TR-274

Performance and Architectural Evaluation of the PSI Machine

by

H. Nakashima, M. Ikeda, (Mitsubishi) K. Taki and K. Nakajima

July, 1987

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

PERFORMANCE AND ARCHITECTURAL EVALUATION OF THE PSI MACHINE

Kazuo Taki Katsuto Nakajima Hiroshi Nakashima and Morihiro Ikeda
ICOT* Mitsubishi† Mitsubishi‡

Abstract

We evaluated a Prolog machine PSI (Personal Sequential Inference machine) for the purpose of improving and redesigning it. In this evaluation, we measured the execution speed and the dynamic characteristics of cache memory, register file, and branching hardware introduced for high-speed execution of Prolog programs.

Execution speed of the PSI firmware interpreter was found to be comparable to that of the DEC-10 Prolog compiled code on the DEC-2060. It was also found that PSI was faster than DEC for executing programs containing much unification and backtracking that require runtime processing.

With the cache memory, the hit ratio for application programs was found higher than 96%; this demonstrates that the Prolog execution has much memory access locality. The memory access frequency and the appearance ratio between Read and Write command were also investigated.

Concerning the register file, use rate of each dedicated access mode was measured and effect of each mode was discussed. In the branching function we confirmed a high appearance rate of conditional branches and multi-way branches based on tag values.

1 Introduction

We, an R & D team of the Fifth Generation Computer System project in Japan, have developed a Personal Sequential Inference machine (PSI) pursuing high-speed execution of an extended Prolog language, larger memory capacity, and an improved environment for interactive program development. The design philosophy, machine architecture, and hardware configuration of the PSI have already been reported.[2,3,4]

This paper reports a result of performance and architectural evaluation of the PSI machine. In the evaluation, we measured the execution time of Prolog application programs and the dynamic characteristics of cache memory, register file, and branching hardware that were introduced for high-speed execution of Prolog programs. Measurement data is reported and effects of each dedicated hardware component are discussed referring to the data. The evaluation was undertaken with the goal of improving and redesigning the PSI. The measurement result also includes interesting data concerning the dynamic memory access characteristic of Prolog application programs.

2 PSI Architecture

2.1 Execution Method and Machine Architecture

The PSI is a high-level language machine dedicated for direct execution of the predicate logic language KL0, which is an extended version of Prolog.

In the PSI, a microprogrammed interpreter interprets and executes machine-resident expressions of KL0 programs (instruction code).[4] For the high-performance interpretive execution, PSI has several hardware supports described in the next subsection. A word format of the PSI consists of an 8-bit tag part and a 32-bit data part. In the instruction code, each atom, predicate name and variable is mainly expressed in a word containing the corresponding tags. If arguments for a predicate don't require one-word length expressions, up to four 8-bit arguments are packed into one word in order to reduce memory consumption.

Execution method of the microprogrammed interpreter is similar to that of the DEC-10 Prolog [6] except for extended control functions [7,9] and number of stacks. That is, four stacks such as local, global, control and trail stacks are used. The local and control stacks are separated in order to make inner clause OR operations efficient. The local stack is an area for local variables. The global stack is an area for variables appearing in compound terms. The trail stack contains address information to make variables unbound for backtracking. The control stack contains 10-word control frames as either environment frame for program execution continuation or choice point frame for backtracking. Control information for the current execution is held in a

*Fourth Research Laboratory, Institute for New Generation Computer Technology, 4-28, Mita 1-Chome, Minato-ku, Tokyo 108, JAPAN.

†Information Processing Dept., Information Systems and Electronics Development Laboratory, Mitsubishi Electric Corporation.

‡New Product Development Dept., Computer Works, Mitsubishi Electric Corporation, 325, Kamimachiya, Kamakura city, Kanagawa 247, JAPAN

register file called work file (WF) and saved to the control stack as necessary.

The four stacks are allocated to independent logical address spaces. Instruction codes and heap vectors (rewritable data structures) are stored in an area called the heap area, which is also an independent logical space. The PSI supports concurrent execution of multiple processes (programs) such as user processes and interrupt handling processes. The heap area is shared by all these programs, while stack areas for each program are allocated to independent logical spaces. We call this independent logical address space simply an area. In order to allocate physical memory pieces to the each area a hardware address translation table is supported.[2]

The PSI CPU contains a sequence control unit, a data processing unit, a memory unit which includes a cache and an address translation unit, and an I/O bus interface. The data processing unit has two source data buses and a destination bus which an ALU, a register file, memory interface registers, etc. are connected with. These hardware components are controlled with a microprogram. Microprogram execution and next microinstruction fetch are done in parallel. A console processor is connected to the CPU through a console bus for the maintenance, initialization, debugging and measurements for the evaluation.[2]

2.2 Hardware Features for High-Speed Execution

The PSI has the following features supporting high-speed execution of KL0 [2] ;

1. Cache memory for high-speed access to stack and heap areas.
2. Large-capacity multiple-functional register file (WF: work file) for the interpreter optimization.
3. Multi-way branching function based on tag values, and versatile conditional branching function.
4. Microprogrammed parallel control function (almost horizontal microprogram control: 64 bits per instruction).

