*Invited Paper*

# Architecture of an AI Processor Chip (IP1704)

MITSUO SAITO*, TAKESHI AIKAWA*, TSUKASA MATOBA*,
MITSUYOSHI OKAMURA* KENJI MINAGAWA* and TADATOSHI ISHII**

The CPU of the AI processor (AIP) called IP704 was developed for Prolog and Lisp, based on RISC architecture with hardware supports. It has been proved that IP704 architecture is effective for both AI languages and general-purpose languages.

An AI processor chip (IP1704) is being developed as a direct successor of the IP704. The architecture has been modified and refined to fit onto a single chip and to improve the execution speed.

Features newly developed for the IP1704 include Overlapping of the decode and register-read stages using a combination of the hardware decoder and micro-programs, and a delayed cache hit check with delayed writing.

It is shown that a RISC-based processor with suitable hardware support is applicable to VLSI and also gives high performance AI languages.

## 1. Introduction

A number of AI processors that give fast execution and a good programming environment have been developed in the past decade [1, 2]. RISC architecture, on the other hand, was proposed, to improve the execution speed for general-purpose languages [3]. Some attempts at porting Lisp or Prolog to RISC architecture have been reported. Lisp on MIPS-X is one software approach [4]. SPUR is a typical approach to making the RISC architecture suitable for Lisp [5]. We developed an AI processor (AIP), consisting of discrete parts [10], based on a new architecture concept. The IP704 is the CPU of the AIP, and is based on RISC architecture with some modifications and extensions to suport simplified WAM (Warren Abstract Machine) instructions [6] and instructions for Lisp. In contrast to SPUR, the IP704 is designed for Prolog and employs micro-program control. Micro-programs and additional special hardware for tag handling and exception handling have been implemented to speed up Prolog and Lisp programs and to improve the program safety without sacrificing the essential advantages of RISC architecture, such as simple hardware architecture, simple instruction format, large register file, delayed branching, and single-cycle execution for simple instructions.

Prolog, Lisp, and C compilers were designed to generate the IP704's primitive instructions, which are executed directly by the hardware. Optimizers manage IP704 pipeline restriction and register allocation.

An AI processor chip (IP1704) is being developed as a direct successor of the IP704. The architecture has been modified and refined to fit onto a single chip and to improve the execution speed [11]. Features newly developed for the IP704 include overlapping of the decode and register-read stages using a combination of the hardware decoder and micro-programs, and a delayed cache hit check with delayed writing.

The main issues of this paper are the design concept and hardware architecture of the IP1704. The design concept is introduced first, and its implementation is described in detail. Then, sone simple performance evaluation results are given.

## 2. Design Concept

It has recently been discovered that reduced instruction set computers (RISC) with a faster cycle time are more cost-effective than traditional computers, especially for VLSI. Though these computers have a larger semantic gap, they can handle this by using compiler techniques, including optimization.

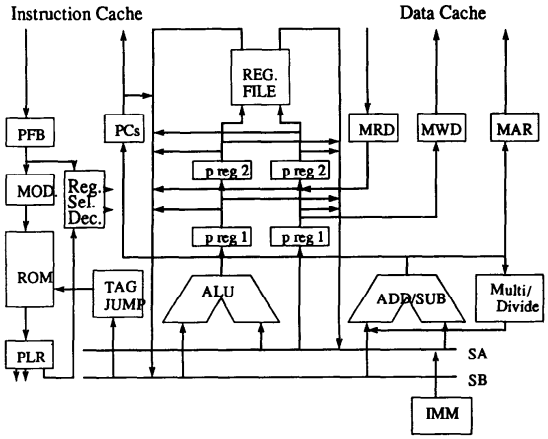
The semantic gap of AI languages is larger than that of conventional languages. Compiler techniques can be extended to AI languages. If this is done, a register machine will provide better performance than a stack machine, even for AI languages.

In practice, however, the cache miss problem results from the large code size, especially in Prolog, owing to the flexibility of unification. Hardware supports are also effective for run-time checking and automatic memory management. The following strategies have therefore been established:

(1) RISC architecture is employed as a basis for simple hardware.

