stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 123/79 NOVEMBER

P. KLINT

HOW INEFFICIENT ARE STACK-ORIENTED ABSTRACT MACHINES ?

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

1980 Mathematics subject classification: 68B10

ACM-Computing Reviews-category: 4.22, 4.6, 6.21

How inefficient are stack-oriented abstract machines ?*)

by

Paul Klint

ABSTRACT

The discussion on instruction-set forms ([1], [2], [3], [4]) has revealed that stack-oriented instruction sets are less efficient than storage-to-storage instruction sets. What are the implications of this observation for the efficiency of stack machines which are frequently used as abstract machine in implementations of high level programming languages?

KEY WORDS & PHRASES: Computer architecture, instruction sets, stack machines, storage-to-storage machines, abstract machine code, portability.

*) This paper is not for review; it is meant for publication elsewhere.

1. INTRODUCTION

In many portable software systems some form of abstract machine code is used as an interface between machine-dependent and machine-independent parts of the system. A well-known example is the PASCAL-P [5] compiler which generates code for an abstract stack machine. How does the instruction-set form of such an abstract machine affect the overall efficiency of that system? This question will be answered by comparing abstract machines with:

- Stack-oriented architecture (S-machines).

- Storage-to-storage architecture with two operands per instruction (2A-machines). For example ADD A,B with meaning A := A + B.

- Storage-to-storage architecture with three operands per instruction (3A-machines). For example ADD A,B,C with meaning A := B + C. We allow the omission of unused operands in instructions; this is useful for monadic operators and assignment: For example MOVE A,B with meaning A := B.

2. AN ASSESSMENT OF THREE ARCHITECTURES

The performance of an implementation of some high level language depends (for the sake of this discussion) on:

- The frequency distribution of statements in the high level language.

- The number of abstract machine instructions needed for the translation of each statement form in the high level language.

- The efficiency of operand addressing for each abstract machine code instruction.

The frequency of statements in the high level language is fixed for each language and does not depend on a particular implementation. (It has even been observed that this distribution is highly independent of the high level language itself [6].) In the sequel we will use the same distribution as used in [7], which is based on the complexity of assignment statements:

	statement form	estimated frequency
S1	A := B	72.1%
S2	A := A + B	14.4%
S3	A := B + C	6.1%
S4	A := (B + C) * (D - E)	2.7%
S5	A := B + C + D - E	4.7%

This distribution covers circa 50% of all expressions in programs.

The five statement forms are represented in an S-machine, 2A-machine and 3A-machine as shown in the following table. Instructions gives the number of instructions required for the translation of each statement form. Size gives the total number of bits required for the instruction fields (I) and for the address fields (A) in the translation. Elements gives the total number of instruction elements (i.e. instruction fields and address fields) in each translation.

	S-machine	2A-machine	3A-machine
S1: A:=B	PUSH B STORE A	MOVE A,B	MOVE A,B
instructions	2	1	1
size	2*I+2*A	1*I+2*A	1*I+2*A
elements	4	3	3
S2: A:=A+B	PUSH A PUSH B ADD STORE A	ADD A,B	ADD A,A,B
instructions	4	1	1
size	4*I+3*A	1*I+2*A	1*I+3*A
elements	7	3	4

S3: A:=B+C	PUSH B	MOVE A,B	ADD A,B,C
	PUSH C	ADD A,C	
	ADD		
	STORE A		
instructions	4	2	1
size	4*I+3*A	2*I+4*A	1*I+3*A
elements	7	6	4
S4: A:=(B+C)*(D-E)	PUSH B	MOVE A,B	ADD T1,B,C
	PUSH C	ADD A,C	SUB T2,D,E
	ADD	MOVE T1,D	MUL A,T1,T2
	PUSH D	SUB T1,E	
	PUSH E	MUL A,T1	
	SUB		
	MUL		
	STORE A		
instructions	8	5	3
size	8*I+5*A	5*I+10*A	3*I+9*A
elements	13	15	12
S5: A:=B+C+D-E	PUSH B	MOVE A,B	ADD T1,B,C
	PUSH C	ADD A,C	ADD T2,T1,D
	ADD	ADD A,D	SUB A,T2,E
	PUSH D	SUB A,E	
	ADD		
	PUSH E		
	SUB		
	STORE A		
instructions	8	4	3
size	8*I+5*A	4*I+8*A	3*I+9*A
elements	13	12	12

Note that in the code for 2A-machines it is assumed that all operands are different in forms S3, S4 and S5. This assumption favours 2A-machines.

