Invited Paper

32-bit Microprocessors Based on the TRON Architecture Specification

KEN SAKAMURA* and TATSUYA ENOMOTO**

1. Introduction

Various general-purpose 32-bit microprocessors are being designed and manufactured on the basis of the TRON (the realtime operating system nucleus) specification developed in a TRON subproject. These chips, which will utilize advanced VLSI technologies of the 1990's to the limit, are to be optimized for operating systems based on TRON specifications developed in other TRON subprojects, and will act as a powerful engine for the entire TRON project.

The chip architecture of the TRON specification has been designed as part of the total architecture, in parallel with the design of the TRON specifications for operating systems. Anyone is and will be allowed to develop new microprocessors freely and independently on the basis of suggested external specifications. This open architecture constitutes an epoch-making advance over existing microprocessor architectures.

Although there are a number of general-purpose 32-bit microprocessors already on the market, the chip architecture of the TRON specification has been newly designed from scratch, and several semiconductor manufacturers are now developing microprocessors with the new architecture. It should be noted that the development of these chips has been triggered by a recognition that a new architecture will be required to match the fields in which microprocessors will be applied in the 1990's; one that will allow the most advanced VLSI technology to be fully implemented, rather than one bound by the need for compatibility with existing architectures.

The first half of this paper gives a brief overview of the TRON project, the design philosophy of the TRON specification for the VLSI CPU, and the chip architecture of the TRON specification, in Sections 2, 3, and 4, respectively. The second half describes the technologies involved in its realization and examples of products. In Section 5, VLSI implementation based on the TRON architecture specification is described in the case of the $G_{MICRO}/100$. Since related development tools and operating systems are necessary for a microprocessor with a new architecture to have practical applications, these will be mentioned in addition to related VLSIs. Results of the evaluation of the $G_{MICRO}/100$ and related development tools are also described in Section 5. An example of an implementation of a realtime operating system running on the $G_{MICRO}/100$ and based on the ITRON specification is described in Section 6 in order to show that the TRON architecture is designed as a total architecture.

*Department of Information Science, Faculty of Science, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113 Japan.
**Advanced Microprocessor Development Dept, LSI R&D Lab., Mitsubishi Electric Corporation, 4-1 Mizuhara, Itami 664 Japan.

2. Overview of the TRON Project

The TRON project aims at establishing a new computer architecture that covers microprocessors, operating systems (OSs), man-machine interface (MMI) and networking [1].

The ultimate goal of the TRON project is to realize a Highly Functional Distributed System (HFDS) in which a tremendous number of computers and computer-controlled objects are connected and work cooperatively to perform various services. An HFDS is a heterogeneous loosely-coupled computer network, whose nodes include both computers and computer-controlled objects. Since a single architecture cannot meet various demands arising from the full range of applications included in the HFDS, the TRON project has several subprojects whose goal is to design a series of different computer architectures one by one.

Industrial-TRON (ITRON) is a realtime, multitask OS architecture for embedded computer systems. Its major objective is to minimize the task dispatching time. The subproject has involved the design of a set of micro-ITRON specifications for an OS that is adapted for each microprocessor used in the system, rather than standardized according to the ITRON specification, in order to get the best realtime response in a small embedded computer system. The OS based on the micro-ITRON specification can be executed even on 8-bit and 16-bit single-chip microcomputers with limited-capacity memories, because its object size can be made as compact as possible.

Business-TRON (BTRON) is a specification of an OS and MMI for high-performance workstations, which

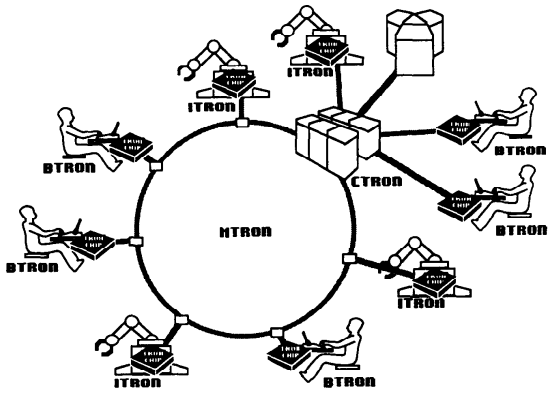


Fig. 1 Overview of a system based on the TRON architecture.

will be used as machines for communicating with other people and machines. The BTRON subproject aims to establish a guideline for the design of MMI. The MMI thus created will be made consistent, so that users can move from one system to another without running into difficulty with incompatible MMI's. The BTRON subproject is also designing a set of micro-BTRON specifications for very small specialized computers to be used as "electronic stationery goods." This will meet the rapidly growing demands for such personal-use computers.

Central-TRON (CTRON) is a specification of an OS for servers and gateways on the HFDS. Although the CTRON-based OS allows multiple users, it has no direct MMI; this is provided by the BTRON-based machine.

Macro-TRON (MTRON) is the key to the realization of the HFDS. It is a network architecture that will control distributed systems based on ITRON, BTRON, and CTRON architectures (Fig. 1).

A TRON VLSI CPU architecture has also been designed to support these OS's and applications efficiently.

The TRON project itself was born in 1984, and is still under way, with the final objective of creating real implementations of HFDS's. Some prototype systems based on ITRON, BTRON, and CTRON architectures have already been built. The current status of 32-bit microprocessors based on the TRON specification will be described later in this paper.

3. Design Philosophy of the TRON Specification for VLSI CPUs [2, 3]

3.1 Design Considerations of the TRON Specification

The first motivation for designing a new chip architecture for the TRON specification is the assessment that microprocessors of the 1990's will require a new architecture that will take account of applications and VLSI technologies to be realized during the decade. Many of the currently marketed 32-bit microprocessors have architectures largely derived from those of the preceding 8-bit and 16-bit microprocessors. Compatibility with previous architectures is important from the point of view of allowing the use of existing software, but architectures become obsolete with time and will no longer be suitable for applications in a new era.

In the TRON project, two operating systems are considered as being the most appropriate to the fields in which microprocessors will be applied in the 1990's: one is based on the ITRON specification for real-time and embedded systems, and the other is based on the BTRON specification for personal-use computers. The chip architecture has been designed in such a way as to maximize the efficiency of these operating systems. That is, we are attempting to make it possible to construct high-performance systems by adopting a "total-architecture" approach.

The second aim of the new architecture is to realize an architecture that makes the best use of the advantages of the von Neumann-type computer. Because of its general-purpose capability and extreme technical refinement, the von Neumann-type computer will undoubtedly be the primary computer used in the 1990's.

The von Neumann computer must satisfy two important features: direct and free access to a large address space and excellent basic performance. To fully exploit these qualities, the TRON specification aims to achieve as large a linear address space as possible. Thus, although the specification originally called for microprocessors with 32-bit addressing, the architecture is being designed to allow for expandability up to 64-bit processors, with full upward compatibility. Consideration is also given to the high-speed execution of frequently-used basic instructions, in addition to the orthogonality and increased functionality of instructions.

The third aim of the new architecture is to provide a standard instruction set as an open architecture. The interface is to be the same, but there will be freedom in the actual method of implementation. While maintaining compatibility, manufacturers are free to compete with each other; this will allow the technology to progress. Use of a standard instruction set will increase the effectiveness of education for the users of the microprocessors and the preparation of development support systems.