*Information Systems Laboratory, Toshiba Research and Development Center, Toshiba Corporation, 1 Komukai, Toshiba-cho, Saiwai-ku, Kawasaki-shi, Kanagawa 210, Japan

**Information and Communication Systems Laboratory, Toshiba Corporation, 70 Yanagi-cho, Saiwai-ku, Kawasaki-shi, Kanagawa 210, Japan




| | | | |
|---|---|---|---|
| PFB | PreFetch Buffer | ALU | Arithmetic Logical Unr |
| MOD | Instruction MODifier | MWD | Memory Write Register |
| ROM | Read Only Memory | MRD | Memory Read Register |
| PLR | Pipe Line Register | SA | Source bus A |
| PC | Program Counter | SB | Source bus B |
| MAR | Memory Address Register | IMM | IMMediate generator |

Fig. 1 Architecture of the IP1704.

(2) RISC architecture will be extended and modified for AI languages.

(3) The extended hardware for AI languages should not affect the cycle time, and should be sufficiently small to be integrated onto a chip.

In line with the above strategies, it has been decided to add the following functions to pure RISC architecture to support both Prolog and LISP efficiently.

(1) Combination of hardware decoding and micro-program control

(2) Instruction-modifying mechanism

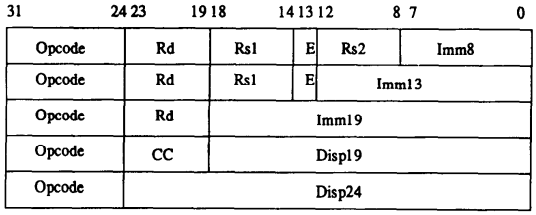
(3) Two-independent write bus

(4) Tag handling hardware and trap mechanism

(5) Expanded memory system.

To design these additional functions, we estimated the effects of the individual functions on the overall performance, considering the frequency of each WAM instruction [7] for Prolog. We counted and compared the number of clock cycles for each WAM instruction executed on the IP1704 and on a pure RISC machine. By considering the frequency of each instruction, we estimated that the functions would increase the speed approximately three times on the average. An example of the clock count for one inference in the case of the Prolog program "append" is given below. The source lists of pure RISC and the IP1704 are given in Appendix 1.

| | |
|---|---|
| Pure RISC | 55 clk. |
| Micro-program | 36 clk. |
| All except (2) | 19 clk. |
| All except (3) | 25 clk. |
| All except (4) | 33 clk. |
| All except (5) | 24 clk. |
| All | 14 clk. |




Rd : Register destination
Rs : Register source
Imm : Immediate data
Disp : Displacement
E : Extention bit

Fig. 2 Instruction Format.

The additional functions are described in detail in the next section.

## 3. IP1704 Architecture

Figure 1 shows a CPU block diagram of the IP1704. It has a simple architecture. A micro-program controller, two register buses, a tag jump controller, and an instruction modifier are added to the ordinary RISC CPU's.

The IP1704 is basically a 32-bit machine with a 32-bit fixed-length instruction set. As shown in Fig. 2, the instruction format is very simple. It has a fixed length 8-bit Opcode field, and a maximum of three register fields with 5-bit length or immediate and displacement field.

A five-stage pipeline is employed for all instructions, namely fetch, decode/register-read, execute, memory-access, and write, whereas a three-stage pipeline was used for the IP704. The memory-access stage is idle except for memory access instructions (see Fig. 3).

The register file is a plain four-port 48-word-by-32-bit register. Sixteen words are used for special instructions and micro-programs. A register window system is not used, because both Prolog and Lisp often include deep recursion. In such cases, the recursion may easily exhaust the register window stack, which imposes a large overhead on the hardware trap.

The functions employed for faster execution of AI programs are given in the following sections.

### 3.1 Combination of Hardware Decoding and Micro-program

The major benefits of micro-program control over pure RISC architecture, especially for AI programs, are as follows:

(1) Control flexibility

Word width is not restricted, and no decoder is needed for the horizontal micro-program approach. This is therefor an easier and faster way of controlling additional hardware for AI languages than the 'macro-instructions in ROM' approach.