The specifications of the cache memory are (a) 8K words capacity, (b) two-set set associative method, (c) store-in (write-back) method, (d) 200n sec access time for hitting and 800n sec for missing, (e) four-word block size, (f) four-word block transfer between the main memory taking 800n sec, (g) specialized Write-stack command which eliminates block read-in on a writing mishit; which is used for the continuous push operation to top of the stack.

The register file called the work file (WF) has 1K-word capacity and can execute a read, and a result writing operation within a microinstruction cycle. The first 16 words of WF have dual ports access capability, and the first 64 words and the last 64-word constant storage area are direct addressable from a microinstruction. The two address registers called WFAR1 and WFAR2 can be used for indirect addressing. They have automatic increment and decrement functions. Base-relative addresses are generated by two methods; one combining the lowest 5 bits of a data register (PDR or CDR) to a base register, and the other combining 5-bit addresses specified by microinstructions to another base register. The interpreter uses the tail recursion optimization method for dynamic optimization.[7] The method is implemented with reserving a pair of buffer area on the work file (called the frame buffer). The buffer caches the local variables for the current execution. Two buffers are used alternately when no local frame have to be saved into the local stack, and local stack accesses are reduced into the work file access. These frame buffer areas in the work file can be accessed by either indirect addressing through WFAR1 or base-relative addressing through PDR or CDR.

2.3 Target Performance

The PSI CPU uses mainly high-speed Schottky TTL MSIs available in the market. The microinstruction cycle time is 200 nanoseconds. Its target execution speed is 30K LIPS (Logical Inference per Second), comparable to the performance of the DEC-10 Prolog compiler on the DEC-2060.

3 Performance Evaluation

3.1 Program Execution Time

We executed some benchmark programs on the PSI and on the DEC-2060 with DEC-10 Prolog Compiler and measured the execution time. Table 1 shows the results and ratios between both measurements. For the DEC-10 Prolog compiler, mode and fast-code declaration were used for the compilation. The execution time of these codes were then measured several times with "statistics" predicate, and the results averaged. Measurements of the PSI used a 1-millisecond built-in timer of the CPU.

Benchmark programs (1) through (10) in Table 1 are a part of programs presented in the first Prolog contest of Japan.[8] They are all small-scale programs that contain frequent list processing. Sample programs (11) through (19) are larger-scale programs for practical use. Among them, BUP and LCP are parsers using different methods for natural language processing; HARMONIZER is a music generation system that attaches harmonies to melodies according to musical knowledge. Number of source program lines of these programs are approximately 300, 1500, 700 lines each. And each of these uses unifications of structural data and backtracking. Especially BUP treats structures larger than eight elements and nested structures. And HARMONIZER uses frequent backtracking. Program (11) and higher were also used in the hardware evaluation described in the next

Table 1: Execution time of benchmark programs on PSI and DEC-2060.

	programs	PSI(msec)	DEC(msec)	DEC/PSI
(1)	nreverse (30)	13.6	9.48	0.70
(2)	quick sort (50)	15.2	14.6	0.96
(3)	tree traversing	51.7	61.1	1.18
(4)	lisp (tarai3)	4024	4360	1.08
(5)	lisp (fib10)	369	402	1.09
(6)	lisp (nreverse)	173	194	1.12
(7)	8 queens (1)	96.9	97.5	1.01
(8)	8 queens (all)	1570	1580	1.01
(9)	reverse function	38.2	41.7	1.09
(10)	slow reverse (6)	99.4	89.0	0.90
(11)	BUP-1	43	52	1.21
(12)	BUP-2	139	194	1.40
(13)	BUP-3	309	424	1.37
(14)	harmonizer-1	657	1040	1.58
(15)	harmonizer-2	1879	2670	1.42
(16)	harmonizer-3	24119	31390	1.30
(17)	LCP-1	379	295	0.78
(18)	LCP-2	1387	1071	0.77
(19)	LCP-3	2130	1656	0.78

section.