For continued progress and widespread use of the microprocessors, it is essential to have an open architecture that can be used freely by anyone. The TRON specification is intended to respond to such needs.

3.2 RISC and CISC

Reduced Instruction Set Computer (RISC) architecture has been gaining popularity for VLSI processors and is being increasingly used in engineering workstations.

The objective of RISC is to make the instruction

cycle time as fast as possible by reducing the functions supported by the hardware. The features of RISC chips are a small number of machine instructions, fixed-length instructions, one-cycle execution, limited addressing modes, load-store architecture, wired logic control instead of microprogram control, and on-chip large-capacity registers. Since RISC processors use a reduced instruction set, the object size and instruction traffic are greater than in Complex Instruction Set Computer (CISC) processors. For this reason, they require a high memory bandwidth, large capacity cache memories, good optimizing compilers, and so on, in order to take full advantage of their features.

RISC chips will probably continue to play an important role in engineering workstations and minicomputers, but they are specialized for high-level languages and limited applications. If their on-chip large-capacity registers can be utilized appropriately, data traffic can be reduced and these processors are then well suited to applications such as scientific and engineering computations, where registers are fully used. However, it takes longer for these chips to execute complicated and frequent handling of external data. Furthermore, they do not provide high-level instructions that are useful for high-speed execution of critical operations in real-time applications, in spite of their high-speed execution of basic instructions. They contain some barriers to the design of low-cost systems. It is thus difficult to imagine that they will find applications in more general areas, such as popular personal-use computers with good price-performance ratio, or embedded systems that put emphasis on real-time performance.

In contrast, equipping CISC chips with appropriate registers and cache memories allows them to function effectively and gives them the ability to deal with complicated applications. Furthermore, not only RISC but also CISC chips tend to adopt techniques such as one-cycle execution and wired logic control to achieve high-speed execution of basic instructions, and the performance gap between them is getting smaller.

The application of microprocessors will be very wide. In the TRON project, attention is focused on two application fields, namely, high-performance personal-use computers and real-time embedded systems. It is ideal to use just one family of general-purpose microprocessors to build personal-use computers, computer-controlled objects, and other future computer systems. This is because the manufacturers can cut down the development of VLSI microprocessors by sharing the basic design among the microprocessor family, and software development systems including compilers can be shared. The users will also find it convenient to deal with only one architecture. The TRON specification has been designed with this generality in mind.

The TRON specification, while providing high-level instructions for compilers and operating systems, has emphasized reducing the object length of frequently used basic instructions. The result is that, in principle, instructions that manage simple processes can be executed at the same speed as in RISC chips. That is, chips based on the TRON specification incorporate the best of CISC and RISC, and will find use in a wide range of applications from high-end workstations to small-scale embedded systems.

4. The Architecture of the TRON Instruction Set [2, 3]

4.1 Basic Instruction Set Architecture

4.1.1 Instruction Format

In the TRON specification, the instruction format is determined in such a way as to achieve both a reduction in instruction length and high orthogonality of the instruction set. Shortening the instruction length and making the object size as compact as possible allows the structuring of systems with limited memory sizes and high-speed program execution. The orthogonality of the instruction set here means that there are few constraints on the combinations of elements that make up an individual instruction (for example, instruction function, addressing mode, and operand size); this makes it easy to develop high-quality compilers and contributes to improved software productivity.

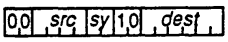
Generally, these two objectives must be traded off against each other; it is difficult to achieve both at the same time. However, in the TRON specification, the instruction length is reduced while the orthogonality is preserved by providing two classes of instruction formats: a general format with good orthogonality and high-level functions, and a short format with limited-level functions but short instructions. The short format in the TRON specification is distinguished by the fact that the functions of the general format instruction and the corresponding short format instruction, if it exists, are the same, including the changes of flags and exceptions. Only the addressing modes, the range of immediate operands (literals), and the operand size are subsets of those of the general format. Moreover, the short format has been introduced for many of the instructions in a unified way. Figure 2 shows the general and short formats of the MOV instruction as an example.

It should be noted that the length of each instruction is variable in multiples of two bytes. Excluding the addressing extension portion, the length of the short format is two bytes, while that of the general format is four bytes.

4.1.2 Register Set

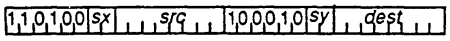
Figure 3 shows the general register set. In the 32-bit specification, there are sixteen 32-bit general registers (R0 to R15). The 64-bit version will have sixteen 64-bit general registers. A general register can be used to hold either data or base address, and as an index register. The stack pointer (SP) and frame pointer (FP) are in-

MOV:S (an example of a short format)



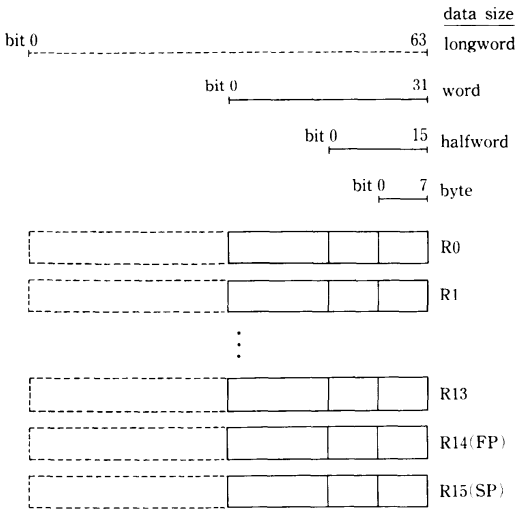
function: transfer from register to memory
src: register of source operand (4-bit code)
dest: addressing mode of destination operand (6-bit code)
sy: size of destination operand (2-bit code)

MOV:G (an example of a general format)



function: transfer from memory to memory
src: addressing mode of source operand (8-bit code)
dest: addressing mode of destination operand (8-bit code)
sx: size of source operand (2-bit code)
sy: size of destination operand (2-bit code)

Fig. 2 Examples of instruction formats of the MOV instruction.



The chip has a memory protection scheme using four-level rings. SP is prepared for each ring (0 to 3) and for interrupt processing, and switched automatically.

Fig. 3 General register set of a microprocessor based on the TRON specification.

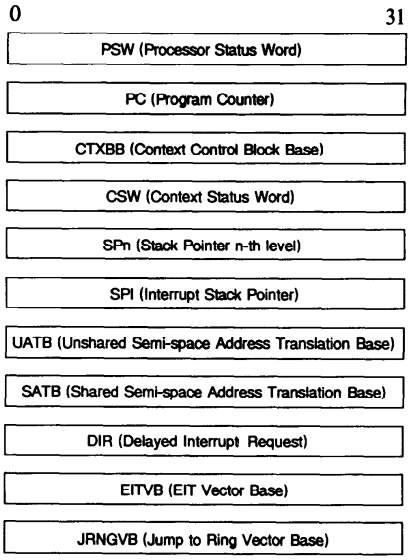


Fig. 4 Examples of control registers in a microprocessor based on the TRON specification.

Table 1. Instruction set of TRON architecture.

