COMPUTATIONAL DESIGN ALTERNATIVES WITH MICROPROCESSOR-BASED SYSTEMS

by

Sigurd L. Lillevik, Assistant Professor
Department of Electrical Engineering
Utah State University
Logan, Utah 84322

and

P. David Fisher, Professor
Department of Electrical Engineering and Systems Science
Michigan State University
East Lansing, Michigan 48824

ABSTRACT

This paper examines and characterizes four elemental hardware computational design alternatives (CDA's) and presents a structured approach to computational section design which incorporates a rigorous, theoretic foundation. The DIRECT CDA incorporates a single microprocessor (μP) and memory. The AU CDA contains a μP, memory, and arithmetic unit. A μP, memory, and calculator chip comprise the CALC CDA. Finally, several μP's and memories in a Master/Slave arrangement implement the multiple-μP mμP CDA. A common set of attributes--precision, speed and cost--facilitates comparison. Using these attributes, Multiattribute Utility Theory assesses a numeric quantity, the utility, to represent each CDA's relative usefulness. Thus, design involves selecting the CDA with the greatest utility.

1. Introduction

Technological advances made within the electronics industry during the early and mid 1970's stimulated the rapid development and broad acceptance of microprocessor-based systems. Today, these systems span a wide range of applications with considerable variation in computational requirements. The purpose of this paper is to investigate and characterize a set of important computational design alternatives (CDA's) couched within a structured approach to design which incorporates a firm, theoretic foundation and which illustrate the use of this theory in characterizing CDA's.

A generalized μP-based system may consist of several sections, e.g., data conversion, data communication, operator interaction, computation, etc., each with specific and unique responsibilities. In such a system each section communicates with the others over a common system bus. The "computation" section functions to accomplish some distinct set of arithmetic calculations required for the particular application. Because of technological progress, various combinations of hardware and software may implement the needed computation. Thus, a computational design alternative (CDA) represents: the specific combination of hardware and software required to accomplish a particular computation.

All "computational" sections of a μP-based system can be decomposed into four "elemental" CDA's, each with their own distinct combination of hardware and software to accomplish computations. First, the DIRECT CDA consists of only a μP and memory connected via the system bus as shown in Figure 1a. Since the memory contains arithmetic (add, subtract, multiply, and divide) and logical subroutines, the DIRECT CDA performs all computations in software. Next, the second CDA employs a μP, memory, and an arithmetic unit (AU) (see Fig. 1b), all joined by the system bus. The AU CDA differs from the DIRECT CDA in how it achieves multiplies and divides. For the AU CDA it writes the two operands (numbers) into AU buffers, starts the multiply or divide operation, and simply reads back the result. But the AU CDA, like the DIRECT CDA, performs adds, subtracts and logical instructions totally in software. The third CDA incorporates a calculator chip to accomplish arithmetic computations as depicted in Figure 1c. Here, the CALC CDA consists of a μP, memory, and the calculator chip, all linked by the system bus. Within the memory its program must simulate depressing the keys as with a hand held calculator; it must also read back the results and decode them when the requested function finishes.

The final elemental CDA applies the concept of simultaneous, or parallel, execution: it executes sections of the problem simultaneously, then adds the partial results together for the completed answer[1-4]. Because of this concept the multiple-microprocessor (mμP) CDA involves several μP's and memories in a Master-Slave arrangement as illustrated in Figure 1d. Each Slave memory contains a replica of the DIRECT CDA memory, arithmetic subroutines, and performs all computations in software (like the DIRECT CDA). The Master's memory holds an "overhead" program responsible for routing data and preliminary results from Slave to Slave, and for forming the completed results. This memory may contain arithmetic subroutines to handle such tasks as testing for convergence and calculating indicies.

The above mμP CDA Master/Slave arrangement does not necessarily produce optimal results; an architecture designed for a unique problem, or class of problems, can potentially decrease execution-time and reduce costs. But the proposed mμP CDA does present a simple, general-purpose architecture that solves many problems well and, additionally, lends itself to straightforward analysis.

For each elemental CDA a unique mixture of hardware and software accomplish computations. Thus, the properties of each CDA vary considerably. So judicious combinations of the four CDA's (DIRECT, AU, CALC, and mμP) will realize the computation section of a μP-based system.