Table 1 shows that the PSI has achieved a target performance comparable to that of the DEC-10 Prolog compiler. Observing the Table 1, it is found that superiority of DEC or PSI depends on the program. DEC tends to be faster than PSI for the program execution like (1) NREVERSE, which processes simple list data and is optimized effectively by the compiler. For example, the compiler can remove the nondeterminacy applying the close indexing method, and can optimize the code for list unification. Oppositely PSI tends to be faster than DEC for executing programs like (11) to (16), which have much unification between structural data and involve frequent backtracking. In other words, these programs require much run-time processing. One reason for this may be, the PSI requires more execution management information to be stacked, which in turn requires more overhead for basic processing on simple programs than DEC. Conversely, PSI is relatively faster when backtracking and unifications between structures appear which are processed completely by the microprogram.

Generally speaking, PSI is suitable at executing application programs which require much run-time processing. However, DEC executes the program LCP faster than PSI, even though LCP processed structural data. One reason may be the developer of LCP, Mr. F. Pereira, is also one of the developers of the DEC-10 Prolog processing system, giving him thorough knowledge of the system's advantages and disadvantages (for example, performance of the structure unification falls down when number of structure elements exceed certain number). To analyze in precise, further research is required.

3.2 Dynamic Characteristics of the Interpreter

We measured the dynamic characteristics of the microprogrammed interpreter when practical-scale application programs were executed. The purpose was to collect statistical data for improving and redesigning the system. Table 2 shows the execution step ratios of each component module of the interpreter. For the sample programs, the window system which is a component of the PSI operating system, and a game program called 8 PUZZLE were added. WINDOW treats few unifications of structure data and less backtracking. So to speak, WINDOW rarely uses the functions of Prolog. 8 PUZZLE is a search problem and contains much backtracking.

Rate of built-in predicates call to the total predicate call were 82% for window and 65% for BUP, which are not included in the table. They are much higher than the user defined predicate call. However the rate of microprogram execution steps for built-in predicates (built) to the total execution steps were 26.2% and 19.2% for each program, seen in the table. These values are much less than the rate of calls. It means that a lot of time is spent for execution control (management of call and return for user defined predicates) not for executing built-in predicates. This may result from the previously mentioned situation of relatively large management information in KL0 (ten-word frame) requiring many processing steps. Table 2 also shows that getting arguments (get_arg) for built-in predicates is also time-consuming. These conditions may be a useful focus for efforts to improve the processing system.

On the other hand, observing the column "unify", values for BUP and HARMONIZER are large, demonstrating that these programs execute much unification. That is, the unification is one of the targets for performance improvement. Concerning the performance of the unification function alone, PSI was measured better than DEC.[6] To improve the unification performance more, new instruction set should be introduced which is suitable for a compiler to optimize unification code.

4 Hardware Evaluation

4.1 Measurement Method

This section reports our measurement method of the dynamic characteristics of the hardware executing sample programs. Measurements were intended for determining whether or not the hardware needed to be redesigned. Dynamic characteristics of the cache memory, work file, and branching functions were measured.

Table 2: Execution step ratios of each component module of the firmware interpreter (%).

programs		control	unify	trail	get-arg	cut	built
(1)	window	31.1	17.1	2.0	13.6	10.0	26.2
(2)	8 puzzle	27.5	11.0	7.5	22.7	0	31.3
(3)	BUP	22.3	43.0	4.7	5.2	5.6	19.2
(4)	harmonizer	25.5	46.4	5.4	7.3	4.0	11.0

Table 3: Execution rate of each cache command in the total microprogram execution steps (%).

programs		read	write-stack	write	write-total	total
(1)	window-1	15.2	3.5	1.2	4.7	19.9
(2)	window-2	15.2	3.0	1.1	4.1	19.7
(3)	window-3	17.6	3.9	1.4	5.3	22.8
(4)	8 puzzle	9.9	3.2	2.8	6.1	16.0
(5)	BUP	15.6	3.5	2.2	5.7	21.3
(6)	harmonizer	15.3	4.6	2.2	6.8	22.1
(7)	LCP	17.0	3.9	2.2	6.1	23.1

Table 4: Access frequency of each memory area (segment) (%).

programs		heap	global stack	local stack	control stack	trail stack
(1)	window-1	49.6	4.6	16.5	26.7	2.6
(2)	window-2	56.6	4.4	12.7	26.3	0.1
(3)	window-3	52.7	6.2	12.1	28.2	0.8
(4)	8 puzzle	31.3	14.3	33.9	14.1	6.4
(5)	BUP	39.0	29.9	17.3	12.0	1.8
(6)	harmonizer	35.2	17.7	30.3	12.8	3.8
(7)	LCP	44.7	22.3	14.1	17.4	1.4