Transfer Instructions
Compare Instructions
Arithmetic Instructions
Logic Instructions
Shift Instructions
Bit Manipulation Instructions
Fixed-length Bit Field Instructions
Variable-length Bit Field Instructions
Decimal Arithmetic Instructions
String Instructions
Queue Manipulation Instructions
Jump Instructions
Multiprocessor Instructions
Control Space and Physical Space Instructions
OS-related Instructions
MMU-related Instructions

cluded in the general registers, R15 being assigned to the SP and R14 to the FP. The program counter (PC) is not included in the general registers but in the control registers. These general registers can support byte (B), half-word (H) and word (W) data sizes in the 32-bit version.

Figure 4 shows examples of the control registers. The program counter (PC) and the processor status word (PSW) are included as basic control registers. The PSW is a 4-byte register that is used for setting and indicating the current processor status and processing mode.

4.1.3 Data Types

The so-called "big-endian" is employed. That is, MSB is assigned as the lowest number (address) of both bit number and byte address, as shown in Fig. 2. Supported data types include integers, floating points, decimals, bits, bit fields, strings, and queues.

4.1.4 Instruction Set

With two-operand instructions as a base, the instruction set ranges from basic instructions to high-level instructions designed for the operating systems of the TRON specifications. Classes of supported functions are listed in Table 1. Section 4.3 describes the high-level instructions.

4.1.5 Addressing Modes

Table 2 shows the supported addressing modes. Including the assignment of registers, the addressing modes are specified by using six bits with the short format and eight bits with the general format. Of these

Table 2. Addressing modes of the chip architecture based on the TRON specification.

Addressing mode	Notation	Note
Register direct	Rn	
Register indirect	@Rn	
Register relative indirect (16-bit displacement)	@(exp:16, Rn)	
Register relative indirect (32-bit displacement)	@(exp:32, Rn)	*
Immediate	#exp	
Absolute (16-bit address)	@exp:16	
Absolute (32-bit address)	@exp:32	
PC relative indirect (16-bit displacement)	@(exp:16, PC)	
PC relative indirect (32-bit displacement)	@(exp:32, PC)	
Stack pop	@SP+	
Stack push	@-SP	
Register based chained	@. .@(Rn, —)	*
PC based chained	@. .@(PC, —)	
Absolutely chained	@. .@(O, —)	

*not supported in the short format.

modes, the chained addressing provided by the TRON specification is a generalized indirect addressing function. The chained addressing mode allows a number of addressing primitives to be combined freely to generate a complex addressing mode. These addressing primitives include addition, scaling operation, and indirect memory reference. The addressing mode can be valuable for artificial intelligence applications and modular programming.

4.2 Memory Management

In the TRON architecture, address translation and memory management are implemented in the hardware. With embedded systems and similar applications, however, a memory management unit (MMU) is frequently not required. A two-bit field (address translation mode) is therefore provided in the PSW to indicate whether or not the MMU is used and whether or not address translation is carried out. By writing in this field, one can declare whether the address translation and memory protection are necessary or not.

The features of memory management in the TRON architecture are (1) virtual memory support to provide a logical space larger than the installed physical memory size, (2) a multiple logical space management function to maintain independence between contexts (tasks and processes) and thereby to facilitate program productivity, and (3) a ring protection function to provide memory protection between the operating system and user programs, and between shared data and user-specific data.

In order to achieve these functions, address translation is performed by two-level paging each time the memory is accessed. With the 32-bit chips, the 4G-byte logical address space is partitioned into two regions according to the S bit (MSB of the logical address). One is a 2G-byte unshared semi-space (US) and the other is a 2G-byte shared semi-space (SS). Each semi-space is further divided into 4M-byte sections, and each section divided into 4K-byte pages. By considering the logical address as a signed number, and allocating the SS to the negative (S=1) address region and the US to the positive (S=0) address region, continuity is guaranteed for each region when extending to a 64-bit address space.

There is an address translation table base register for each semi-space; these are called UATB and SATB, and are shown in Fig. 4. Only the UATB is changed whenever context switching takes place, thereby realizing a multiple logical space for each context. That is, a different physical space is allocated for each context, and the US is therefore used mainly by user programs. On the other hand, in the SS a common physical space is allocated for all contexts, and it is thus mainly used by the operating system and the interrupt handler.

In the TRON specification, memory is protected by a four-level ring. Protected information is specified for each page independently of SS/US classifications. If no MMU exists, page-based memory protection cannot take place. However, four-ring memory protection can be performed on addresses as an option.

4.3 High-Level Instructions Supporting OSs and High-Level Languages

In the TRON project, the chip architecture has been designed in conjunction with the operating system architectures of ITRON and BTRON specifications, on the basis of the total architecture concept. High-level instructions are provided for high-speed execution of system software. This approach ensures efficient execution of these operating systems on microprocessors based on the TRON specification, thereby improving the overall performance of TRON-based computer systems.

High-level instructions implemented for the operating system of the ITRON specification include context switching instructions (LDCTX and STCTX) and queue manipulation instructions (QSCH, QINS, and QDEL), while those for the operating system of the BTRON specification include variable-length bit field manipulation instructions (BVPAT, BVMAP, and BVCPY) and string instructions (SSCH, SMOV, SCMP, and SSTR).

The introduction of high-level instructions and the high-speed execution of frequently-used basic instructions are both important, but in different ways. Increasing the execution speed of the basic instructions does contribute to the average overall performance, but does not significantly improve the performance in critical situations. In realtime applications such as those in which an operating system is based on the ITRON specification, what is important is the response time of

the critical points rather than the overall performance. The introduction of high-level instructions is essential to improving the performance of these critical points. For example, the QSCH instruction can be used in the operating system of the ITRON specification when inserting a task that has just been made executable into the ready queue, and this point has a particularly strong influence on the response time in realtime applications.

Besides, variable-length bit field (bitmap) manipulation instructions make it possible to transfer a bit-block of any length, and are effective for executing high-speed window operation that will make a significant contribution to man-machine interface in the BTRON specification.

The bitmap instructions manipulate lines of bits. They read source bits and destination bits, perform logical functions on them, and store the results in the destination bit fields, as shown in Fig. 5. Successive bit pairs are obtained by scanning the source bit line and the destination bit line forward from the heads or backward from the tails. An additional feature of the TRON specification is that the scanning direction can be specified as either forward or backward. This is useful when the source bit field and the destination bit field are partly overlapped.

If the execution times of bitmap manipulation instructions are compared with a loop program for repeating 32-bit data transfer by the MOV instruction, the bitmap instructions are often two to four times as fast as the loop program. In addition to having a superior performance, the bitmap instructions are more flexible, because they can manipulate bit lines of any length at any position [4].

Software productivity is becoming more and more important. Any new microprocessor should provide an instruction set on which high-quality compilers for high-level languages can be built. In the TRON specification, great emphasis is placed on enabling compilers to generate efficient object codes. The provision of sufficient numbers of general registers and of a general instruction format with high orthogonality are two examples of such features. In addition, functions for operations on different data sizes and subroutine call instructions (ENTER and EXITD) are also provided for high-level languages. The subroutine calls in high-level languages require not only that the return address should be saved, but also that the frame pointer should be set, the local variable area created, and the general registers saved. These processes are, however, all contained within the one ENTER instruction. Similarly, those processes that must be carried out when exiting a subroutine call are all carried out by the EXITD instruction.