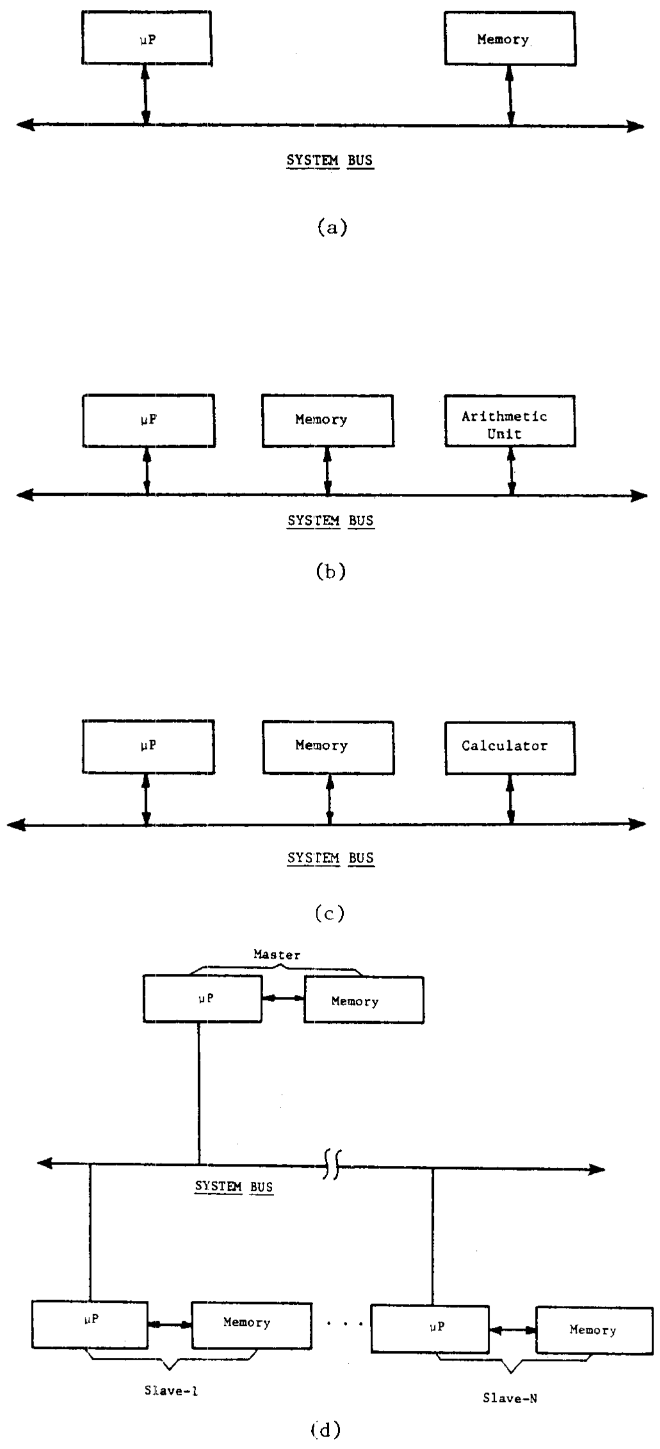

Figure 1. Block diagram of the (a) DIRECT, (b) AU, (c) CALC, and (d) mμP computational design alternatives.

2. CDA Attributes

Comparison of a group of objects occurs through evaluation of a common subset of characteristics, or properties. Moreover, the selected subset of attributes must reflect the important features and qualities of each member. When comparing a set of CDA's, several attributes satisfy these requirements: precision, execution-time, cost, power dissipation, circuit complexity, programming language, circuit reliability, packaging demands, maintenance schedule, etc. Too few attributes results in incomplete examination of the objects while too many attributes may cause unnecessary confusion. For the above list, all but the first three attributes depend strongly on the remainder of the P-based system and, thus, fail to depict attributes indigenous to the CDA's. But precision, execution-time, and cost represent convenient and illustrative attributes for demonstrating the principal characteristics of CDA's. And Table I summarizes typical device attributes used throughout this paper.

2.1 Precision

Two common number representations, fixed-point and floating-point, both identify discrete values on the real-number line. For the fixed-point format an implied binary-point lies between two specific bits in the word, and with floating-point numbers the binary-point varies (this requires storage of the position). In the research reported here, all numbers conform to the fixed-point format. Thus, precision involves the quantity of 8-bit memory words used to form a particular number; e.g., p=1 for single-precision, p=2 for double precision, etc. Since the μP's perform two's complement arithmetic, all numbers correspond to multiple-precision, fixed-point, two's complement quantities.

2.2 Execution Times

Execution-time represents the number of seconds needed to complete a specific computation. Since addition/subtraction and multiplication/division both correspond to complementary arithmetic and digital logic functions, the execution-time of an addition roughly equals a subtraction and a multiplication roughly equals a division. These assumptions greatly facilitate analysis of execution-times. Thus, if the number of adds and multiplies can approximate the execution-time of an application program, then these values multiplied by the time to perform an add and multiply sum to give the exution-time. Symbolically, let

A_i = number of adds

M_i = number of multiplies

a_i = add time

m_i = multiply time

$$T_i = A_i a_i + M_i m_i = \text{execution-time} \qquad (1)$$

where,

$$i = \begin{cases} 1, \text{ DIRECT CDA} \\ 2, \text{ AU CDA} \\ 3, \text{ CALC CDA, and} \\ 4, \text{ m}\mu\text{P CDA.} \end{cases}$$

In Equation 1 the specific CDA architecture dictates the expressions for a_i and m_i, while the algorithm of the application program configures constants A_i and M_i.