The sample programs mentioned in the previous section were used, and several tools were prepared for data collection and hardware evaluation. For data collection, a simple interpreter system called COLLECT was installed in the console processor of the PSI. This system was made to execute command chains for CPU activation and data collection. Single steps or continuous steps up to a breakpoint were executed repeatedly, and microinstruction addresses and the contents of registers or memory were dumped onto a flexible disk each time the CPU stopped. For analysis of the work file and branching circuit, we built a tool called MAP and analyzed microinstruction patterns. Using an address pattern of microinstructions traced by COLLECT, MAP counts the number of specific pattern appears in a specific microinstruction field. For analyzing the dynamic characteristics of cache memory, we also made a cache memory simulator called PMMS. Hit ratios and its variations according to the cache memory size were obtained by PMMS with cache command patterns and memory addresses collected by COLLECT.

4.2 Cache Memory

We collected data concerning the frequency of cache memory access and cache hit ratios. We used these data to evaluate cache memory of PSI which decreased access time both to the stack and heap areas.

Table 3 lists the appearance frequency of each cache command. It shows that 16 to 23.1% of all microinstruction steps include cache commands; in other words, about one in every five microinstruction steps is a request for memory access. The ratio between Read and Write commands is approximately 3 and 1, i.e., Read commands appear more often than write commands. The Write Stack command accounts for 50 to 75% of the total Write commands, indicating that the command introduced for stacking data is frequently used.

Table 4 shows the frequencies of accesses to areas. When the accesses are grouped into those to the heap areas and those to the four stacks, accesses to the heap area account for 30 to 55% of the total. Accesses to the heap area are mainly made to fetch instruction codes. Only the program WINDOW uses data of the heap vector type, and thus the frequency of its access to the heap area increases. It is also found that the frequency of instruction fetch varies depending on the programs.

The frequencies of accesses to the global, local, and control stacks is also program dependent. The global stack is frequently accessed if a program processes many structured data, as is the local stack if a program contains many variables as arguments (other than structured data elements) or is the control stack if a program calls many predicates for few arguments. Access frequencies to the trail stack are low, 6.4% at highest.

Table 5 shows cache hit ratios for areas. Most of hit ratios are higher than 96% except for WINDOWs. Programs (5) through (7) had especially high hit ratios, while they are practical-scale application programs effectively us-

Table 5: Cache hit ratios of each memory area (segment) (%).

programs		heap	global stack	local stack	control stack	trail stack	total
(1)	window-1	94.1	92.8	98.9	99.4	99.6	96.4
(2)	window-2	87.2	90.0	98.5	99.3	95.2	91.9
(3)	window-3	84.5	92.8	97.4	98.6	98.7	90.7
(4)	8 puzzle	99.2	99.4	99.6	99.2	97.7	99.3
(5)	BUP	98.2	96.8	99.0	98.2	99.7	98.0
(6)	harmonizer	97.5	98.4	99.4	98.2	97.9	98.4
(7)	LCP	96.6	93.8	99.2	99.1	98.6	96.2