The average instruction size (AIS) and the average number of elements (ANE) can be computed if one takes into account the frequency distribution of statement forms. The result is shown in figure 2.1.

	Average instruction size (AIS)	Average number of elements (ANE)
S-machine	2.85*I+2.43*A	5.28
2A-machine	1.31*I+2.62*A	3.93
3A-machine	1.15*I+2.72*A	3.87

Figure 2.1. Average instruction size (AIS) and average number of elements (ANE) per instruction.

AIS_S, AIS_{2A} and AIS_{3A} are plotted in figure 2.2 as functions of the ratio between A and I (i.e. A/I) and are expressed relative to the instruction field size (i.e. AIS/I). This figure illustrates that S-machines are worse than both 2A-machines and 3A-machines in all practical situations. 3A-machines are slightly better than 2A-machines when A and I are of the same order of magnitude. The following equalities hold:

$AIS_{2A} = AIS_{3A}$ for A/I = 1.5
$AIS_S = AIS_{2A}$ for A/I = 5.9
$AIS_S = AIS_{3A}$ for A/I = 8.1

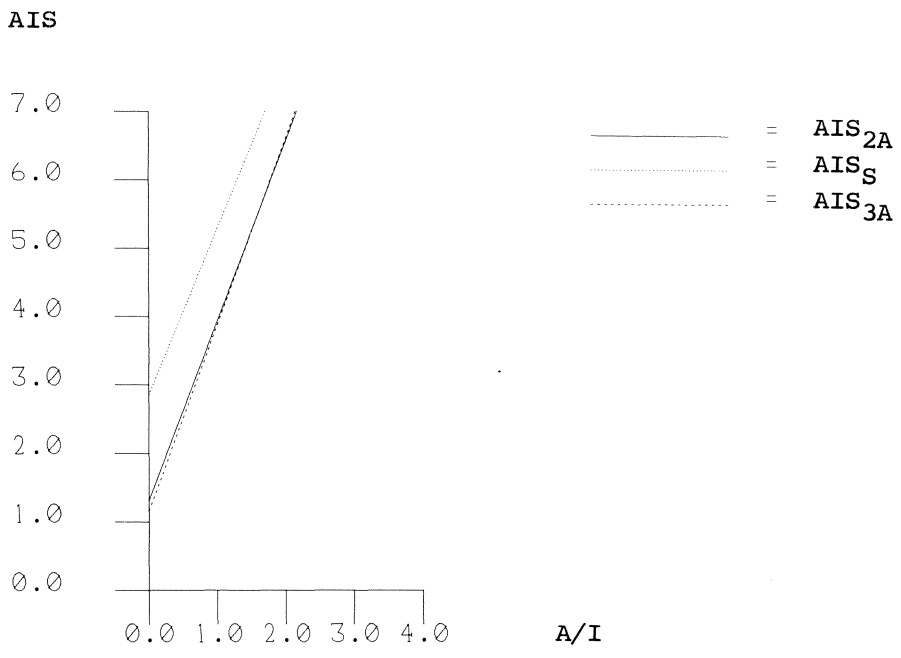

Figure 2.2. Average instruction size (AIS).

For smaller values of A, S-machines become less and less efficient with regard to average instruction size. Figure 2.3 gives an impression of this phenomenon. In high level language architectures addresses tend to be short (say $A \leq 8$) since all variables are accessed relative to some base address (i.e. globals, locals, formals) and the number of different variables in each class tends to be small. This makes S-machines less attractive for this particular application.