(2) No conditional or multiway jump as penalty

Wide micro-program words and a small address space

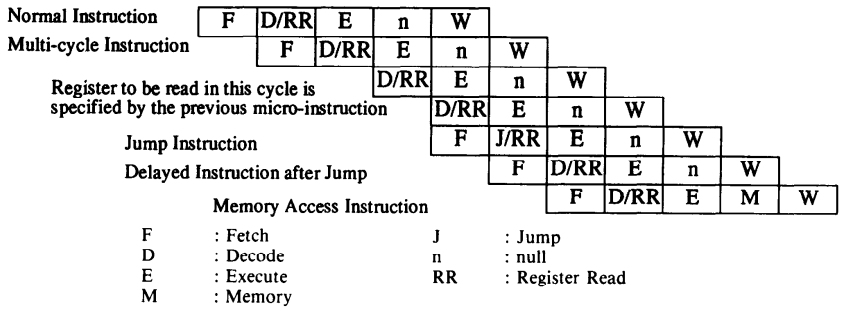



Fig. 3 Pipeline Stages of the IP1704.

make it possible to include conditional or multiway jump fields and ALU functions in one word. Thus, conditional or multiway jumps are executed in parallel with ALU operations, which are frequently used in AI programs for run-time data type checking. The multiway jump is especially effective for the unification and fail process in Prolog.

(3) Small macro-code size

Since Prolog has little user-definable control structure, a flexible and complex control mechanism is required for the Prolog machine. If a Prolog program is compiled for a RISC machine, the code size becomes large, the cache miss rate increases, and the performance is degraded.

The IP1704 has 96 bits x 1 K words of ROM for a micro-program. AI programs take advantage of the parallelism of micro-program control. The following is an example of a micro-instruction used in the get·list instruction:

xxx x, x, s, h, xxxxxxxxxxxxxxx(previous micro-instruction)

prsb nl, s, x, x, alls, gltrl, ww, s, hep, wm at sx

This micro-instruction performs the following operations:

(a) Set the memory address register to the data of the '*s*' register, and set the access mode to the heap area, (b) Write the data of the register '*h*' merging the list tag to the memory address specified by (a), and also set the register '*s*', (c) Compare register '*s*' with '*h*' and set the condition code, (d) Set the CPU mode to the 'write mode' and jump to the label 'gltrl', in one cycle.

These functions correspond to several instructions for a RISC processor.

As mentioned above, micro-program control is quite suitable for supporting AI languages. However, micro-program decoding is slower than that of the hardware decoder. It takes one pipeline stage to decode for a micro-program. This is claimed to be one of the major benefits of RISC over CISC.

It was not a significant problem for the IP704, because the access speed of the register file was fast enough in relation to its cycle time. However, it is a serious problem for the IP1074, because the access speed of the register file is comparable to its cycle time, so a full pipeline stage is required. One extra pipeline stage before the execution stage causes a large overhead for branch instructions. In this case, it needs two delayed slots, which are very difficult for the compiler to fill.

To circumvent this disadvantage, the hardware decoder and micro-program have been combined in the IP1704. The registers to be read or the immediate data needed in the first cycle of the macro-instruction are hardware decoded, and the registers are read in parallel with micro-program ROM. Then, the registers to be read in the next clock cycle are specified by the present microinstruction (see Fig. 3) Because the decode and register read are overlapped, the micro-program does not affect the cycle time or pipeline stage. Micro-program control can be efficiently employed without sacrificing the advantages of RISC.

### 3.2 Instruction Modifying Function

In Prolog, the arguments of predicates are not defined as either input or output until the execution stage. To support this capability, the processor mode corresponding to input or output is defined in WAM, and some instructions behave differently according to the processor mode. The IP1704 has the capability to modify instruction behavior according to the processor mode, without additional cycle time. This function is also useful for Lisp garbage collection.

### 3.3 Two-independent Write Bus Architecture

Two-independent write bus architecture consists of two destination buses and two write ports register file. It makes it possible to define instructions that modify two registers, or modify one register and access memory in one cycle. The stack operation, frequently used in AI languages, is executed in one cycle. The architecture is also effective for automatic memory management.

### 3.4 Tag Handling and Trap Mechanism

Data tag checks and data tag merges are managed by

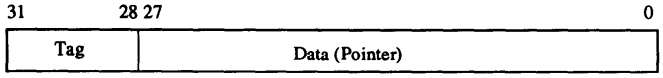



Fig. 4 Data Format.

