

Makoto NAGAO Jun-ichi TSUJII Katsuto NAKAJIMA Kei MITAMURA Hachidai ITO

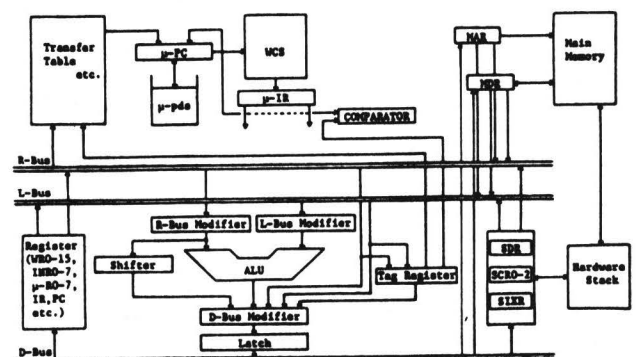
Department of Electrical Engineering
Faculty of Engineering
Kyoto University
Kyoto, Japan

1. INTRODUCTION

On the other hand, for natural language understanding systems the more flexible control structures are required than the recursive one the ordinary LISP systems provide. The idea of the 'society of experts'[HEWITT 1977] is also very important in natural language understanding systems. Implementing such a co-routine environment by an ordinary LISP may decrease the efficiency considerably. It is desirable that the hardware system supports such mechanisms.

with various hardwares such as a specially designed hardware stack. We have completed the hardware of NK3 and implemented the interpreter of LISP1.5 on it. We are now developing more sophisticated LISP interpreter, compiler, and micro coded instructions for the machine step by step in evolutionary approaches. We have measured the performance of the machine from various viewpoints: how often the data transfer between the hardware stack and the main memory occurs, how frequent each micro coded instruction is used etc. Based on these measurements, we are improving the constructions of the interpreter, compiler and micro coded instructions.

Fig.1 shows the overall architecture of the LISP



625

machine NK3. NK3 is a micro-program machine. Micro-programs are stored in WCS(writable control store), and the macro-programs and LISP cells are stored in the main memory. The machine has three data buses, ordinary registers, ALU and some other functions. Special care is taken in the architecture to realize rapid transfer of the pointer information in the system. Automatic management of the stack areas and the guarantee of the high speed, flexible automatic pushing down and popping up operations are the other important design issues of the system. Some typical features of NK3 are the followings.

(i) Writable Control Store (4 kw, 42 bit/word)

This contains micro-programs of most built-in functions and most parts of the interpreter.

(ii) Main Memory (512kB, 800nsec)

The main memory is of the mini-computer INTERDATA 8/32 and is shared by NK3. Because of this memory sharing the access speed to the memory from NK3 takes about 2 μ seconds, which is very slow. The input/output operations of NK3 are performed by INTERDATA 8/32. Since the OS is also shared, NK3 does not need to have any general facilities such as the file management.

(iii) Hardware Stack (16 words, 32 bits/word)

The top part of the stack is stored in these registers. The popping-up and pushing-down operations are generally performed in ordinary LISP programs alternately. Long consecutive popping (pushing) operations are rare. In such a situation, the usual double buffering technique is inadequate. It causes frequent data transfer. To avoid this, we have adopted quadruple buffering for the hardware stack. The hardware stack is divided into four blocks, each of which consists of four words. By transferring data from/to the main memory asynchronously with the CPU by the unit of four words, the hardware stack is guaranteed to contain always at least four words of the stack and four word spaces for the push-down information. Special circuits are provided to detect the errors such as stack overflow and underflow. The data transfer is done by burst DMA mode independently with the LISP program. Any part of the main memory can be assigned to be the stack area, and any number of areas can be reserved for different stacks in the memory to realize co-routine environments. Some special instructions are prepared to switch the active stack from the current one to the other etc. Because frequent switching among stacks is expected to occur, it is not reasonable to have a larger hardware stack than the present one. It will cause a data transfer of large amount, when the active stack is changed.

We have measured how often the consecutive popping and pushing operations occur in the usual execution, and how often the data transfer is performed between the hardware stack and the memory (See Table 1).

	push-down (=pop-up)	PUSH-DMA	POP-DMA	DMA-RATE
TPU-1	196,240	7,470	8,009	0.158

$$\text{DMA-RATE} = \frac{4 * (\text{PUSH-DMA} + \text{POP-DMA})}{\text{push-down} + \text{pop-up}}$$

Table 1. Stack Operations and Data Transfer

The bench mark problem here is the theorem prover based on 'resolution principle' [Chang 1973]. From this table, one can see more than 84% of stack operations are effectively carried out without any data transfer.

(iv) Transfer Table (1 kw, 15 bit/word)

The interpreter frequently checks the data types of the cells and determines the jump position in the program. The data types of cells are expressed in the tag parts of the memory words (Fig. 2). The Transfer Table is used by micro-programs to determine the jump position directly from the description of the tag parts.

