ISE-TR-80-16

HARDWARE ARRAY BOUND CHECKER ON TAGGED ARCHITECTURE

by

Kozo Itano

Tetsuo Ida

July 1, 1980

INSTITUTE
OF
INFORMATION SCIENCES AND ELECTRONICS

UNIVERSITY OF TSUKUBA

Hardware Array Bound Checker on Tagged Architecture

by

K.Itano [*] and T.Ida [**]

[*] Institute of Information Sciences and Electronics,
University of Tsukuba,
Sakura-mura, Niihari-gun, Ibaraki 305, Japan.

[**] Institute of Physical and Chemical Research,
Hirosawa, Wako-shi, Saitama 351, Japan.

Index terms

Array bound checker, tagged architecture, probabilistic checking.

ABSTRACT

A hardware scheme to check the correctness of array accesses in high speed is presented. The new scheme is based on a tagged architecture. A check is made whether the protection code of an array matches with the tag of the memory word. In order to check a large number of arrays efficiently by limited number of bits for the tag, a probabilistic check algorithm is introduced.

1. Introduction

Use of arrays is one of the most essential part of programming in scientific computations. Since arrays form a large percenage of variables used in the actual programming [1], arrays must be referenced correctly and with high efficiency.

Despite of the progress in hardware and software engineering, little effort has been made to systematically approach, especially from architectural view point, the troubles with address range violation during the access of arrays. We often encounter with an error message such as "memory protection fault!" or "instruction invalid!" as the result of an illegal access to the non-existent area or to the instruction area. Or even worse, no error is reported even if incorrect results are obtained due to erroneous accesses to unintended memory areas. This type of errors are difficult to locate. With commonly existing software, we are forced to resort to methods such as debug mode compilation sacrificing run-time speed or examination of linkage maps for finding the location where the error occurred.

Axiomatic approach to verifying the correctness of programs in array referencing can cope with limited class of programs [2]. Hence, the hardware checking scheme would contribute more easily and directly to the detection of the range violation during referencing arrays.

In this note, we propose a hardware array access checking scheme [3] to facilitate debugging process without sacrificing run-time speed. Our scheme is based on tagged architecture,

which was originally advocated to check data types at run time efficiently [4,5].

2. Basic scheme for array bound checker

2.1 Hardware error detection principle

Let M[0:n] be a memory space, and V[0:n] be a one-dimensional array mapped on M. Associated with V, we introduce t, protection code of V stored in each memory word onto which V is mapped as shown in Figure 1. In making access to the element of V by the address of M, a check is made whether the protection code of vector matches with the protection code t' of the memory word to which an access was made. If t=t', we conjecture that the access is legal. This conjecture is correct as long as:

(1) no two vectors are mapped to the same memory location, and
(2) unique protection code is assigned to each vector.

Validity of condition (1) lies on the programming language design. We assume that with a suitable language translation technique condition (1) holds true. Our goal is to provide hardware basis for condition (2).

2.2 Tagged architecture

Our idea is to encode protection code in the tag associated with each memory word. The protection code is checked at each memory access. In Figure 2 we give the design of a hardware array bound checker for a general register machine model. The essential part of the hardware is the tag handling mechanism. The processor contains tag registers which hold protection codes associate with the arrays. The memory system is equipped with tag memory which holds protection code associate with the array stored there. A hardware comparison mechanism is provided to check the equality of the protection codes.

In the traditional error detection scheme the indices of the arrays are usually compared with the upper and lower limits before making access to the element of arrays. On the other hand, our method is to defer the error detection to the time of the read-out of the physical memory. Comparison of the tag with protection code can be overlapped with the instruction execution, hence no overhead is incurred from the check of protection codes. In the case of write operation, the tag should be read out preceding the write operation of the data of the array element.

In our scheme, all the elements of an array need not be allocated in a single consecutive area of the physical memory. Partitioning of a single array, in case that storage management policy requires it, poses no efficiency problem to our boundary checking scheme, as opposed to the traditional scheme performed by inserting upper and lower bound checking instruction codes.