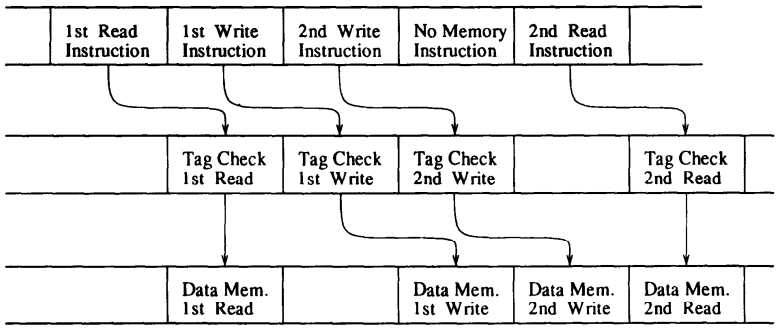



Fig. 5 Memory Access Timing.

special hardware. The data format is shown in Fig. 4. The tag size is limited to four bits in order to keep the wide virtual address space in conventional 32-bit words. Therefore, only 16 different data types are distinguished. The trap mechanism for exception handling has been extended to detect data tag mismatches, avoiding extra check cycles. It also detects data types that are not supported by the hardware.

### 3.5 Cache Memory Support

The cache memory support function has been integrated in the IP1704. When the access speeds of data memory and tag memory are comparable, the additional time needed to determine whether a hit or miss has occurred can be avoided by pipelining the hit check. This is called "delayed hit check," and is employed to give a faster cycle time.

Generally, the delayed hit check needs two cycles to write. As shown in Fig. 5, both data memory read and tag memory read are performed in the same cycle, and it is sufficient to cancel the read data if the tag data are different. However, the data cannot be written until the tag has been checked, or else data in the cache will be lost. To circumvent this disadvantage, the IP1704 has the capability to check the tag for the second address while the first data are being written. By using this pipeline technique, both 'Read' and 'Write' to cache memory are completed in one cycle, unless 'Read' is directly followed by 'Write.' This restriction can easily be handled by the compiler optimizer.

### 3.6 Memory Interface

The IP1704 employs the so-called Harvard Architecture. The typical system configuration is shown in Fig. 6.

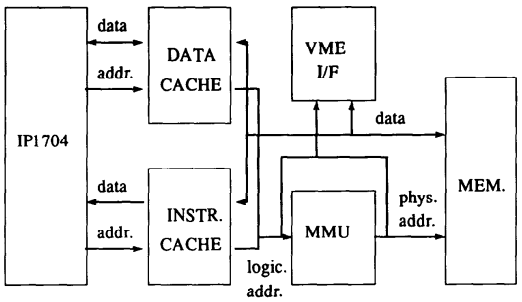



Fig. 6 System Configuration.

This architecture makes the IP1704 run without data and instruction bus collision for memory accesses. The disadvantage of the Harvard Architecture is cache inconsistency between the data and the instruction cache memories. Since self-modifying programs and meta-programming are allowed in AI languages, these may cause problems. For this reason, the IP1704 supports special instructions to flush the instruction cache memory and data cache memory, and the access type check mechanism.

Each memory access instruction has an access type (code, static-data, stack, heap, etc.) to check whether it is a legal memory access. General instructions are not allowed to modify the code area. An instruction that modifies the code area should be followed by cache flush instructions to maintain consistency.

The access type bits are also useful for detecting illegal access to different memory areas. When the heap area is exhausted, an access to the non-existent heap area is recognized as an illegal memory access. It trig-

gers garbage collection, and is notified to the trap handler, in the same way as the page fault signal. In this way, the software need not check the region boundary.

### 3.7 Instruction Set

The instruction set of the IP1704 has been established with a view to the possibility of the optimization, the efficiency of each primitive function, trap handling capability, and the ROM size. For the C language, the optimization possibility is important, because essentially no checking is done during execution. The combination of simple instructions with the minimum number of cycles gives the best performance. The instruction set is therefore restricted to single-cycle instructions, except for multiplication and division.