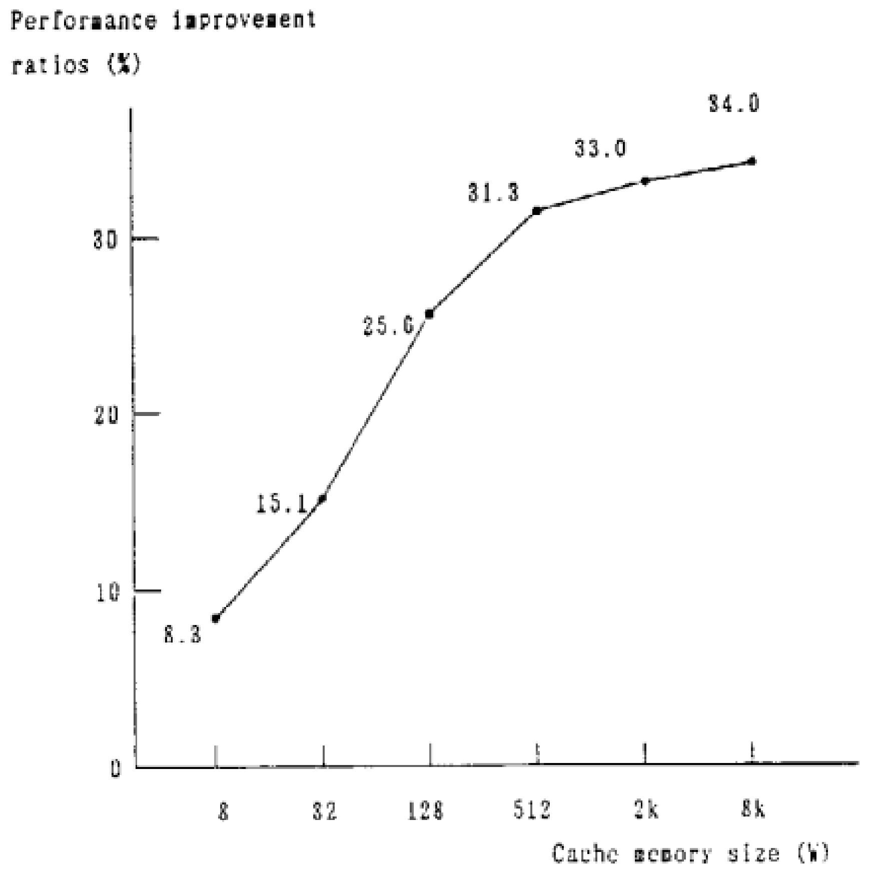

Figure 1: Performance improvement ratios against the cache memory size.

ing backtrack and unification functions. This means that the memory access locality for Prolog program execution is high enough and the cache memory size of PSI is sufficient. The reason why the hit ratios for WINDOW-2 and WINDOW-3 are low is considered that these program executions contained process switching for I/O services several times, and that the locality of instruction codes decreased because of the object oriented features of the ESP, a system description language of PSI, in which frequent predicate call across "the class" occurred. Whether or not object-oriented programming decreases cache hit ratios, it may have to be determined by further evaluation.

For the next, the capacity of cache memory and the number of cache sets are investigated. Data was obtained by the cache memory simulator which simulated various specifications of cache memory using a trace information of program WINDOW. Figure 1 shows performance improvement ratios obtained by changing the cache memory capacity from 8 words to 8K words. Other specifications are same with the cache memory of the PSI.

Definition is:

Performance improvement ratio = (Tnc/Tc) - 1 x 100

Tnc: Execution time measured when cache memory does not exist

Tc: Execution time measured when cache memory exists

As shown in Figure 1, the improvement ratio saturates near the capacity of 512 words. This means that the 8K-word capacity of cache memory can be reduced to some extent.

On the other hand, the direct mapping cache (single set cache) is more desirable than the two sets cache concerning the design and hardware cost. Therefore, we investigated performance improvement ratios when using two 4K-word sets and one 4K-word set. The ratios obtained with one set cache for WINDOW, 8PUZZLE, or BUP was only 3% lower than those obtained with two sets for the same programs. This value may be useful for judging the trade-off between cost and performance.

To evaluate the store-in method, the performance improvement ratio of this method was compared with that of the store-through method using the cache memory simulator. Store-in was 8% higher than store-through, confirming the effectiveness of the store-in method.

4.3 Work File

We measured the appearance frequency of each work file (WF) access mode for sample programs execution. Measurement employed the microinstruction pattern analysis tool MAP. Table 6 shows the measurement results obtained when BUP was executed. Close results to the BUP were obtained when other programs were executed.

As shown in the "total" row of Table 6, the frequency of access to the WF was 56.4% for being specified by microprogram Source 1 (controlling ALU input-1) field, 29.1% by Source 2 (controlling ALU input-2) field, and 36.6% by Destination field (controlling ALU output bus). WF is used as ALU input-1 in more than half of microprogram execution steps, while a part of Source 2 and Destination access were performed with Source 1 access concurrently and the rest were performed independently.

Table 6: Dynamic frequency of the Work File (register file) access mode (%).

access mode		source 1 (ALU in-1)	source 2 (ALU in-2)	destination (ALU out)
(1)	WF00-0F	12.2†/ 6.9‡	100†/29.1‡	33.0†/12.1‡
(2)	WF10-3F	58.5/33.0	—	63.6/23.3
(3)	Constant	23.0/13.0	—	—
(4)	@PDR/CDR	1.3/ 0.8	—	0.3/ 0.1
(5)	@WFAR1	4.6/ 2.6	—	2.8/ 1.0
(6)	@WFAR2	0.07/ 0.04	—	0.3/ 0.1
(7)	@WFCBR	0.3/ 0.2	—	0.0/ 0.0
	total	100/56.4	100/29.1	100/36.6

†Rate in the total Work File access counts.
‡Rate in the total microprogram execution steps.