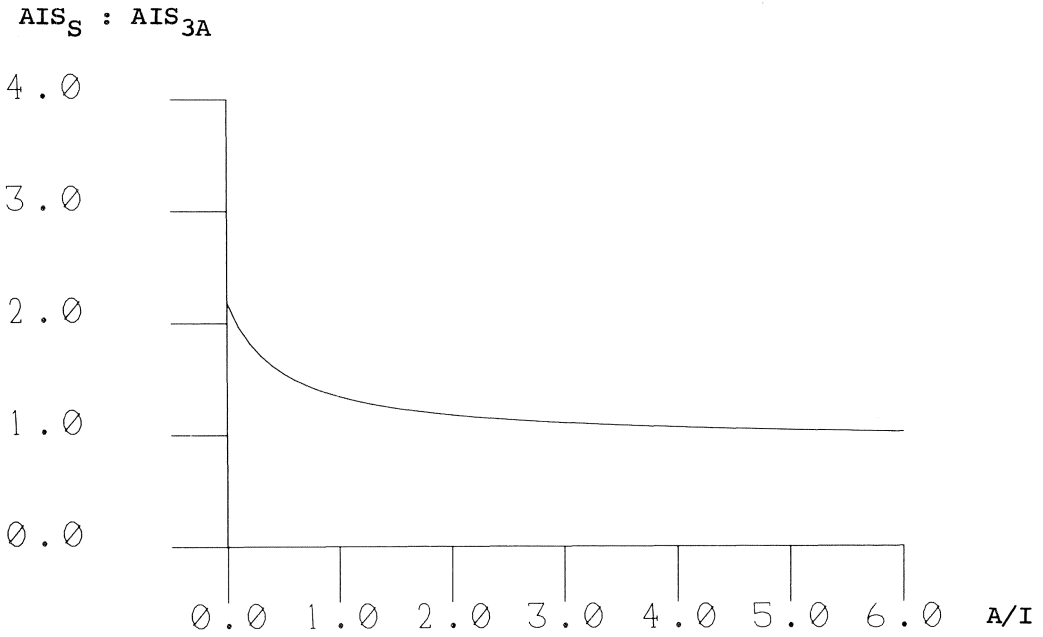

Figure 2.3. Ratio average instruction size (AIS) for S-machine and 3A-machine.

3. MEASUREMENTS

To compare the run-time efficiency of the S-machine, 2A-machine and 3A-machine, one can realize these architectures in two ways:

EXP: Expand each abstract machine instruction to executable machine code. With regard to execution time this is the most efficient way to implement each abstract machine.

INT: Represent each instruction by an opcode followed by zero or more operands and let a (software) interpreter execute these instructions.

Implementation of this scheme on a PDP11/45 (with cache memory) gives

the following results:

	EXP	INT	EXP:INT
S-machine	12.5	36.5	0.3
2A-machine	7.2	21.2	0.3
3A-machine	7.5	19.3	0.4

These figures show the execution time in seconds for the execution of one million statements distributed over the five statement types as described above. These figures can also be interpreted as average instruction execution time (AIT) in micro-seconds.

Finally, we give measured relative execution times:

	EXP	INT
AIT_S : AIT_{2A}	1.7	1.7
AIT_S : AIT_{3A}	1.7	1.9
AIT_{2A} : AIT_{3A}	1.0	1.1

4. CONCLUSIONS

- 2A-machines and 3A-machines are at least 1.7 times as fast as S-machines.

- The average instruction size on S-machines is greater then on 2A-machines and 3A-machines. 3A-machines are slightly better than 2A-machines when the sizes of instruction fields and address fields are of the same order of magnitude.

5. ACKNOWLEDGEMENTS

Jan Heering and Arthur Veen commented on drafts of this paper.

6. REFERENCES

[1] Myers, G.J., The case against stack-oriented instruction sets, Computer Architecture News, 6(1977)3, 7-10.

[2] Doran, R.W., Letter to the Editor, Computer Architecture News, 7(1978) 1, 25-28.

[3] Keedy, J.L., On the evaluation of expressions using accumulators, stacks and store-to-store instructions, Computer Architecture News 7(1978) 4, 24-27.

[4] Keedy, J.L., More on the use of stacks in the evaluation of expressions, Computer Architecture News, 7(1979) 8, 18-21.

[5] Nori, K.V., Amman, U., Jensen, K. & Naegeli, H.H., The PASCAL (P) compiler: implementation notes, ETH Zuerich, report 10, 1974.

[6] Halstead, M.H., Elements of Software Science, American Elsevier, New York, 1977.

[7] Myers, G.J., The evaluation of expressions in a storage-to-storage architecture, Computer Architecture News 6(1978) 9, 20-23.