Multiple-precision arithmetic operates on each word of the number individually to produce the final result; e.g., consider the multiple-precision add flowchart in Figure 2a. Here, corresponding words of the two operands add to produce the partial sum and the carry ripples through from word to word. Since each pass through

Table I. Representative characteristics of computational devices

a) µP's	b) Solid-state memories
8-bit word length	1-8k bits/chip
8 µsec. cycle-time	1 µsec. assess-time
$10.00 cost	$0.01/bit cost
c) Calculator and support chips	d) Arithmetic units
multiple-precision word-length	8-bit word length
50 msec. execution-time	100 nsec. execution-time
$200.00 cost	$100.00 cost

the loop involves about 10 instructions, then for

t_0 = cycle-time (8 µsec.)

p = precision, and

$$a_1 = 10pt_0. \qquad (2)$$

Also, the AU and mµP CDA's employ identical add algorithms, so

$$a_2 = 10pt_0, \text{ and} \qquad (3)$$

$$a_4 = 10pt_0. \qquad (4)$$

But with the CALC CDA each operand requires decoding and encoding from binary to BCD formats, and each arithmetic operation demands "digit intry" (simulation of key depresses) and "function" time (see Fig. 2b and 2c). Decoding/encoding tasks and the answer reads occur in secs., but when compared to the digit entry and function delays (msecs.) they contribute no significant time to the add, or multiply, execution-times. To determine the approximate number of digits sent to the calculator chip each two's complement number roughly ranges in magnitude $\pm 2^{8p-1}$, and on the average the number decoded falls in the middle, 2^{8p-2}. For $1 \leq p \leq 4$ these numbers contain around 2p decimal digits; e.g., if p=1 and $2^{8p-2}=2^6=64$, then the µP sends 2 decimal digits to the calculator chip. With these ideas the execution-time of a multiple-precision add instruction for the CALC CDA becomes

t_e = digit entry time (40 msecs.)

t_a = "add" function time (90 msecs.), and

$$a_3 = 2(2p+1)t_e + t_a. \qquad (5)$$

Equations 2 through 5 yield estimates for the multiple-precision add times of all CDA's and Figure 3 illustrates these values for various precisions. In this figure the most noticeable observation concerns the difference in add times: nearly 3 orders of magnitude.

For a p-precision, two's complement, fixed-point binary number multiplied by another, the result yields a 2p-precision product. Since both the DIRECT and mµP CDA's do not contain hardware multiply instructions they may use the "add-and-shift" algorithm[5] to determine C = A*B (see Fig. 4). On the average, this algorithm executes 4p, p-precision shifts. Using Equation 2 for an estimate of the add time and assuming that each shift requires p cycle-time