In contrast, rather complex instructions (basically WAM instruction) have been adopted for Prolog. However, these instructions are still restricted to simple functions, so as not to sacrifice the possibility of optimization and the consistency when a trap occurs. These instructions make Prolog programs run with little overhead for data type checking and memory management, particularly for the unification function.

The tag checking instructions and generic arithmetic instructions have been implemented for Lisp. They normally take the minimum number of cycles. They also handle exceptional cases such as big-num, using trap routines.

### 3.8 Specifications of the IP1704

Table 1 summarizes the specifications of the IP1704.

## 4. Implementation

Since the IP1704 is a direct successor of the IP704, most of its functions are the same as those of the IP704. Because of the restriction of chip size and the speed factor difference, the hardware architecture has been changed a lot. However, the impact on the software of this difference in architecture has been minimized. The major modifications are as follows:

(1) Opcode has been changed to simplify the hardware decoder.

(2) Complex instructions have been divided into combinations of simpler instructions become of the ROM size restriction and trap consistency.

(3) Delayed branching is adopted for better performance.

The IP1704 has approximately 113,000 transistors. Nearly half of them are used for ROM, but this occupies less than 10% of the total space, as shown in Fig. 7. This means that the micro-programs are still cost-effective for VLSI implementation.

## 5. Benchmarks

Some benchmark tests have been done to evaluate the IP1704's performance. The compilers for the IP1704

Table 1 IP1704 Processor Specifications.

| | |
|---|---|
| **Functional specifications** | |
| Registers | 32 bit*32 |
| Data format | 32 bit or 4-bit tag+28-bit data |
| Address space | 256 M Bytes for Lisp and Prolog, 4 G Bytes for C |
| Instructions | 157 |
| Cache memory | 4 K ~ 256 K Bytes direct mapping support |
| **Physical specifications** | |
| Clock | 16.7 MHz (60 ns) expected |
| Design rule | 1.2 $\mu$m CMOS |
| Chip size | 14.5 mm*14.5 mm |
| Transistor | 113 K |

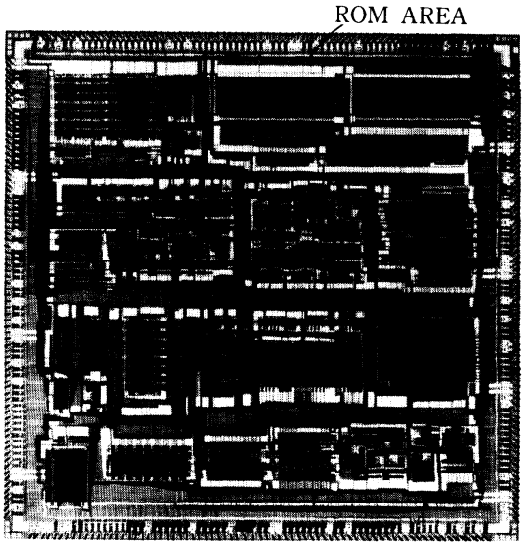



Fig. 7 Layout of the IP1704 Processor Chip.

have not yer been completed. A translator has been developed to convert the IP704's assembly code to that of the IP1704. Benchmarks are obtained by using the translator and the IP1704 hardware simulator. The tests assume that the cache memory hit rate and clock speed are 100% and 60 ns (16.7 MHz), respectively.

Table 2 shows the simulation results for Prolog. These results show that the IP1704 has a better performance clock ratio than the IP704. This is mainly because of the delayed branching and refinement of the micro-code.

Table 3 shows a part of the results of the LISP Gabriel benchmark test [9]. They were obtained by a simulator, and manual optimization was done only for major loops. The results show that the IP1704 is more than 10 times as fast as the Symbolics 3600. It has about 1.3 times the performance of MIPS-X simulated results (50 ns), even without optimization, and about 1.6 times with optimization, in spite of its slower cycle time

Table 2 Performance of Prolog (first prototype).

| Benchmark | IP1704 | IP704 |
|---|---|---|
| Append (deterministic) | 1.2 M lips 14 clk/Inf. | 667 K lips 15 clk/Inf. |
| Nrev 30 Total 497 Inf. | 1.1 M lips 7140 clk | 534 K lips 9307 clk |

