativity in CPU Caches. *IEEE Transactions on Computers*, Vol. 38, No. 12, Dec. 1989.

[5] *MC88100 User's Manual*. Motorola Inc., Phoenix, Arizona, Nov 1988.

[6] *MC88200 User's Manual*. Motorola Inc., Phoenix, Arizona, Nov 1988.