$$m_1 = 4p(20p\, t_0) + 8p(pt_0) \qquad (6)$$
$$= 88p^2 t_0.$$

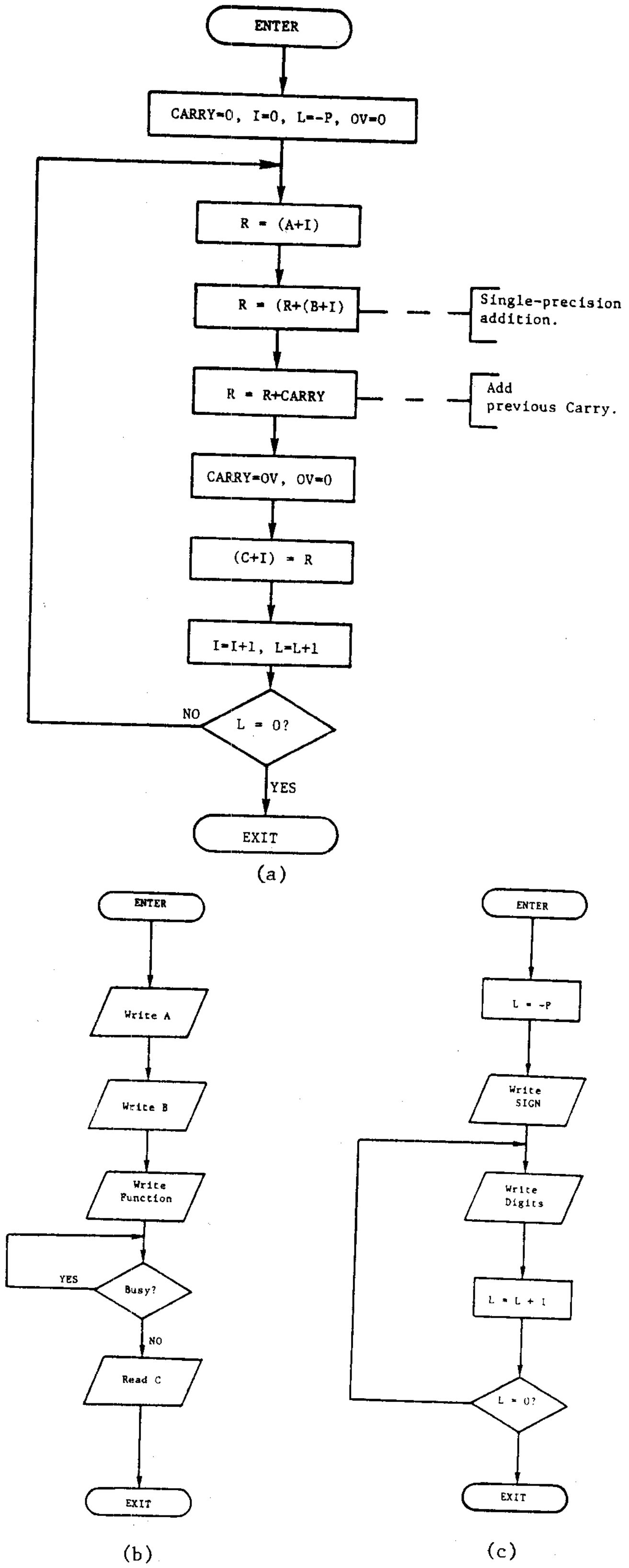

Figure 2. (a) Multiple-precision addition flowchart. (b) "Operation" flowchart for CALC CDA. (c) [illegible] entry" flowchart for CALC CDA.

And since the mμP CDA uses the same software,

$$m_4 = 88p^2t_0. \qquad (7)$$

With the AU CDA the problem changes dramatically due to its 8-bit by 8-bit hardmultiply. If the P needs two I/O instructions to write the parameters into the AU, one to wait for the multiply to finish, and two to read the 16-bit product, then one single-precision multiply requires $(5t_0)$ time. But for multiple-precision numbers the task becomes much more difficult; consider the notation:

$$
\begin{aligned}
A &= (a_p + a_{p-1} + \ldots + a_1) = \text{multiplier} \\
B &= (b_p + b_{p-1} + \ldots + b_1) = \text{multiplicand, and} \\
C &= AB \\
&= (a_p b_p + a_p b_{p-1} + \ldots + a_p b_1) \\
&+ (a_{p-1} b_p + a_{p-1} b_{p-1} + \ldots + a_{p-1} b_1) + \ldots \\
&+ (a_1 b_p + a_1 b_{p-1} + \ldots + a_1 b_1), \\
&= \text{product.}
\end{aligned}
$$

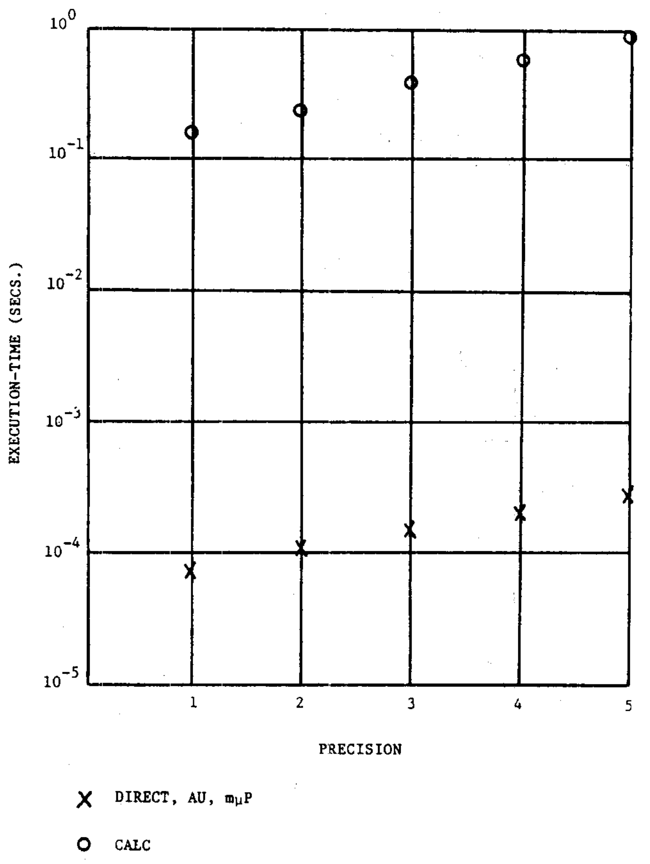

Figure 3. Multiple-precision addition time for various CDA's.