4.4 Exception, Interrupt, and Trap (EIT)

EIT processing occurs asynchronously with the normal execution of a program and requires the currently running program to be discontinued and another program to be executed. EIT stands for an exception, an interrupt, and a trap. An exception takes place because of an error or violation when an instruction is executed. The original program is restarted by re-executing the instruction that caused the exception from the beginning. An interrupt is independent of the program context, and may be raised by an external hardware signal or by a request from software. Certain traps take place in the same way as exceptions. Others are raised by the programmer to invoke system calls. The original program is restarted from the instruction next to the one that caused the trap.

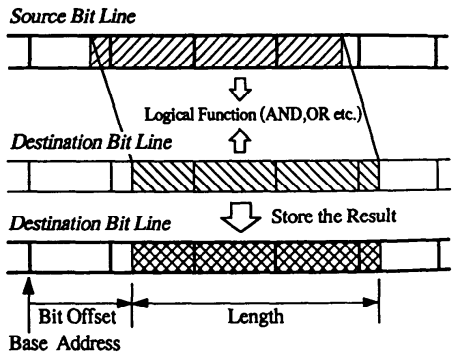


Fig. 5 Function of variable-length bit field (bitmap) instructions.

Since EIT functions have a significant influence on the structure of an operating system and its performance, careful consideration is needed in designing the EIT specification.

In the TRON specification, the occurrence of EIT's causes the processor to shift to an EIT processing state. In EIT processing, the processor fetches an EIT handler address and an EIT handler information from the EIT vector table (EITVT), and then sets the PC and part of the PSW according to this information. The PC and PSW at the time of the EIT occurrence and EIT-related information are saved on a stack. The format of these informations saved on a stack is determined according to EIT types by considering which information is necessary for system recovery. The program flow is then transferred to the EIT handler (software), which takes care of the cause of EIT. The EIT handler is a program that is built into and is part of the normal operating system. The execution of the return instruction (REIT) at the end of the EIT handler restores the PC and PSW that were saved in the stack and invokes the original program flow that was interrupted at the time of EIT occurrence.

EIT types are classified in detail and the vector number which is used to refer to the EIT vector table is assigned to each EIT type. Examples are shown in Table 3.

One of the features of the TRON specification is that not only the PC but also part of the PSW can be up-

Table 3. EIT types specified by TRON architecture.

Vector number	Address offset	EIT name	Remarks
—	—	System error	Operation is stopped with EIT processing.
00	000	Reset interrupt	
10	080	Self debug exception/trap	*1
11	088	Bus access exception Bus access trap	*2
12	090	Address translation exception Address translation trap	*2
13	098	Page out exception Page out trap	*2
14	0a0	Reserved instruction exception	
15	0a8	Privileged instruction violation exception	
16	0b0	Reserved function exception	
17	0b8	Reserved stack format exception	
18	0c0	Ring transition violation exception	
19	0c8	Odd address jump exception/trap	*1
1a	0d0	Zero divide trap	
1b	0d8	Invalid operand exception	
1f	0f8	Conditional TRAP instruction	
20(n=0) to 2f(n=15)	100 to 178	TRAPA instruction n (n=0 to15)	n is specified by the TRAPA instruction.
30(n=0) to 37(n=7)	180 to 1b8	Coprocessor disconnection exception CPID n (n=0 to 7)	n is specified by the CPID in the PSW.
38	1c0	Coprocessor execution exception	
39	1c8	Coprocessor command exception	
40(n=0) to 4e(n=14)	200 to 270	External interrupt INT n (n=0 to 14)	n corresponds to an interrupt level number.
50(n=0) to 5e(n=14)	280 to 2f0	Delayed interrupt DI n (n=0 to 14)	n corresponds to an interrupt level number.
5f	2f8	Delayed context trap	
80 to fe	400 to 7f0	External interrupt	The vector number is supplied externally for EIT processing.

*1 changes, depending on the implementation.
*2 can be either, depending on the implementation.

dated according to the entry in the EITVT at the start of the EIT handler. What this means is that it is possible to re-write the PSW bits that indicate interrupt mask, address translation specification field, and debugging mode, and then to start the EIT handler. This function allows flexible processing by the EIT handler. For example, it is possible to mask uninteresting interrupts automatically and to disable address translation temporarily.

Other useful features of the TRON specification are a delayed interrupt and a delayed context trap, both of which occur as a result of software. A delayed interrupt is used when registering a processing request not related to the user context or during a serialized processing sequence. An example of its application is the post-processing of an external interrupt. The delayed interrupt occurs when an interrupt request level is set by software in the DIR register shown in Fig. 4. When the request level is not higher than the IMASK level in the PSW register, the interrupt request is not accepted, and is held until the IMASK value is changed so that the acceptance condition is satisfied.

A delayed context trap is used to register a processing request for a user asynchronous event or during a serialized processing sequence.

5. VLSI Implementation the GMICRO/100

Currently six manufacturers are developing eight kinds of 32-bit microprocessor chip that are based on the TRON architecture specification and have software compatibility. Features of these chips are summarized in Table 4. There are now three chips for the GMICRO series: the GMICRO/100, GMICRO/200, and GMICRO/300. The GMICRO/200 is a standard microprocessor chip with on-chip MMU and cache memory. Binary coded decimal (BCD) arithmetic instructions and a coprocessor interface are also supported [5, 6]. The GMICRO/300 has a larger on-chip cache memory and more functions. Designed with a view to application in business workstations, it supports enhanced BCD arithmetic instructions, so that COBOL could be efficiently supported without much hardware [7]. As with the GMICRO/100, the TX1 and MN10400 are aimed at real memory systems [8, 9]. The O32 processor is implemented by utilizing a 0.8-micron CMOS process, and is noted for its aim of supporting fault-tolerant functions. In addition to its six-stage pipeline, a branch

Table 4. Specification of VLSI chips.

CHIP NAME	TX1	TX3	$G_{MICRO}/100$	$G_{MICRO}/200$	$G_{MICRO}/300$	MN10400	O32
CLOCK (MHz)	23	33	20, 25	20, 25	20, 25	20	33
MIPS							
PEAK	12.5	33	12.5*	12.5*	25*	20	15
AVERAGE	5	15	8*	8*	17*	8	10
DHRYSTONE	NA	NA	16 K*	16 K*	34 K*	NA	NA
No. of Inst.	92	138	92	122	133	95	102
MMU	NO	YES	NO	YES	YES	NO	YES
CACHE	NO	8 KB (I) 8 KB (D)	256 B (I)	1 KB (I) 128 B (Stack)	2 KB (I) 2 KB (D)	1 KB (I)	1 KB (I) 1 KB (D)
TRANSISTOR	450 K	1.2 M	340 K	730 K	900 K	400 K	700 K
TECHNOLOGY	1 μm	0.8 μm	1 μm	1 μm	1 μm	1.2 μm	0.8 μm
PACKAGE	155 PGA	155 PGA	135 PGA 160 QFP	135 PGA	179 PGA	144 PGA	176 PGA

*at 25 MHz

prediction table, an instruction cache, and a data cache, it also has features such as a processor bus comparator, a stack boundary check function, and a partial cache invalidation function [10].

The $G_{MICRO}/200$ uses the technique known as distributed cache, by which even a comparatively small cache memory is distributed and allocated to locations where there are likely to be bottlenecks in fetching and executing instructions. A block diagram is shown in Fig. 6. In fact, besides 1 K bytes allocated as an instruction cache and 128 bytes as a stack cache, the chip also has a four-entry branch window and a one-entry store buffer. The instruction cache has 64 entries and four words per entry, or a total of 1 K bytes. The stack cache is built in to improve the speed of the processes required for such stack operations as procedure calls and context switching. Despite its relatively small size, it can be expected to achieve a high hit rate. Each entry of the branch window has two values written to it: the head address of the branch instruction at the branch target address, and the address value, which is the branch target address plus four.