2.3 Probabilistic error detection

The scheme completely detects the address range violation, when the number of tag bits is k and $N(<2^k)$ vectors are processed. When number of vectors active at one time, N exceeds 2^k, complete detection is not possible since $N/(2^k)$ vectors are assigned to the same protection code. Hence, we can detect the range violation with the following probability q:

$$q = \frac{1 - \frac{1}{2^k}}{1 - \frac{1}{N}} \qquad \text{where } N > 2^k, \quad \text{--- (a)}$$

and $\quad q = 1 \qquad \text{where } N \leq 2^k.$

With four bits assigned to a tag (k=4) in handling 32 arrays, the probability q is about 0.97, for example.

Suppose that we make p debug runs and that for each debug run different protection code is randomly assigned, the probability of the detection, r gets as:

$$r = 1-(1-q)^p .$$

The operating systems of large computers usually provides

memory protection schemes incorporated in paging or segmentation. By the use of the combination of these protection techniques, the probability of the detection of the range violation will further be improved.

3. Design of instructions for a high speed check

In this section, let us explain how the run-time check is performed. As a simple example, we shall use a program to find the maximum element of an array. In the subsequent description, we shall use FORTRAN and IBM system 360/370 assembly language. First, let IX(10) be an array to be processed, and all the elements of array IX has the value 5 as the tag. Program P1 is straightforward coding of the algorithm.

```
      DIMENSION IX(10)
      MAX=0
      DO 10 I=1,10
10    IF (MAX.LT.IX(I))  MAX=IX(I)
```

Program P1. Find a maximum number in an array

Program P1 is compiled to program P2 to take advantage of the hardware mechanism as explained in section 2.2. Here, we introduce an instruction LTAG. The function of this instruction is to load a tag value into a specified tag register. When we use a general register for index modification for array referencing, the same numbered tag register as the general

register is used to check the protection code. The tag check mechanism is activated when this tag register holds a non-zero value. In the following example, the access to an array is checked in the two statements: *1 and *2.

```
MAX     EQU     5
TEMP    EQU     6
I       EQU     7
TAG     DC      F'5'
X       DS      10F'0'
.................
        L       MAX,=F'0'
        L       I,=F'1'
        LTAG    TEMP,TAG
LOP10   LR      TEMP,I
        SLA     TEMP,2
*
* Tag register 6 is used for bound checking in parallel
* with the following execution of instructions *1 and *2.
*
        C       MAX,X-4(TEMP)     *1
        BL      TEST
        L       MAX,X-4(TEMP)     *2
TEST    A       I,=F'1'
        C       I,=F'10'
        BL      LOP10
.................
```

Program P2. An example using tag register in assembly language

4. Multi-dimensional arrays

In order to check each index of multi-dimensional array, we divide the array into single dimensional arrays (columns), and assign different protection code to the tags associated with those single dimensional arrays. In the case of two dimensional array, the head addresses of all the single dimensional arrays are kept as a dope vector [6] with row and column tags in each entry of the table as shown in Figure 3.

Suppose we want to reference element A(i,j) of two-dimensional array $A(n, m) = (A_1, A_2, \ldots\ldots\ldots, A_m)$ where A_i is a vector of length n. We assign protection codes to m row vectors and a dope vector. Given two protection codes s and t, the algorithm to check the array bound check is as follows (cf. Figure 3):

(i) Load tag s and make access to the j-th element of the dope vector.

(ii) If the row-tag of the dope vector does not match with s, array bound error is detected.

(iii) Load tag t and address of the j-th vector,

(iv) Make access to the i-th element of this vector.

(v) If the tag of the element does not match with t, array bound error is detected.

We can expand this address table scheme for higher dimensional arrays.

5. Concluding remarks

In designing the hardware scheme for array bound checking, we started from the following observations:

* Correct programs should run with no overhead of the array bound check.
* One can not tell wehther "correct" programs are really correct except for simple (provable) programs.

"Correct" programs may suffer from round-offs, unintended

coercions and array range violation in actual runs, although they are seemingly correct. The proposed scheme can reduce the possibility of hidden errors incurred from array bound violation at run time, and to detect at debug time the array bound violation almost completely.

We showed that with small number of bits for a tag incorporated to existing framework of architecture, array bound violation can be detected with high probability.

Probabilistic check would be a matter for further discussion. However, we believe that our standpoint is justified from the viewpoint that our hardware array bound checker scheme is meant to enhance the "reliability of software." At the current level of the art of software engineering technology, we cannot expect the complete reliability of the software and we have to use programs which might contain bugs; most of them might appear at small probability. In these environment, the probabilistic error detection scheme would contribute to develop the total reliability of the software.