Thus, multiple-precision multiplies for the AU CDA involves p^2, single-precision multiplies (partial products) and p^2, 2p-precision adds (see Fig. 5); combining,

$$
\begin{aligned}
m_2 &= p^2(5t_0) + p^2(20pt_0) \qquad (8) \\
&= 5p^2(1 + 4p)t_0.
\end{aligned}
$$

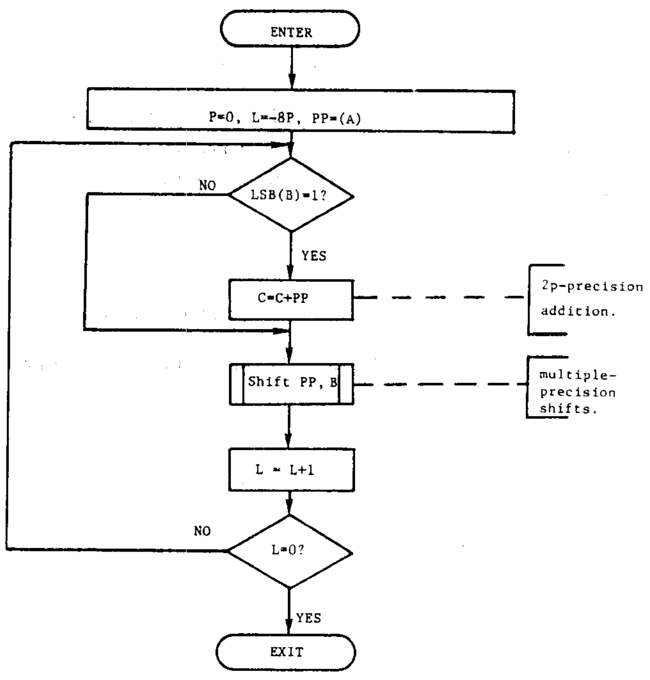

Figure 4. Multiple-precision multiply flowchart for DIRECT CDA.

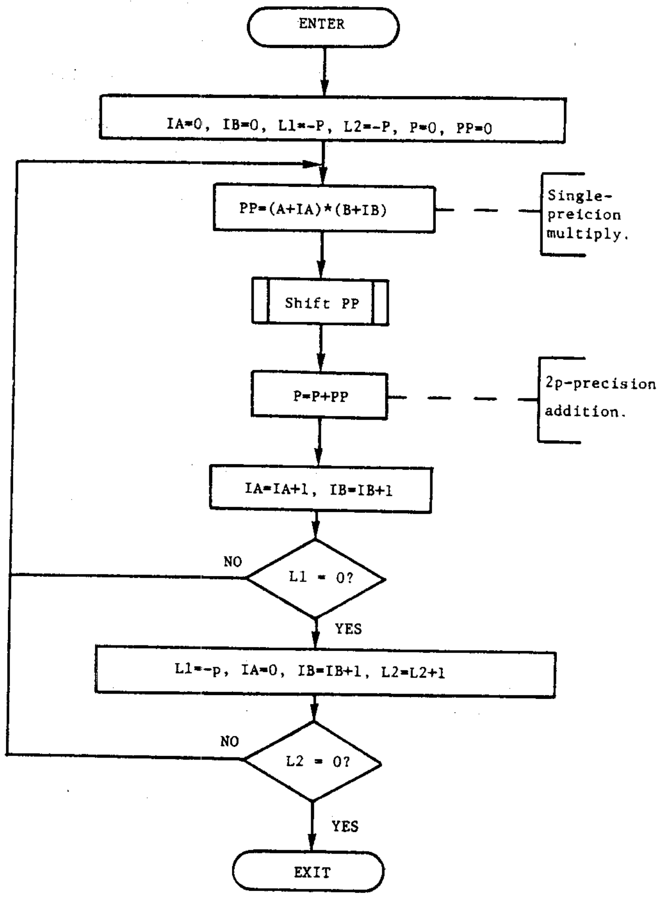

Figure 5. Multiple-precision multiply flowchart for AU CDA. •

For the CALC CDA the analysis of Equation 5 directly applies to finding the multiple-precision multiply time; only the "multiply" function replaces the "add" function time; i.e.,

t_e = digit entry time (40 msec.)

t_m = "multiply" function time (120 msec.), and

$$m_3 = 2(2p + 1)t_e + t_m. \qquad (9)$$

So Equations 6 through 9 give estimates for the multiple-precision multiply times of all CDA's, and Figure 6 illustrates these values for various precisions. From this Figure the most striking feature pertains to the difference in multiply times of the CALC CDA compared to the others: roughly 2 orders of magnitude. Also, the change in execution-time with precision, or slope, varies with each CDA. The AU CDA rises the fastest, then the DIRECT, and the CALC CDA contains the slowest increase. This observation causes the DIRECT's multiply time to become faster than the AU's for p 3. Now, once the constants A_i and M_i have been found for an application, Equation 1 with Table II can give an estimate of the execution-time for each CDA.