This section describes features of $G_{MICRO}/100$ and some considerations in the design of the chip, as well as evaluation results.

5.1 Design Objectives of the $G_{MICRO}/100$ [11]

High performance, specifically high-speed operation, is the primary objective of $G_{MICRO}/100$ design. The promising application of $G_{MICRO}/100$ includes high-performance and low-cost personal workstations supporting real me mory system and embedded industrial control equipments. A five-stage pipeline structure has been adopted to meet this requirement, in conjunction with a pre-jump processing scheme. The $G_{MICRO}/100$ does not have an on-chip MMU, cache, or FPU. Even the co-processor interface was removed. Hence, the die size of the $G_{MICRO}/100$ can be made relatively small, and this gives it the potential to be used as a core processor in an application specific IC (ASIC) processor. The $G_{MICRO}/100$ also supports some higher-level instructions such as variable-length bit field manipulation. These instructions are effective not only for bitmap processing but also for implementation of operating systems based on the ITRON specification. The block diagram of the $G_{MICRO}/100$ is shown in Fig. 7.

5.2 Pipeline Scheme

Pipelining is one of the most efficient technologies for implementing a high-speed microprocessor. A five-stage pipeline, as shown in Fig. 8, was adopted for the $G_{MICRO}/100$ [12]. The stages are IF (instruction fetch), D (instruction decode), A (operand address generation), OF (operand fetch), and E (execution). Each stage performs one operation in two clock cycles. The effective execution rate of basic instructions is two clock cycles per instruction, and thus the peak performance of the chip is 12.5 MIPS (million instructions per second) at a clock rate of 25 MHz. The functions of each pipeline stage are as follows:

IF-stage: The IF-stage contains an instruction queue (16 bytes) and a branch buffer (256 bytes). The branch buffer is a special purpose instruction cache with a direct-mapping scheme. When the queue holds no instructions, the branch buffer stores the instructions fetched from an external memory. For example, when the queue is flushed by execution of a branch instruction, the branch buffer caches the instruction that is called by the jump instruction. The branch buffer is also available for use as a general cache memory. The IF-stage fetches instructions from the external memory or the branch buffer and stores the instructions in the instruction queue. These instructions are sent to the D-stage via the instruction queue.

D-stage: The D-stage decodes the operation code of the instruction into a control code (D-code), which specifies an operation. Using an address offset and addressing mode in the instruction, an operand address code (A-code) is generated and sent to the A-stage. When two operands are fetched from the external memory for a memory-to-memory instruction, the D-stage sends the A-code twice.

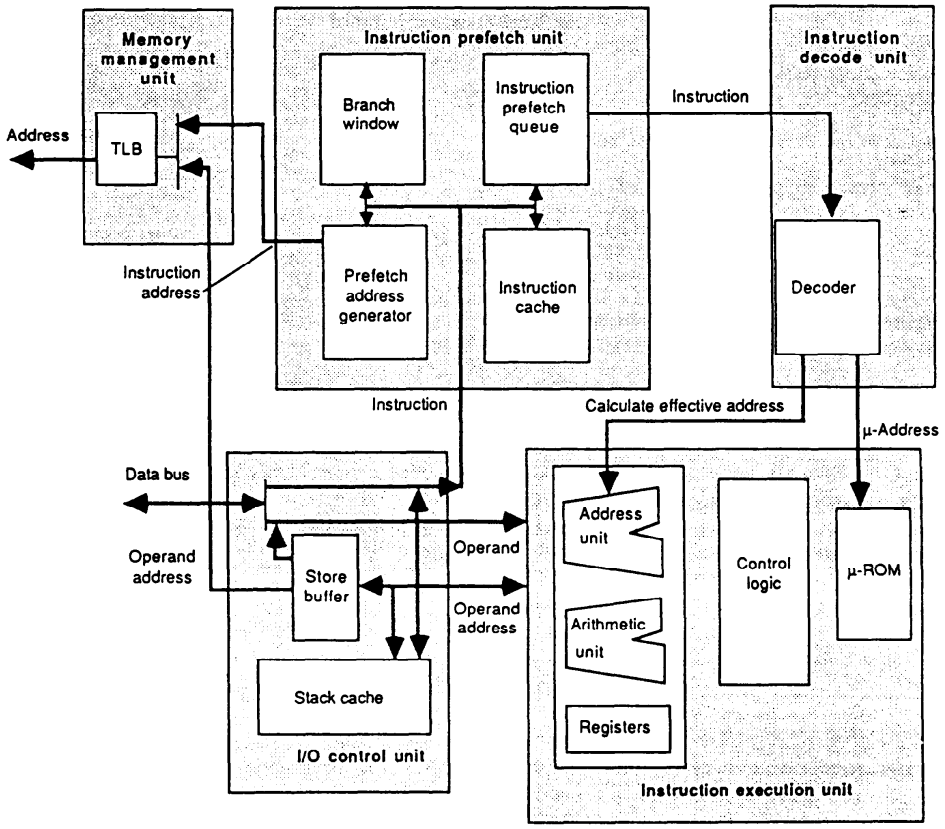


Fig. 6 Block diagram of the $G_{MICRO}/200$.

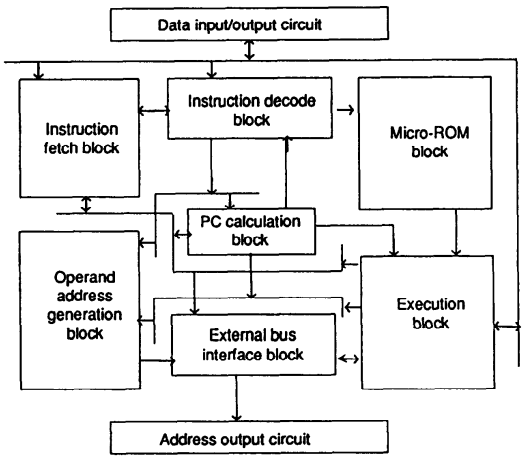


Fig. 7 Block diagram of the $G_{MICRO}/100$.

A-stage: Operand address generation and the second decoding of the instruction are performed in parallel in the A-stage. An operand address is generated from the A-code and sent to the OF-stage as an F-code. The D-code sent from the D-stage is decoded to generate the microprogram entry address (R-code).

OF-stage: The OF-stage fetches the operand specified by the operand address (F-code). A read-out of the microprogram form the microprogram ROM is also done in this stage. If the instruction requires multiple micro-instruction steps to execute the instruction, the micro-instruction fetch is repeated.

E-stage: The execution stage (E-stage) includes a 4-byte store buffer. The E-stage is controlled by the microprogram. An instruction is executed in one or more steps. The execution result is, if necessary, written into the external memory via the store buffer. Therefore, the E-stage can perform the following execution without waiting for the completion of its memory write cycle.