The addressing modes (1) through (3) are variations of the direct addressing. These modes were used by 90% or more of all accesses to the WF. While one third of the 64-bit length of a microinstruction word is used as a direct addressing field for WF, the frequency of the use of this modes confirms the usefulness of this field.

Functions (4) and (7) for base-relative addressing and functions (5) and (6) for indirect addressing are evaluated below. The base-relative addressing function (4), which uses PDR or CDR as an offset from the base, was used less frequently than expected. This function was most frequently used to execute WINDOW (not shown in the table), but the highest frequency was only 3.3% when PDR was used and 1.7% when CDR was used. If each packed argument(8-bit) in a word can be also used as an offset from the base for pointing variables on the WF (on the frame buffer), the use rate of the function may increase somewhat. The indirect addressing function (5) through WFAR1 is used to access the local frame buffer, as is function (4). 90% or more of indirect addressing successfully use the automatic increment function. Function (6) is used to access the trail buffer, [1,4] and function (7) is used for general purposes. The use rates of both were so low that the buffering of trail stack and supporting functions (6) and (7) may have to be reconsidered.

More than 99% of all accesses to the WF were found as accesses to directly addressable 128-word areas and two 64-word local frame buffers. In this situation, reducing the work file capacity much from 1K-word may decrease hardware cost with little effect on performance.

Table 7: Dynamic frequency of the branch operations in the microprogram execution steps (%).

operation		BUP	window	8 puzzle
Type1				
(1)	no operation	7.2	6.7	4.8
(2)	if (cond) then	16.0	16.5	12.1
(3)	if (not(cond)) then	19.2	17.0	20.3
(4)	if tag(src2) then	2.7	5.2	3.1
(5)	case (tag(n,P/CDR))	10.9	8.6	9.1
(6)	case (irn)	2.8	4.6	4.9
(7)	case (ir-opcode)	0.5	1.4	1.5
(8)	goto	3.7	1.4	2.7
(9)	gosub	4.0	5.7	6.5
(10)	return	3.8	5.4	6.5
(11)	load-jr	0.8	0.4	0.7
(12)	goto &jr	1.4	0.6	0.7
Type2				
(13)	no operation	9.6	7.8	7.7
(14)	goto	10.9	11.7	15.2
Type3				
(15)	no operation	6.5	7.0	4.2
(16)	goto &jr	0.0	0.04	0.05

4.4 Branch Functions

Table 7 shows the dynamic frequencies of each operations appeared in microprogram branch field for three sample programs execution. The frequencies were measured with the microinstruction pattern analysis tool. In the "operation" column, mnemonics of branch instructions are shown which are grouped into three instruction types. Each group includes the No Operation function. The total appearance frequencies of all the operations, (1) to (16), is 100%, and that of all branch functions can be obtained by subtracting the appearance of No Operation functions, (1), (13) and (15), from 100%. This total is very high ranged from 77 to 83%. That is, around 80% of all the microinstruction steps contain branch operations. In these steps, it was also measured that approximately 50% were executed with data manipulation and approximately 30% without data manipulation.

In Table 7, 64 types of conditional branches are grouped into functions (2) and (3). Conditional branch (4) is based on comparison to a given tag value. The total of requests for these branches accounted for 35 to 39% of all steps. The need to reinforce conditional branch functions is thus confirmed.

The tag dispatch function, a multi-way branch function based on tag values of the PDR and CDR registers, is shown as (5). The multi-way branch function using packed operands tags (3-bit tags in 8-bit packed operand) is shown as (6). The total requests for these functions accounted for 13 to 14% of all steps. In other words, every eighth step is a request for multi-way branch. Therefore, the functions are also very useful.

The indirect branching functions (12) and (16) via JR (Jump Register) [2] have low frequencies. JR is used as a loop counter rather than an address register, its use accounted for 9 to 12% of all steps. Therefore, preparing a dedicated loop counter may be a better strategy.

5 Conclusion

We measured and evaluated the performance of the PSI and component hardware added for high-speed execution of Prolog programs. We also presented measured values of dynamic memory access characteristics.

We confirmed similarities in performances of the PSI and DEC-10 Prolog compiler on DEC-2060. However, DEC is faster than PSI when executing programs that can be optimized well at compile time, while PSI is faster than DEC when executing application programs requiring much run-time processing. This characteristic can be considered as a feature of a microprogrammed interpreter.

The cache memory hit ratio for an application program was measured as high as 96%. Therefore, the cache memory is evaluated as sufficiently effective, and also the store-in method as effective, and the 8K-word capacity as reducible. The measurement of dynamic memory access characteristics demonstrated that the appearance ratio of Read and Write commands is 3 and 1, that every fifth microinstruction is a request for memory access, and that access to the heap area (mainly the instruction fetch) accounts for 30 to 50% of all microinstruction steps.