References:

[1] D.E. Knuth, "An empirical study of FORTRAN programs," Software Practice and Experience, Vol.1, pp. 105-133, 1971.

[2] N. Suzuki and K. Ishihata, "Implementation of array bound checker," Proc. 4th ACM symposium on princiles of programming languages, 1977.

[3] K. Itano and T. Ida, "A high speed array bound checking scheme using a tag (in Japanese)," Proc. 21th annual conference of Information Processing Society of Japan, 1980.

[4] E.A. Feustel, "The Rice Research Computer - a tagged architecture," SJCC, pp. 369-377, 1972.

[5] E.A. Feustel, "On the advantgae of tagged architecture," IEEE trans. Comput., Vol. C-22, pp. 644-656, 1973.

[6] D. Gries, Compiler construction for digital computers, Wiley, 1971.

Appendix Proof of probability (a)

We assume that erroneous accesses to N-1 arrays are equally likely. Out of N-1 arrays, the number of arrays that have the same protection code is:

$$\frac{N}{2^k} - 1.$$

Detection fails when access is made to those arrays.
Hence, the probability of detecting the array bound violation is:

$$1 - \frac{\frac{N}{2^k} - 1}{N - 1}.$$

Figure Captions

Figure 1. Array mapped on a memory space.

Figure 2. Array bound checker on tagged architecture.

Figure 3. Data structure for multi-dimensional array.