5.3 Pre-jump Processing Mechanism

Pipelining increases performance by executing multiple instructions in parallel. However, jump instructions work against an increase in performance. When a jump instruction is executed at the execution stage of a pipeline, any instruction stream in the pipe should be flushed, and a new instruction stream fetched. Since the

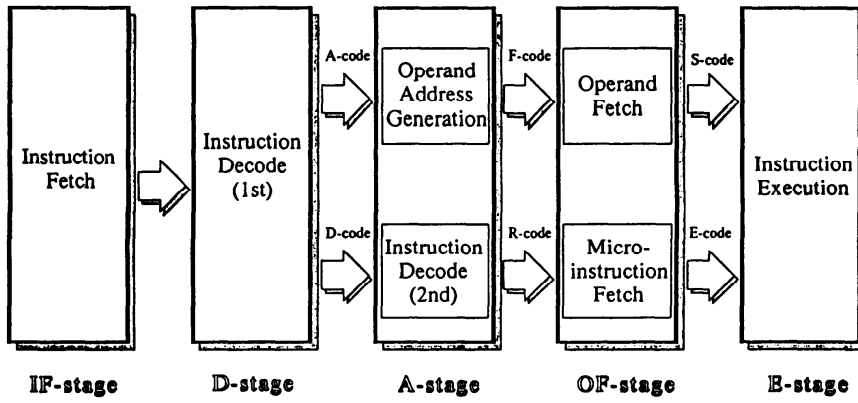


Fig. 8 Pipeline scheme of the $G_{MICRO}/100$.

$G_{MICRO}/100$ uses a multi-stage pipelining scheme, the performance loss caused by jump instructions has to be improved. An advanced pre-jump mechanism was applied to minimize the penalties related to jump instructions [13]. The pre-jump mechanism of $G_{MICRO}/100$ was built into the D-stage of the pipeline. There are two types of pre-jump processing: pre-branch for branch instructions, and pre-return for return instructions from subroutines.

5.3.1 Pre-branch Processing

The pre-branch processing for BRA (branch always) and BSR (branch to subroutine) instructions is simple. When these instructions are decoded in the D-stage, pre-branch is always taken. The branch address is calculated by adding the branch displacement to the PC value, using the PC adder. As for a conditional branch instruction (Bcc), a dynamic branch prediction mechanism is used. Pre-branch is taken in accordance with the branch history. The $G_{MICRO}/100$ contains a branch prediction table for the pre-branch processing of a Bcc instruction. This branch prediction table is constructed by a 1-bit X 256-entry direct mapping configuration according to the lower nine bits of a Bcc instruction address (the lowest bit is always 0). A prediction as to whether the Bcc instruction will or will not take a branch is made on the basis of the history of the most recently executed branch instruction.

ACB (add, compare, and branch) and SCB (subtract, compare, and branch) instructions have a high probability of branching, since these instructions are used for loop sequence controls. Therefore, the $G_{MICRO}/100$ always execute the pre-branch processing for these instructions.

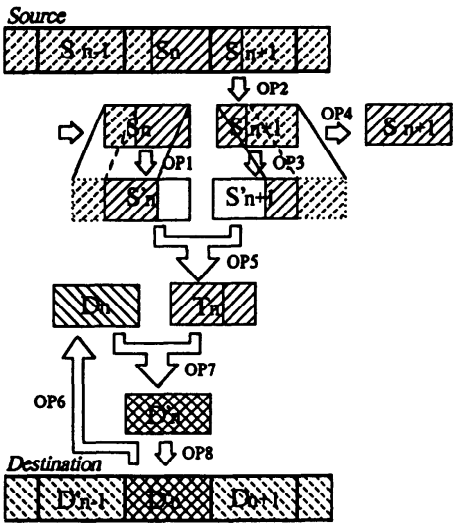
5.3.2 Pre-return Processing

For RTS (return from subroutine) and EXITD (exit and deallocate stack frame) instructions, the $G_{MICRO}/100$ takes pre-return at the D-stage, using an on-chip PC stack. The PC stack is the stack memory, consisting of a 32-bit X 8-entry configuration. The return address of the instructions depends on the instruction that calls the subroutine. Consequently, the abovementioned method of using the branch history is not effective. When a subroutine call instruction, such as a BSR instruction, is executed, the PC that specifies the return address is saved in the stack on the external memory. The $G_{MICRO}/100$ stores the return address in the on-chip PC stack as well as in the external stack. When an RTS instruction is decoded, the on-chip PC stack is popped up to obtain the return address (PC1) and a pre-return operation is performed. In detail, the chip pre-fetches the instruction specified by the address obtained from the on-chip PC stack. At the operand fetch stage, the return address (PC2) is fetched from the memory by popping the external stack. The value of PC1 is compared with the value of PC2 at the execution stage. If both values coincide, the pre-return is successful and jump processing is not required at the execution stage. Otherwise, jump processing is performed. As a result, the instruction stream on the pipeline is flushed and a new instruction specified by PC2 is fetched.

5.4 Microprogram Design

The pipeline scheme is effective for simple instructions that remain in every stage for the same number of clock cycles. However, it is not effective for complicated instructions such as bitmap instructions, because they spend most of the execution time in the E-stage. For them to be fast, the E-stage must be fast.

The $G_{MICRO}/100$ achieves fast execution of the bitmap instructions by pipelining the micro-operations in the E-stage [4]. Figure 9 is an illustration of the micro-operation loop in the BVMAP (manipulate of variable-length bit field) execution sequence, in which the source and the destination bit lines are processed by the 32-bit block repeatedly. The loop consists of eight micro-



OP1: Shift to the left the source block $S_n \Rightarrow S'_n$
OP2: Fetch the next source block S_{n+1} from the memory
OP3: Shift to the right $S_{n+1} \Rightarrow S'_{n+1}$
OP4: Save S_{n+1} in a working register for the next cycle
OP5: Logical OR of S'_n and $S'_{n+1} \Rightarrow T_n$
OP6: Fetch the destination block D_n from the memory
OP7: Perform a logical function between T_n and $D_n \Rightarrow D'_n$
OP8: Store the result in the memory

Fig. 9 Micro-operation Loop in the BVMAP Execution Sequence.

operations: OP1 and OP3 are shift operations; OP2, OP6, and OP8 are memory access operations; and OP4, OP5, and OP7 are ALU operations.

If the loop is programmed directly, it takes five microinstruction steps, as follows:

Step 1:	OP1 and OP2	(n)
Step 2:	OP3 and OP4	(n)
Step 3	Op5 and Op6	(n)
Step 4:	OP7	(n)
Step 5	OP8	(n)

Unrolling the loop and pipelining micro operations of the (n+1)-th cycle of the loop, the three-step loop is constructed as follows:

Step 1:	OP7	(n-1)
	OP1 and Op2	(n)
Step 2:	OP8	(n-1)
	OP3 and OP4	(n)
Step 3:	OP5 and OP6	(n)
Step4:	OP7	(n)
=Step 1	OP1 and OP2	(n+1)
Step:	OP8	(n)
=Step:	Op3 and Op4	(n+1)

The three-step loop is optimal because the memory bus is busy at every step. The BVCPY (copy variable-length bit field) and the BVMAP are optimized by the same method and achieve optimum use of the memory bus.