The flowcharts for various multiple-precision add and multiply instructions provide only the basic outline. Among the functions added to an implemented algorithm include initialization, zero and sign checks, overflow/underflow detection, etc. But these extra duties do not contribute significantly to the overall execution-time, nor do they represent more than second and third-order terms in the execution-times (Equations 2-9).

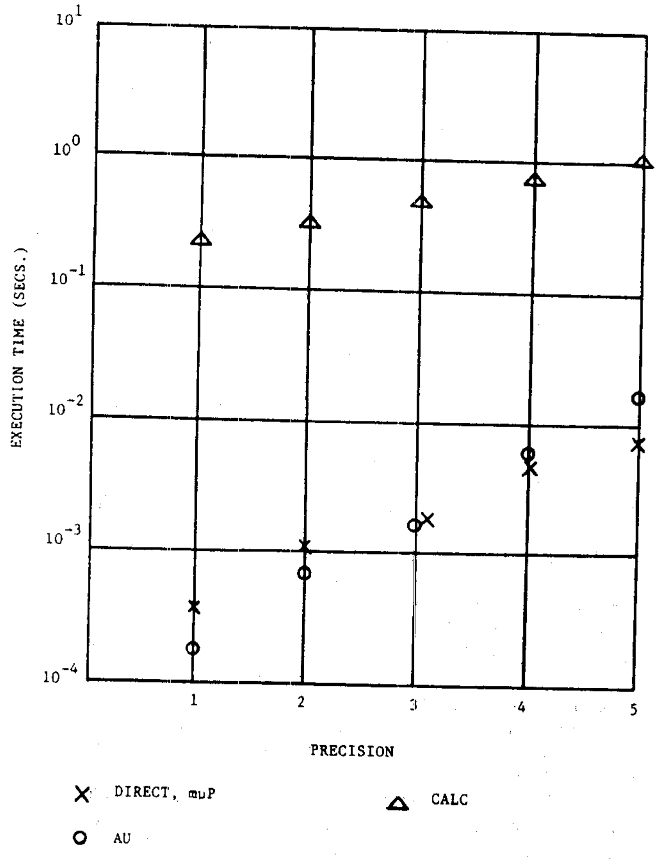

Figure 6. Multiple-precision multiply time for various CDA's.

Table II. Add and Multiply execution-times for various CDA's.

i	a_i	m_i
1	$10pt_o$	$88p^2t_o$
2	$10pt_o$	$5p^2(1+4p)t_o$
3	$2(2p+1)t_e+t_a$	$2(2p+1)t_e+t_m$
4	$10pt_o$	$80p2^{4p-1}t_o$

2.3. Costs

The monetary expense incurred with each CDA could include several components, many of which depend on the remainder of the μP-based system, so this development focuses on the semiconductor parts cost. For all CDA's two principal terms add to give the total cost, one corresponds to an elemental (or basic) CDA cost and the other to an application dependent cost:

C_{ei} = elemental CDA cost

C_{ai} = application cost, and

$$C_i = C_{ei} + C_{ai};\ 1 \le i \le 4; \qquad (10)$$

= total cost (in \$'s).

First, the elemental cost term C_{ei} relates to the basic expense of procuring a CDA and its mathematical subroutine's memory. And three main factors comprise C_{ei},

IO_i = I/O device cost

R_i = mathematical subroutine's cost

CPU_i = μP cost, and

$$C_{ei} = IO_i + R_i + CPU_i;\ 1 \le i \le 4. \qquad (11)$$

Second, the application cost term C_{ai} pertains only to the added expense of the application memory. This involves two terms, one for program memory and one for data storage:

w_o = cost per word (\$0.08)

P_i = program memory

D_i = data memory, and

$$C_{ai} = w_o(P_i + pD_i);\ 1 \le i \le 4. \qquad (12)$$

Because term C_{ei} in Equation 10 does not depend on the application it can be determined now, yet term C_{ai} must be deferred until an application program is defined. Hence, the three terms IO_i, R_i, and CPU_i in Equation 3.11 need evaluation for each CDA.

For the DIRECT CDA the I/O devices and P costs involve simple estimates; since no I/O devices reside on the bus IO_1 = \$0.00 and, today, an estimate for CPU_1 = \$10.00. But estimating R_1 involves much more analysis and approximation. From Figure 2a, the multiple-precision add flowchart, if each step involves about 1 word of memory, then 30 words corresponds to a reasonable estimate of the number of words for the instruction. Similarly, the multiply instruction illustrated in Figure 4 uses the add instruction so the number of words roughly equal 20. A summation of these values yields 50 words and doubling this to account for the subtract and divide subroutines results in 100 words. So 100 words at a cost of w_0=\$0.08 per word gives $R_1=100w_0$=\$8.00.