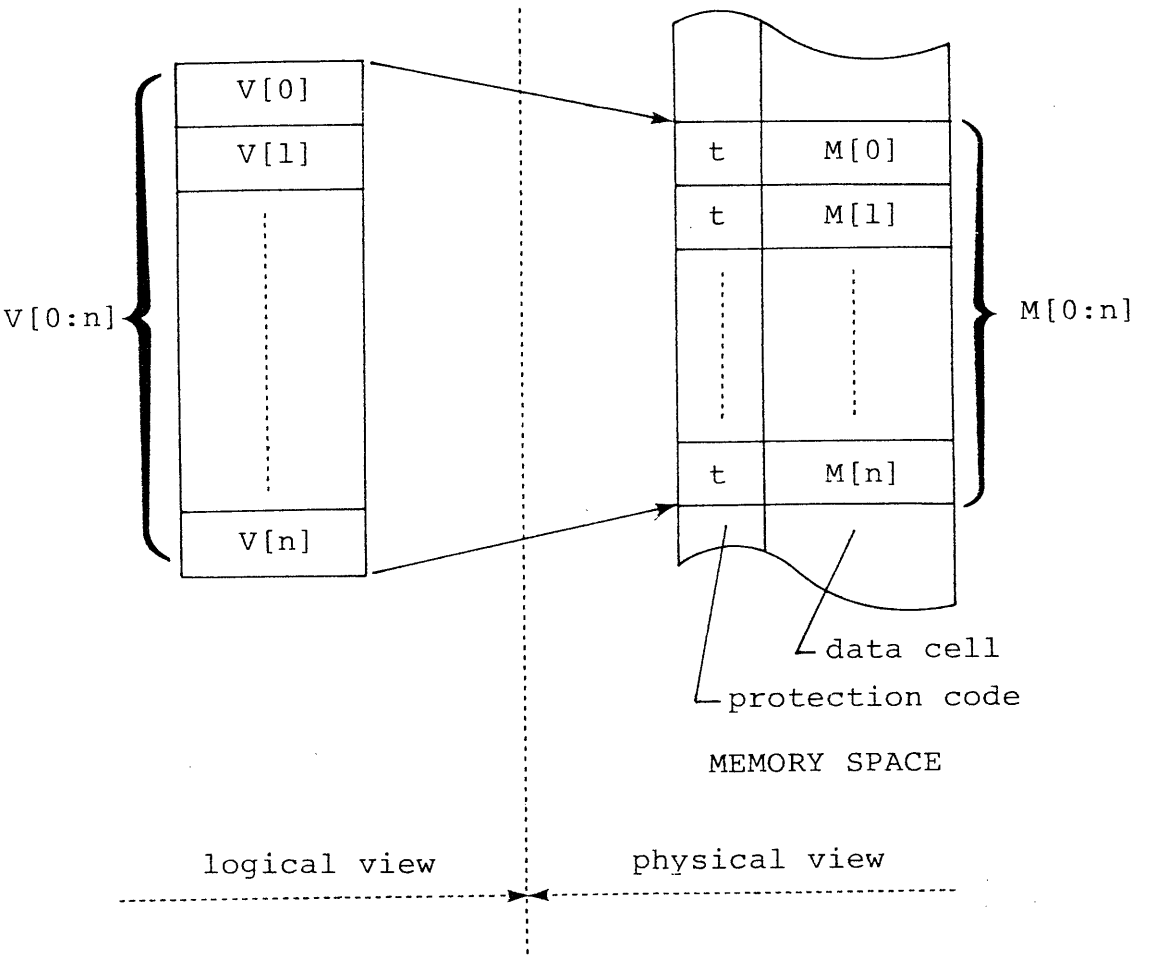

Figure 1. Array mapped on a memory space.

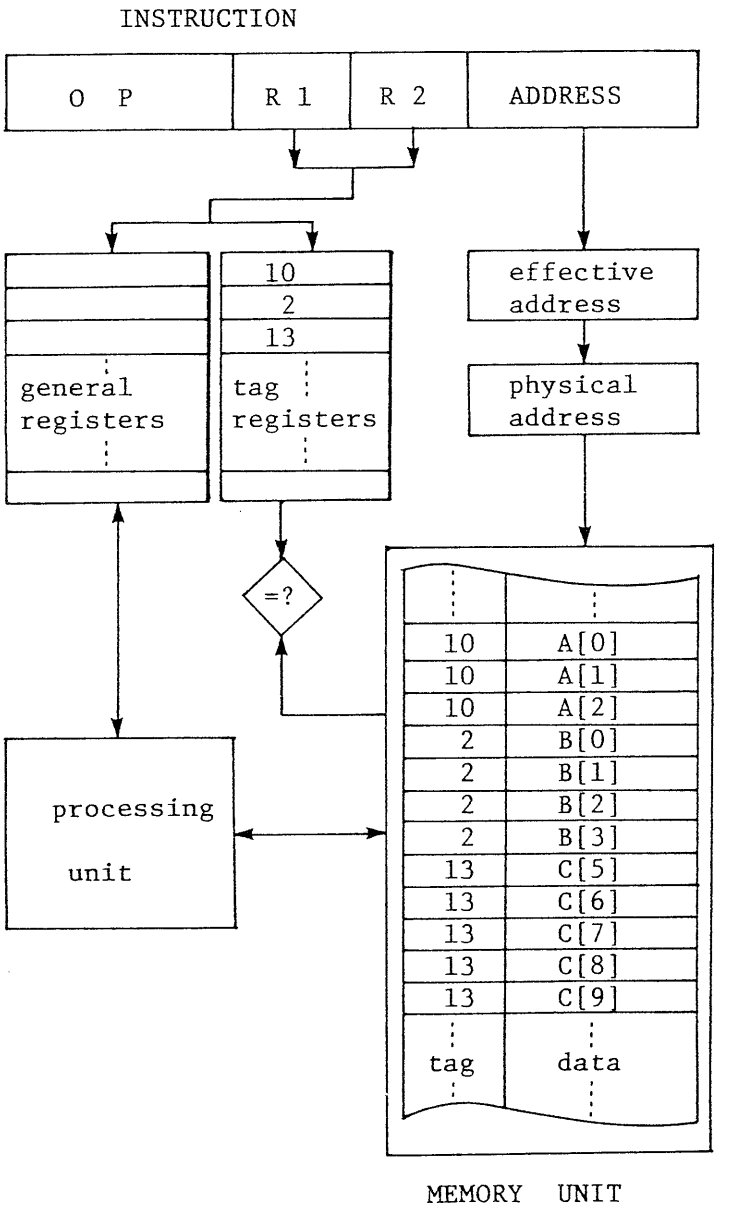

Figure 2. Array bound checker on tagged architecture.