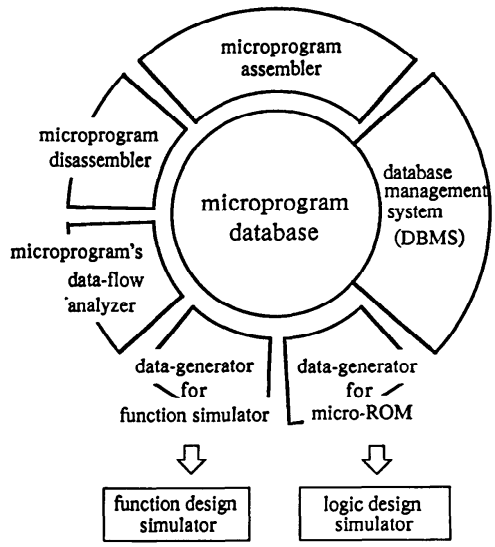


Fig. 10 Development environment of a microprogram.

Fig. 11 Microphotograph of the $G_{MICRO}/100$ chip.

The microprogram was designed in the environment shown in Fig. 10 [14-16]. The procedure for microprogram development was as follows:

1) The microprogram was designed on the assumption that any two micro-operations may run in parallel if they do not use the same function unit.

2) The microprogram was stored in a relational database management system, in which a micro-instruction word was stored in a record and a micro-operation was stored in the record's field. The INGRES data base management system was used for this purpose.

3) The database management system was used to retrieve a set of micro-operations in a field, and a set of micro-operation combinations in some fields. Based on the results of the retrievals, the micro-instruction for-

mat was designed to implement parallelisms required for all the instructions, including high-level instructions such as bitmap instructions, and to minimize the number of micro-operation fields.

5.5 Physical Design

Figure 11 shows a microphotograph of the $G_{MICRO}/100$ fabricated by using 1.0 micron CMOS process technology with two-level metalization. The chip is partitioned into three major sections. The upper part consists of the instruction fetch unit and the microprogram ROM. The middle part contains the control logic. The lower part contains the 32-bit datapath with 32-bit ALU, 32-bit barrel shifter, and the bus interface units. The bus interface unit arbitrates between the $G_{MICRO}/100$ and the external system bus, and sends and receives addresses, data, and control signals to and from other chips. The bus interface unit was divided into two parts, which were placed at different locations.

Most parts of the chip, which include the data path, branch buffer, ROM, and some decoder circuits, were designed by hand so that a small die size could be obtained. On the other hand, the control logic was designed by standard cell methodology. This standard cell was placed and routed automatically.

5.6 Evaluation

The characteristics of the chip are summarized in Table 5. The chip is housed in a 135-pin PGA package. The power dissipation of the chip is less than 1.5 watts in the worst case with a 25-MHz clock.

There are many benchmarks for evaluating microprocessor performance. Dhrystone is one of the most widely used. The fabricated $G_{MICRO}/100$ was operated on a single-board computer at a 25-MHz clock frequency. The performance of the chip was evaluated by the single-board computer, using the Dhrystone version 1.1 program written in C language. The results of the benchmark test showed 16,000 Dhrystones per second at a 25-MHz clock rate with no wait state. Based on the results, the average performance of the $G_{MICRO}/100$ is 8 MIPS. Figure 12 shows the performance as a function of the memory wait cycle for two clock rates, 20 MHz and 25 MHz. A low-end computer system often uses a relatively slow memory system. For one-wait state access at 25 MHz, the benchmark showed 12,600 Dhrystones per second. Since the current C compiler is still under development, the Dhrystone value is expected to be increased by improving the compiler optimization.

5.7 Total System for G_{MICRO} Products

In addition to microprocessor chips, the system provides versatile peripheral chips such as a floating-point coprocessor, an interrupt controller, a direct-memory-access controller, and a cache controller with cache memory.

Since a microprocessor with a new architecture requires related development tools in addition to related VLSIs in order to have practical applications, cross software such as assemblers, compilers, and a simulator/debugger is also provided to allow the development of application software. Application programs can be developed in C, Modula-2, or assembly languages and then translated to object modules. Since the $G_{MICRO}/100$ has a sophisticated pipeline structure, branch prediction scheme, and high-level instructions, the debugging environments require a function for monitoring the processor's internal states as well as one for monitoring the external bus cycle. The emulator was also developed by fully utilizing such debugging sup-

Table 5. Specification of the $G_{MICRO}/100$.

CLOCK	20 MHz, 25 MHz
PERFORMANCE	
PEAK	12.5 MIPS(at 25 MHz)
AVERAGE	8 MIPS(at 25 MHz)
BUS	
ADDRESS	32 BIT
DATA	32 BIT
MIN. BUS CYCLE	2 CLOCKS
PIPELINE	5-STAGE
BUFFER MEMORY	
INSTRUCTION QUEUE	16 BYTE
BRANCH BUFFER	4-BYTE × 64 ENTRY
STORE BUFFER	4 BYTES
BRANCH PREDICTION TABLE	1-BIT × 256 ENTRY
TECHNOLOGY	1.0 MICRON CMOS DOUBLE LEVEL METAL
DIE SIZE	11.47 × 8.89 mm
PACKAGE	135-PIN PGA, 160-PIN QFP
POWER DISSIPATION	MAX. 1.5 W

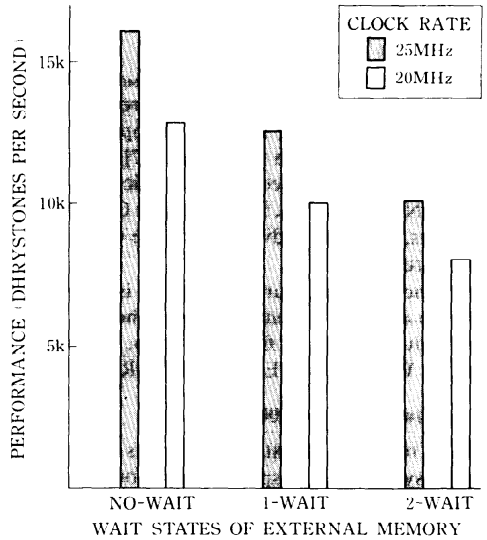


Fig. 12 Results of the Dhrystone benchmark test of the $G_{MICRO}/100$.

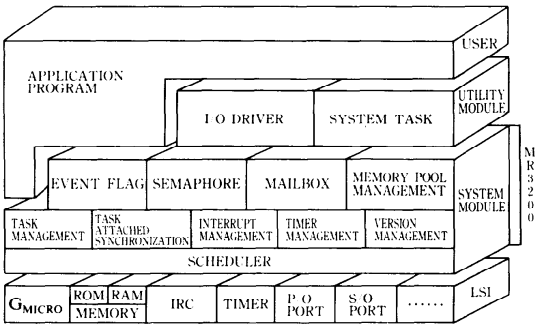


Fig. 13 Functional structure of MR3200, a realtime operating system based on the micro-ITRON specification.

port functions of the GMICRO/100 as step execution and the break function.

6. Implementation of a Realtime OS on the GMICRO/100-the MR3200

Since a primary objective of the GMICRO chips is to realize a high-speed computation environment in conjunction with operating systems, an operating system called MR3200 has been developed [17]. This is based on the micro-ITRON specification, which is a subset of the ITRON specification. It is a realtime OS for embedded systems, and takes advantage of the GMICRO instruction-set architecture.

The functional structure of MR3200 is shown in Fig. 13. There are nine system modules. The task management module supervises five task states, namely, RUN, READY, WAIT, SUSPEND, and DORMANT.