Table 3 Performance of Lisp. Gabriel Benchmark times in msec/ratio

| Benchmark | IP1704 non. | IP1704 opt. | IP704 | 3600 | MIPS-X |
|---|---|---|---|---|---|
| tak | 75 | 61/0.81 | 114/1.52 | 430/5.73 | 72/0.96 |
| destructive | 201 | 146/0.73 | 304/1.51 | 2180/10.8 | 361/1.79 |
| derivative | 320 | 281/0.88 | 520/1.62 | 3790/11.8 | 381/1.19 |
| average ratio | 1 | 0.81 | 1.55(1.91) | 9.46(11.7) | 1.31(1.62) |

Table 4 Performance of C language. benchmarks/ratio

| Benchmark | IP1704 | IP704 | SUN3/260 | SUN4/260 |
|---|---|---|---|---|
| Dhrystone 1.1 No Optimized | 14700 | 8400/1.75 | 5800/2.54 | 10500/1.4 |
| Dhrystone 1.1 Optimized | 22000 | 10500/2.1 | 6350/3.46 | 19000/1.16 |

(60 ns).

The performance for the C language (Dhrystone) is presented in Table 4. Library functions (strcopy, strcmp) have been optimized manually as much as possible for both results, and the optimization of the Dhrystone program has been done manually in order to achieve the same optimization level as that of SUN4. The benchmark test shows that it has a better performance than that of SUN4 with the same cycle time.

## 6. Conclusion

We designed the architecture of the AI chip IP1704 to fit onto a VLSI by modifying the IP704 of the board version. We attained a higher clock performance ratio than with the IP704, in spite of the restriction on the transistor count of the VLSI. In achieving this goal, we greatly changed the architecture, while minimizing the impact on the software.

The delayed cache write and the combination of micro-program decoding and hardware decoding have proved to be very effective in obtaining a high performance for AI languages.

The chip's predecessor, the IP704, obtained a very high performance in comparison with other AI machines. All benchmarks show that the IP1704 has a better performance clock ratio than the IP704. Though the evaluation is still incomplete, the newly developed IP1704 has inherited the characteristics of the IP704. Therefore, we believe that the average performance of the IP1704 is about twice that of the IP704.

The IP1704 is currently in the evaluation phase. We believe that this low-cost and high-performance VLSI will be extremely useful in practical AI applications.

**References**


1. MOON, D. A. Architecture of the Symbolics 3600, *The 12th Annual International Symposium on Computer Conference Proceedings* (June 1985), 76–83.
2. UCHIDA, S. and YOKOTA, M. Outline of the Personal Sequential Inference Machine: PSI, *New Generation Computing*, 1-1, 1 (1983).
3. PATTERSON, D. A. Reduced Instruction Set Computers, *Comm. ACM* (Jap. 1985), 8–21.
4. STEENKISTE, P. LISP on a Reduced-Instruction-Set Processor: Characterization and Optimization, Stanford University Technical Report: CSL-TR-87-324 (March 1987).
5. TAYLOR, G. S. et al. Evaluation of the SPUR Lisp Architecture, *Proceedings of the 13th Annual International Symposium on Computer Architecture*, ACM, Tokyo (June 1986), 444–452.
6. WARREN, D. H. D. An Abstract Prolog Instruction Set, Tech. Note 309 Artificial Intelligence Center, SRI Internationa (Oct. 1983).
7. TICK, E. Prolog Memory-Referencing Behavior, Tech. Report No. 85-281, Computer Systems Laboratory, Stanford Univ, (Sept. 1985).
8. WILK, P. F. Prolog Benchmarking, AIAI University of Edinburgh, Technical Report 14 (May 28, 1984).
9. GABRIEL, R. P. Computer Systems Series. Volume?: Performance and Evaluation of Lisp Systems, The MIT Press (1985).
10. SAITO, M. et al. An AI Processor: RISC Architecture with Hardware Supports for AI Languages, IEEE Denshi, Tokyo 1988, No. 27, (Feb. 1989), 39–43.
11. MAEDA, K. et al. Mechanisms for Archiving Parallel Operations in a Sequential VLSI AI Processor, Procedings of the Third Parallel Processing Symp. (March 1989).


(Received October 5, 1989)