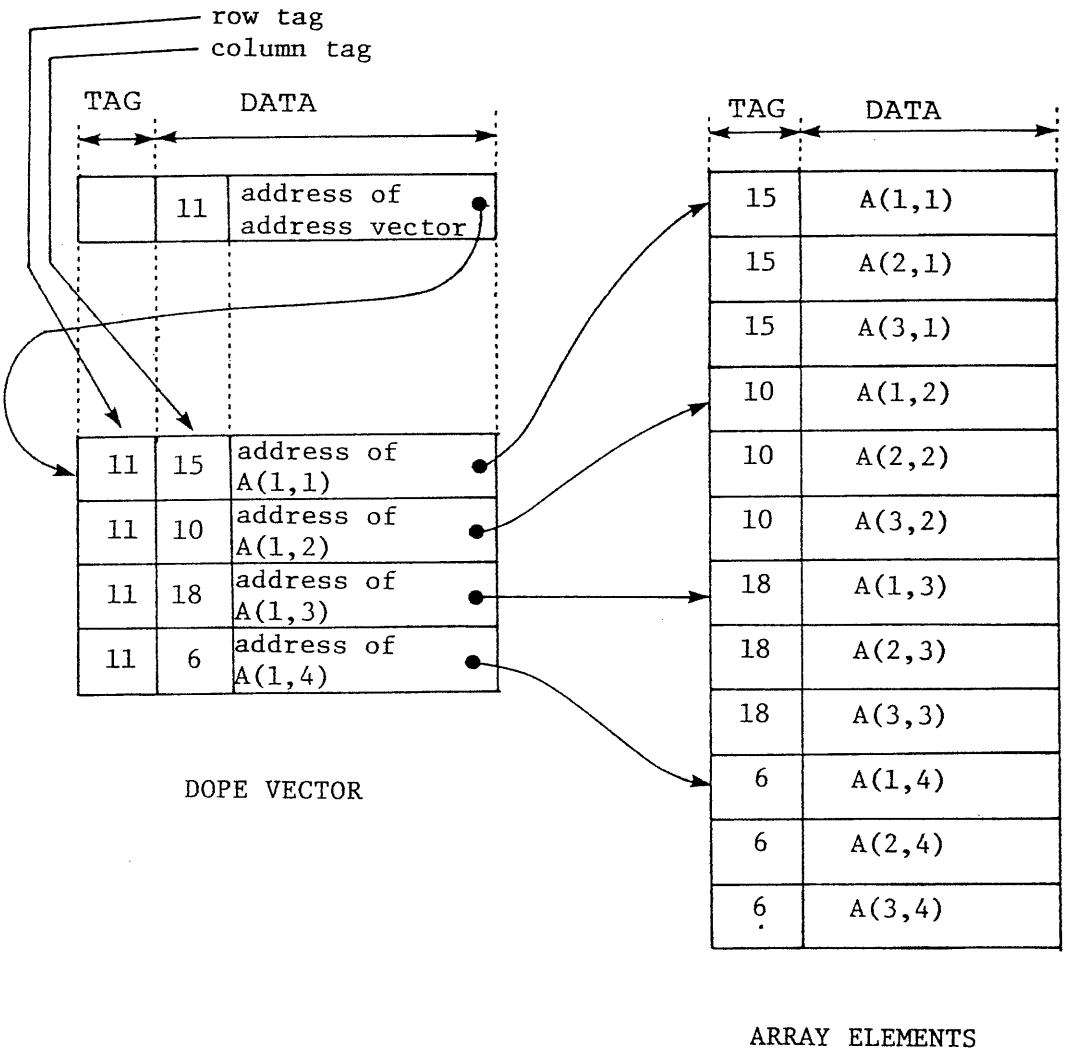

Figure 3. Data structure for multi-dimensional array.

INSTITUTE OF INFORMATION SCIENCES AND ELECTRONICS
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI 305 JAPAN

REPORT DOCUMENTATION PAGE	REPORT NUMBER ISE-TR-80-16

TITLE

Hardware array bound checker on tagged architecture

AUTHOR(s)

Kozo Itano

Tetsuo Ida

REPORT DATE	NUMBER OF PAGES
July 1, 1980	16
MAIN CATEGORY Computer systems	CR CATEGORIES 6.2, 6.3

KEY WORDS

Array bound checker, tagged architecture, probabilistic checking.

ABSTRACT

A hardware scheme to check the correctness of array accesses in high speed is presented. The new scheme is based on a tagged architecture. A check is made whether the protection code of an array matches with the tag of the memory word. In order to check a large number of arrays efficiently by limited number of bits for the tag, a probabilistic check algorithm is introduced.

SUPPLEMENTARY NOTES