6.1 Features of MR3200

Quick response to external requests is crucial in a realtime system. The response time should be constant so that it can be estimated by the application software writer. The issue is thus to achieve a short and constant interrupt masking time that does not depend on the number of tasks or on the priority. Since the state of the ready queue changes according to the number of tasks and priority in general, a constant response time is hard to obtain. However, through the use of specific instructions of the GMICRO/100 such as variable-length bit field manipulation instructions and bit manipulation instructions, MR3200 enables task scheduling or dispatching to be executed by a single instruction, resulting in a constant response time.

The ready queue is constructed by a bidirectional queue. It is managed by a bit field table with a size of 256 bits, each bit of which corresponds to the task priority and indicates the existence of any priority tasks. Dispatching means to search for the ready task that has the highest priority, and then to switch the context. It can be implemented by utilizing the BVSCH instruction, which is a variable-length bit field manipulation instruction. This instruction searches for a specific bit with the value "1" from the bit field table. The task can then be obtained from the ready queue. This gives a more constant response time than using compare and branch instructions repeatedly. The time required to search for a bit from the bit field increases by steps of 0.2 μsec every time a 32-bit boundary is passed. It takes approximately 1.7 μsec to search for a bit from the first bit to the last 256th bit. Since a searching time of less than 1.7 μsec was obtained, a realtime response for the external request was assured without the maximum masking time for interrupt being affected.

In system call processing, a task is inserted into or deleted from the ready queue. This scheduling can be performed by using the BSET (set bit) and BCLR (clear bit) instructions to set and clear the priority bit in accordance with changes in the queue status. These instructions can set or clear any bit, and thus the ready queue can be handled in a constant time independent of the number of priorities.

Table 6. Specification of MR3200.

TARGET PROCESSOR	GMICRO SERIES
NUMBER OF TASKS	65535 MAX
PRIORITIES	1 TO 255
NUMBER OF SYSTEM CALLS	53
OS CODE SIZE	
MINIMUM SYSTEM	2100 bytes
MAXIMUM SYSTEM	11600 bytes
TASK SWITCHING	
WAKE-UP TASK	18 μsec
EVENT FLAG SET	26 μsec
MAXIMUM INTERRUPT MASKING TIME	
WAKE-UP TASK	9 μsec
CHANGE PRIORITY	15 μsec

6.2 Performance of MR3200

The performance of the realtime OS MR3200 was evaluated with the GMICRO/100 microprocessor [18]. A task was woken up in 18 μsec by the wake-up-task system call. The interrupt masking time for the system call was 9 μsec. The maximum interrupt masking time was 15 μsec for the change-priority system call. Since MR3200 has a library structure, a system can be configured by linking only system calls that should be used in the application. The code size of the OS that utilizes all the system calls is approximately 11,600 bytes, while the size is 2,100 bytes when four basic system calls are utilized to implement the smallest real-time systems with multitask function. The specifications of MR3200 are summarized in Table 6.

7. Conclusion

The chip architecture of the TRON specification has been designed as part of a total architecture in parallel with the design of TRON specifications for operating

systems. The instruction set architecture is optimized for implementing a compiler as well as various operating systems.

A 32-bit microprocessor, the GMICRO/100, based on the TRON specification was implemented by using a pipelining structure with a unique pre-jump scheme and a new microprogram design strategy. The results of the benchmark test showed 16,000 Dhrystones per second at a 25-MHz clock rate. A realtime operating system, MR3200, was also developed according to the micro-ITRON specification. Thanks to the high-speed operation of the GMICRO/100, a task wake-up time of 18 μsec was realized.

A number of semiconductor manufacturers have started to supply samples of 32-bit general-purpose microprocessors based on the TRON specification. It is expected that both suppliers and users of the chip will make and improve the support environment for efficient development in cooperation and will compete freely and fairly to sell their products based on this architecture, resulting in increasingly widespread use of the microprocessors and related application products. It is essential for such activities that we should continue to maintain rigidly the concept of an open architecture that is not subject to limits imposed by any particular manufacturers. We hope that this architecture will be of great service as common property of mankind.

References

1. SAKAMURA, K. The Objectives of the TRON Project, TRON Project 1987, Springer-Verlag (December 1987), 2-16.
2. SAKAMURA, K. Architecture of the TRON VLSI CPU, *IEEE Micro* (April 1987), 17-31.
3. SAKAMURA, K. TRON VLSI CPU: Concepts and Architecture, TRON Project 1987, Springer-Verlag (December 1987), 199-238.
4. SHIMIZU, T. et al. A 32-Bit Microprocessor with High Performance Bit-map Manipulation Instructions, Digest of tech. papers, ICCD '89 (October 1989), 406-409.
5. TAKAGI, K. et al. Outline of GMICRO/200 and Memory Management Mechanism, TRON Project 1987, Springer-Verlag (December 1987), 259-272.
6. INAYOSHI, H. et al. Realization of GMICRO/200, *IEEE Micro* (April 1988), 12-21.
7. ITOH, M. Architecture Characteristics of GMICRO/300, TRON Project 1987, Springer-Verlag (December 1987), 273-280.
8. NAMIMOTO, K. et al. TX Series Based on TRONCHIP Architecture, TRON Project 1987, Springer-Verlag (December 1987), 291-308.
9. KIYOHARA, T. et al. Design Considerations of the Matsushita 32-bit Microprocessor for Real-Memory Systems, TRON Project 1988, Springer-Verlag (December 1988), 263-272.
10. ITO, N. et al. Architectural Features of the OKI 32-bit Microprocessor, TRON Project 1988, Springer-Verlag (December 1988), 247-262.
11. TOMISAWA, O. et al. Design Considerations of the GMICRO/100, TRON Project 1987, Springer-Verlag (December 1987), 249-258.
12. SHIMIZU, T. et al. A 32-bit Microprocessor Based on the TRON Architecture: Design of the GMICRO/100, Digest of tech. papers, Compcon '88 (February/March 1988), 30-33.
13. YOSHIDA, T. et al. 32-bit MPU GMICRO/100 Improving Pipeline Efficiency by a Pre-Jump Scheme, Nikkei Electronics, No. 477 (1989) (in Japanese) 185-196.
14. IWATA, S. et al. Implementation of a 32-bit microprocessor GMICRO/100 Based on the TRON Specification-(1) Implementation and Evaluation of Microprogram, Digest of tech. papers, Information Processing Society of Japan (March 1989) (in Japanese), 1518-1519.
15. WATANABE, Y. Implementation of a 32-bit microprocessor GMICRO/100 Based on the TRON Specification-(2) Management of Microprogram and Design of Microcode by Relational Database, Digest of tech. papers, *IPS Japan* (March 1989) (in Japanese), 1520-1521.
16. SAWAI, K. et al. Implementation of a 32-bit microprocessor GMICRO/100 Based on the TRON Specification-(3) Microprogram Support System on the Relational Database, Digest of tech. papers, *IPS Japan* (March 1989) (in Japanese), 1522-1523.
17. TSUBOTA, H. et al. Realtime OS based on the Micro-ITRON Specification for GMICRO Series, MR3200, Workshop on Microcomputers and Workstations, *IPS Japan* (June 1989) (in Japanese).
18. YAMAMOTO, O. et al. Performance Evaluation of MR3200: A Realtime OS based on the Micro-ITRON Specification, Technical Papers of 3rd TRON Technical Study Group Meeting, (1989) (in Japanese), 39-50.

(Received November 6, 1989)