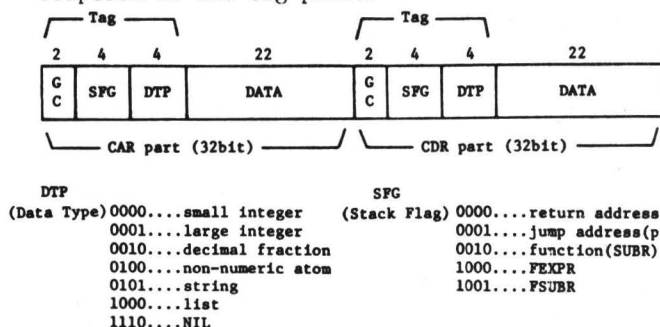


Fig. 2 Structure of LISP Cells

(v) Micro-Instructions

There are six types of micro-instructions. Those types are (1) Arithmetic and Logical operations, (2) Conditional Jumps, (3) Emit operations, (4) Transfer Table operations, (5) Patch operations, and (6) Shift operations. By Patch operations, we can patch arbitrary portions of L-Bus data and R-Bus data together. Combining this operation with the Shift operations enables us to manipulate the tag information efficiently. The format of the type 4 instruction is shown in Fig. 3.

3	4	8	5	3	10	5
011	ES	SL	EP	EL	TO	MSC

Fig. 3 Format of Type 4 Instruction

This instruction extracts the portion of the bus data (specified by ES - Extraction Source (R-Bus or L-Bus), EP - Extraction Position, and EL - Extraction Length), and adds it to TO (Table Offset).

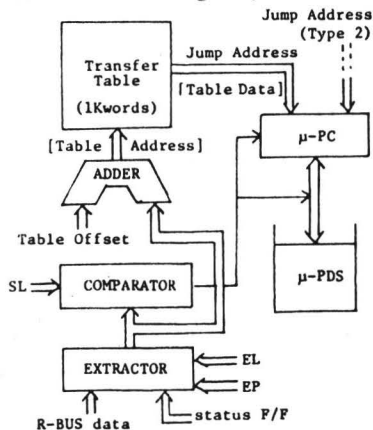


Fig. 4 Transfer Operation

3. HIGH LEVEL MACRO INSTRUCTIONS

NK3 is essentially a micro programmable computer which is equipped with various special hardwares. Therefore, in order to make the execution efficient, we should devise macro instructions which utilize these special hardwares well and are suitable for LISP operations. At present, most of built-in functions of LISP can be executed by single macro instructions. Moreover, some of them operates directly on the top of stack. Because of the hardware stack, these instructions are executed very fast.

A set of special macro instructions which facilitate the writing of LISP interpreters is provided. These are the parts of the interpreter which conceptually perform single tasks. For example, searching the value of a variable from the A-list, jumping to the appropriate locations in Eval or Apply according to the types of the arguments and so on are performed by single instructions. The instruction HASH is another example of high level macro instructions. The hashing operation on the atom's print-name is carried out by this instruction. It is very time consuming by the ordinary program to identify all the input character strings with the atoms already in the system. The HASH instruction performs this task very efficiently.

Some other special instructions have been devised to manipulate the hardware stack for implementing flexible control mechanisms directly. The instructions EXCH and INIT are the examples. The EXCH and INIT enables us to implement co-routine environments. Because NK3 is equipped with a single hardware stack, only one of the co-routines can be active at a time and the stack infor-

mation for the routine is stored in the hardware stack. The EXCH is used to exchange the content of the hardware stack, in order to activate another co-routine in place of the current one. The older context will be appropriately saved. The instruction INIT is for invoking a new co-routine. It initializes a stack area and reserves a certain amount of memory area for that routine.

4. PERFORMANCE EVALUATION

We have written a LISP interpreter for LISP 1.5 by using the macro instructions described in 3, and have measured the performance and the effectiveness of these instructions. We are still seeking much more efficient set of macro instructions for LISP. Up to present we have experienced a drastic improvement of the execution times, for example two to three times, by setting up a good instruction set. On the other hand, another trial is in progress, which makes the macro instructions as big as possible in their functions. The limit in this direction is that the stack operations cannot be realized by micro programs. Because of this weak point, the interpreter as a whole cannot be realized by micro instructions only.

We also compared the execution speed of NK3 with some other LISP systems on big and small computers. The bench mark problems were a sorting program, a theorem prover, and some other typical LISP programs. The performance of NK3 might be judged as : it has a half of the power of very large machines (i.e. M190), and has several times of the power of small computer versions of LISP. If NK3 can use the main memory exclusively without sharing with INTERDATA 8/32, and if the stack mechanism can be utilized freely from micro-programs, the execution speed will improve drastically. These may be the main improvement to be taken soon.

REFERENCES

- [CHANG 1973] C.Cang, R.Lee, "Symbolic Logic and Mechanical Theorem Proving," Academic Press, 1973
- [HEWITT 1977] C.Hewitt, "Viewing Control Structures as Pattern of Passing Messages," Artificial Intelligence, vol. 8, No. 3, 1977
- [MOSES 1975] J. Moses et.al., "Proposal for Personal Computer Capable for Executing Large LISP Programs," MIT, 1975
- [NAGAO 1975] M.Nagao, J.Tsujii, "PLATON - A New Programming Language for Natural Language Analysis," Proc. of 2nd USA-Japan Computer Conference, Tokyo, 1975