With the AU CDA the expense of the AU changes IO_2; today, a reasonable estimate of IO_2=\$100.00. Still, CPU_2 remains the same as CPU_1, or CPU_2=\$10.00. Following the procedure which determined R_1, the add instruction contributes 30 words while the AU multiply instruction (see Fig. 5) attaches an additional 45 words. Together, they sum to 75 words which when doubled for the subtract and divide subroutines give 150 words; this sets $R_2=150w_0$=\$12.00.

Next, the CALC CDA, like the AU CDA, embodies an expensive I/O device (the calculator chip) and a contemporary estimate for this term assigns IO_3=\$200.00. Yet the CPU estimate continues at CPU_3=\$10.00. With this CDA the exact same flowchart, see Figure 2b and 2c, can accomplish all arithmetic subroutines because only the function code (a parameter) need change from instruction to instruction; e.g., from add to multiply. These figures suggest that about 50 words can retain the program, and $R_3=50w_0$=\$4.00.

Analysis of the last CDA, the mμP alternative, for the C_{e4} term parallels the DIRECT CDA except in quantity. Since the mμP CDA replicates the DIRECT CDA hardware and software, and if it engages S Slaves, then IO_4=\$0.00, $R_4=SR_1$, and $CPU_4=(S + 1)\ CPU_1$. The CPU_4 term multiplies CPU_1 by (S + 1) because the Master μP creates an additional term.

Thus, Table III delineates the estimates for IO_i, R_i, CPU_i, and C_{ei} for all CDA's, $1 \leq i \leq 4$, based on 1978 device costs. From this table the elemental cost of the mμP CDA grows rapidly as the number of Slaves increases. It surpasses the AU CDA by S=6 and the CALC CDA by S=12. Also, the contribution of R_i to C_{ei} for the AU and CALC CDA's (and the mμP CDA for small S) is insignificant because of the expensive IO_i term.

Table III. Estimates of C_{ei} for various CDA's.

i	IO_i	R_i	CPU_i	C_{ei}
1	--	\$8.00	\$10.00	\$18.00
2	\$100.00	\$12.00	\$10.00	\$122.00
3	\$200.00	\$4.00	\$10.00	\$214.00
4	--	\$(S)9.00	\$(S+1)10.00	\$(S)19.00 +10.00

3. Application of Multiattribute Utility Theory

Von Neumann and Morgenstern[6] have developed a decision mechanism, termed Multiattribute Utility Theory (MUT), that can be applied directly to selecting the "best" CDA for a particular application. MUT assigns a numeric quantity, the utility, to each CDA. This scaler quantity indicates a given CDA's usefulness with respect to the other alternatives. Because the attributes satisfy the eight Axioms of MUT plus utility and preferential independence definitions,[7-10] the decision mechanism reduces to an additive utility function; i.e.,

$$u(\bar{x}) = \sum_{i=1}^{3} k_i u_i(x_i) = \text{utility}, \qquad (13)$$

where

$\bar{x}$ = (x_1, x_2, x_3) = consequence,

x_1 = precision, x_2 = execution-time, x_3 = cost,

$u_i(x_i)$ = marginal utility function,

and

k_i = scaling constant, where $\sum_{i=1}^{3} k_i = 1$.

Here, marginal utility functions convert specific attributes of a dimension to a numeric quantity that represents its usefulness. So the utility of a CDA involves a weighted sum of their marginal utilities, and the "best" CDA consists of the alternative with the greatest numeric utility.

By analyzing three examples, the investigation clarified the techniques used to evaluate attributes, and its elucidated use of the additive utility function. These applications--linear regression,[11] matrix inversion,[12] and fast-fourier transform [13] computations--exemplified the overall procedure and, moreover, led to conclusions that characterized each CDA. Specific details are given elsewhere[10] with only a summary of the results given below.

For the matrix inversion example, a typical consequence set included the following marginal utilities: DIRECT = (66, 99, 100), AU = (00, 100, 51), CALC = (100, 00, 00), and mμP = (33, 99, 91). With the weight vector $\bar{k}$ = (0.3, 0.3, 0.3), Equation 13 formed the utility of each CDA as the average of its marginal utilities; i.e., DIRECT = 80, AU = 45, CALC = 33, and mμP = 67. Since the DIRECT CDA posessed the largest utility, it represented the "best" CDA in the above consequence set. As the values in the weight vector changed, the utility of CDA's with strong marginal utilities that correspond to emphasized $\bar{k}$ dimensions increased, while the others decreased, and the "best" CDA varied accordingly. So determining the weight vector $\bar{k}$ involves important evaluation. Similarly, the other examples strengthened these results.