For the work file evaluation, 90% or more of work file access use direct addressing. However, the need for an improvement of base-relative addressing was confirmed, as was the possibility of reducing the 1K-word capacity of the work file. The multi-way branch functions based on tag values and conditional branch functions were evaluated as sufficiently effective.

Based on the above evaluation results, we have been redesigning the PSI hardware and improving the instruction code suitable for the compile time optimization. Results will be reported in the other paper.[8]

Acknowledgment

We would like to thank the chief of our fourth laboratory, Dr. S. Uchida, for his advice and the Director of ICOT, Dr. K. Fuchi, for giving us this opportunity to do research.

References

[1] Nakajima,K., Nakashima,H., Yokota,M., Taki,K., Uchida,S., Nishikawa,H., Yamamoto,A. and Mitsui,M. : Evaluation of PSI micro-interpreter, Proc. of Compcon Spring 86, pp.173-177 (1986).

[2] Taki,K., Yokota,M., Yamamoto,A., Nishikawa,H., Uchida,S., Nakashima,H. and Mitsuishi,A. : Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI), Proc. of the INternational Conference on Fifth Generation Computer Systems 1984, pp.398-409, Tokyo, (1984).

[3] Yokota,M., Yamamoto,A., Taki,K., Nishikawa,H. and Uchida,S. : The Design and Implementation of a Personal Sequential Inference Machine PSI, New Generation Computing, Vol.1, No.2, pp.125-144, Ohmsha, (1983).

[4] Yokota,M., Yamamoto,A., Taki,K., Nishikawa,H., Uchida,S., Nakajima,K. and Mitsui,M. : A Microprogrammed Interpreter for the Personal Sequential Inference Machine, Proc. of the International Conference on Fifth Generation Computer Systems 1984, pp.410-418, Tokyo, (1984).

[5] Takagi,S., et al. : Introducing extended control structures for Prolog, Proc. of 26th Inter-domestic conference, Information Processing Society of Japan, No.4D-11, (1983), In Japanese.

[6] Warren,D.H.D. : Implementing Prolog -Compiling Predicate Logic Programs, Vol.1,2, D.A.I. Research Report No.39,40, Dept. of Artificial Intelligence, Univ. of Edinburgh, (1977).

[7] Warren,D.H.D. : An Improved Prolog Implementation which optimizes Tail Recursion, Proc. of the Logic Programming Workshop, Hungary (July 1980).

[8] Okuno,H. : The Report of The Third Lisp Contest and The First Prolog Contest, Proc. of the research working group, SYM33-4, Information Processing Society of Japan, (1985), In Japanese.

[9] Nakashima,H. and Nakajima,K. : Hardware Architecture of the Sequential Inference Machine : PSI-II, Proc. of Fourth Symposium on Logic Programming, San Fransisco, (1987).

Association for Computing Machinery
11 West 42nd Street

Asplos ii

Title of work:

Author(s):

Kazuo Taki
4th Research Laboratory
Institute for New Generation Computer Tech.
4-28, Mita 1-Chome, Minato-ku
Tokyo 108 JAPAN

Air mail

A. TRANSFER AGREEMENT

Copyright to the above work (including without limitation, the right to publish the work in whole or in part in any and all forms and media, now or hereafter known) is hereby transferred to the ACM (for U.S. Government work, to the extent transferable*) effective as of the date of this agreement on the understanding that the work has been accepted for publication by ACM.

However, each of the authors reserve the following:

(1) All proprietary rights other than copyright (and the publication rights transferred to ACM), such as patent rights.

(2) The right to use in future works of the author's own, such as articles or books, all or part of this paper with acknowledgment to ACM, and also with prior notice to ACM if the use is for direct commercial advantage.

(A copy of this form must be signed by all authors or, in the case of a "work made for hire," by the employer and must be received by the Association for Computing Machinery — See Box C — before processing of the manuscript for publication can be completed. This form may be duplicated for signing by co-authors. Authors should understand that consistent with ACM's policy of encouraging dissemination of information each published paper will appear with the following notice:

"Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.")

Signature	Kazuo Taki	Signature	[illegible]
Print Name	Kazuo Taki	Print Name	Katsuto Nakajima
Title, if not Author		Title, if not Author	
Date	25 - June - 1987	Date	25 - June - 1987

B.* DECLARATION FOR U.S. GOVERNMENT WORK

This certifies that the above author(s) wrote the paper, (a.) as a part of work as U.S. Government employees or, (b.) as other noncopyrightable Government work.

Signature

Agency

Title, if not author

Date signed

C. WHERE TO RETURN FORM

Author: Please return this form to:

Lee Blue
Computer Society of IEEE
1730 Massachusettes Avenue, NW
Washington, DC 20036
(202) 371-1012

D. NAME AND DATE OF CONFERENCE

ASPLOS '87
Palo Alto, California
October 5-8, 1987

Association for Computing Machinery
11 West 42nd Street

Asplos ii

Title of work:	
Author(s):	

Kazuo Toki
4th Research Laboratory
Institute for New Generation Computer Tech.
4-28, Mita 1-Chome, Minato-ku
Tokyo 108 JAPAN

Air mail

A. TRANSFER AGREEMENT

Copyright to the above work (including without limitation, the right to publish the work in whole or in part in any and all forms and media, now or hereafter known) is hereby transferred to the ACM (for U.S. Government work, to the extent transferable*) effective as of the date of this agreement on the understanding that the work has been accepted for publication by ACM.

However, each of the authors reserve the following:

(1) All proprietary rights other than copyright (and the publication rights transferred to ACM), such as patent rights.

(2) The right to use in future works of the author's own, such as articles or books, all or part of this paper with acknowledgment to ACM, and also with prior notice to ACM if the use is for direct commercial advantage.

(A copy of this form must be signed by all authors or, in the case of a "work made for hire," by the employer and must be received by the Association for Computing Machinery — See Box C — before processing of the manuscript for publication can be completed. This form may be duplicated for signing by co-authors. Authors should understand that consistent with ACM's policy of encouraging dissemination of information each published paper will appear with the following notice:

"Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.")

Signature	Morihiro Ikeda	Signature	Hiroshi Nakashima
Print Name	Morihiro Ikeda	Print Name	Hiroshi Nakashima
Title, if not Author		Title, if not Author	
Date	22 - June - 1987	Date	22 - June - 1987

B.* DECLARATION FOR U.S. GOVERNMENT WORK

This certifies that the above author(s) wrote the paper, (a.) as a part of work as U.S. Government employees or, (b.) as other noncopyrightable Government work.

Signature	
Agency	
Title, if not author	
Date signed	

C. WHERE TO RETURN FORM

Author: Please return this form to:

Lee Blue
Computer Society of IEEE
1730 Massachusettes Avenue, NW
Washington, DC 20036
(202) 371-1012

D. NAME AND DATE OF CONFERENCE

ASPLOS '87
Palo Alto, California
October 5-8, 1987

acm Association for Computing Machinery

11 WEST 42nd STREET
NEW YORK, NY 10036
(212) 869-7440
Telex: 421686

TO: Authors Submitting Papers for the Proceedings of ACM Sponsored Conferences

FROM: ACM Director of Publications

SUBJECT: ACM Copyright Procedures

Thank you for submitting a paper for the conference. ACM's publications are read throughout the world, and we must deal with requests for reprinting, translating, anthologizing, and other actions.

It is the policy of ACM to own the copyrights on its technical publications to protect the interest of ACM, its authors, their employers, and at the same time to facilitate the appropriate reuse of this material by others.

The United States Copyright Law requires that the transfer of copyright of each contribution from the author to ACM be confirmed in writing. It is necessary that all authors sign either Part A or Part B of the copyright form and return it with the manuscript to the address on the form.

If you are employed and you prepared your paper as a part of your job, the rights to your paper may initially rest with your employer. In that case, when you sign the copyright transfer form, we assume you are authorized to do so by your employer. If not, it should be signed by someone so authorized.

For jointly authored papers, an original signature is required from each co-author. For this purpose, the form may be duplicated before signing.

Authors who are U.S. Government employees and/or whose papers are not copyrightable as part of certain Government contract work, are not required to sign Part A. but any co-authors outside the Government contract are.

Part B of the form is to be used instead of Part A only if all authors are U.S. Government employees and they prepared the paper as part of their job, or the work is an uncopyrightable product of a Government contract.

ACM authors have all rights scientific authors have historically enjoyed, including the right to present orally the submitted or similar material in any form; the right to make minor reuse, with credit, in publications (and major reuse in works of the authors' own with notice and with credit to ACM); the right to republish with notice and credit to ACM, in works published by the employer or for the employer's internal purposes, the right to reproduce for peer review in reasonable quantities; and all proprietary rights other than copyright.

Although it is not part of ACM's copyright policy to grant to authors or their organizations the sole right to approve permissions for republishing by third parties, ACM always seeks the approval of its authors in weighting such requests. This is done as a matter of personal professional courtesy.