In characterizing the CDA's, the three applications reflected the fundamental features of Tables II and III. Specifically, the CALC CDA always exhibited the slowest execution-time because of its inherent speeds. For the DIRECT and mμP CDA's, which both perform adds and multiplies in software, the m P's speed invariably surpassed the DIRECT CDA due to its simultaneous, or parallel, execution. But this increase varied significantly from example to example, and it primarily depended on the degree of "parallelism" found in the specific example and the severity of the Master's overhead. As precision increased, the computation times and rates of change varied with each CDA. The CALC's rate changes the least, then the DIRECT CDA, and the AU's increased the most. Since the CALC CDA always finds the answer to a fixed number of digits, greater precision only requires additional digit entries and this contributes little to the execution-time. So as precision increased the DIRECT and mμP CDA's computation times become faster than the AU CDA. For larger problem dimensions, all CDA costs increased because of additional data (not program) memory demands. Consequently, the mμP CDA's cost grew dramatically due to repeated hardware and software.

Precision changes did not alter costs significantly since the memory requirements of the problem dimensions dominate those of added precision.

4. Conclusion

As with any research, the investigation reported here points toward several areas of additional study. For example, rather than characterize CDA's through three specific applications, perhaps a generalized algorithm would induce broader conclusions. If integer N measures the size or dimension of a generalized algorithm, then the time complexity, T(N), expresses the number of seconds required to complete the task. Similarly, the space complexity of the algorithm, S(N), denotes the number of memory words needed to execute the task (this quantity relates closely to the attribute cost). So for fixed execution interval and memory size, such expressions could give limits to the problem dimensions for an entire class of problems.

Besides extending this research project to match technological growth, the fundamental concepts of MUT could be used to investigate and characterize various mμP systems in detail. With enhanced throughput, improved reliability, increased system response, and modular expansion capabilities, the mμP arrangement deserves much attention. Still a number of problems exist which need careful research before implementation progress can occur. But MUT paves the path since it can facilitate analysis of both hardware and software.

Also, since the methodology described in this paper does not assume any specific hardwaredevices or software algorithms, it could aid a parametric study of either these two topics. Such investigations may lead to optimal design strategies for μP-based systems which consider both hardware and software tradeoffs.

Using the results of this research, designers of microprocessor-based systems will be able to synthesize advanced systems with improved performance. Additionally, by judicious use of these characteristics they will be able to open new application areas previously deemed unrealistic to solve with computer-based systems. Second, these results will benefit LSI chip designers by influencing what properties they will embody in the next generation of devices. Since some features assist computations while others detract, these new LSI devices will contain the desirable characteristics and minimize the action of the undesirable ones. Third, the results of this study will assist developers of μP-based systems by providing them with the methodology to investigate and characterize future systems as technology advances. Because the theoretic development begins at the axiom level, they may easily modify the methodology to fit their particular goals.

5. Acknowledgements

The authors wish to thank Prof. John J. Forsyth for his helpful suggestions dealing with the basic addition and multiply algorithms, Mr. James E. Siebert for his critical review of this work, and Miss Lisa Helmer for typing the manuscript.

REFERENCES

1. J. L. Baer, "Multiprocessor systems," IEEE Trans. Computers, vol. 25, pp. 1271-1277, Dec. 1976.

2. A. J. Weissburger, "Analysis of multiple-microprocessor system architectures," Computer Design, vol. 16, pp. 151-163, June 1977.

3. J. W. Bowra and H. C. Torng, "The modeling and design of multiple function-unit processors," IEEE Trans. Computers, vol. 25, pp. 210-221, Mar. 1976.

4. P. M. Russo, "An interface for multi-microcomputer systems," Proc. IEEE Fall COMPCON, pp. 277-282, Oct. 1976.

5. F. J. Hill and G. R. Peterson, Digital Systems: Hardware Organization and Design. New York: J. Wiley and Sons, 1973, p. 159.

6. J. Von Neumann and O. Morgenstern, Theory of Games and Economic Behavior. Princeton, N.J.: J. Wiley and Sons, 1967, pp. 15-31.

7. R. L. Keeney, "Quasi-separable utility functions," Naval Res. Log. Quart., vol. 15, pp. 367-390, 1968.

8. R. L. Keeney, "Multiplicative utility functions," Opns. Res., vol. 22, pp. 22-34, Jan. 1974.

9. W. Edwards, "How to use multiattribute utility measurement for social decision making," IEEE Trans. Sys. Man and Cybern., vol. SMC-7, pp. 326-340, May 1977.

10. S. L. Lillevik, "Computational Design Alternatives with MicroprocessorBased Systems," Ph.D. Thesis, Michigan State University, pp. 34-43 (1978).

11. D. G. Moursund and C. S. Duris, Elementary Theory and Application of Numerical Analysis. New York: McGrawHill, 1967, p. 64.

12. J. W. Cooley and J. W. Tukey, "An algorithm for the machine computation of complex Fourier series," Math. Computation, vol. 19, pp. 297-301, Apr. 1965.

13. W. D. Stanley, Digital Signal Processing. Reston, Virginia: Prentice-Hall, 1975, p. 270.