THE DESIGN OF A POLYMORPHIC ARITHMETIC UNIT*

Allan L. Lang
Florida Technological University
Orlando, Florida 32816

and

Bruce D. Shriver
University of Southwestern Louisiana
Lafayette, Louisiana 70501

Summary

This paper presents results which stem from a research effort concerned with the specification and design of arithmetic units which can execute nonstandard integer and floating-point arithmetic. An arithmetic unit is proposed whose characteristics are based on user specifications and subsequently is termed a Polymorphic Arithmetic Unit (PAU). The user binds the identity of the PAU by specifying the contents of various descriptors and semantic interpretation tables which the PAU accesses during its execution. This capability removes several of the restrictions found in commercially available arithmetic units and potentially assists in makink mathematically software portable.

Introduction

The design of machines for scientific computation is not keeping pace with the growing needs of the scientific community. Since the introduction of second generation computing systems and wide spread usage of floating-point arithmetic, there have been few improvements to arithmetic unit generality and organization. However, arithmetics are needed that more accurately resemble their intended use[1]. This paper discusses a study[2,3] which defines a versatile, user controllable arithmetic facility and which presents an operational definition of this facility which allows for alternative hardware/firmware/software implementations.

A General Description of the PAU

The Polymorphic Arithmetic Unit can execute nonstandard integer and floating-point arithmetic. The PAU has the user specifiable characteristics of:

1) variable length integer operands,
2) variable length coefficient and exponent fields for floating-point operands,
3) automatic conversions between integer and floating-point operands,
4) a selection of rounding strategies which can be employed during a floatimg-point operation, and during conversion between operand types,
5) augmented operand representations to include special representations for numbers too large or too small to be represented ($\pm\infty$), numbers too close to zero to be represented ($\pm\varepsilon$), and undefined (U),
6) integer and floating-point operations defined on combinations of representable and non-representable numbers (i.e. augmented representations), and
7) a variety of specifiable bases, β (β = 2, 3, 4,..., 16).

The operations the PAU support are the standard arithmetic operations +,-,*,/, and the relationals <,>,=. There are four different operand interpretations which the PAU can operate upon: single/extended precision integer (I1,I2), and single/extended precision floating-point (F1,F2). Each integer operand is composed of one sign-magnitude scalar, and similarly two sign magnitude scalars. Operands have representations which are specified by the Arithmetic Operand Descriptor (AOD). The AOD is the seven-tuple (i_1, i_2, e_1, f_1, e_2, f_2, β) where β is the common base, and the other elements of the tuple indicate the number of base β digits that are contained in the integer, exponent and fraction fields of each single and extended floating-point operands.

Interpretation and the Description Problem

An example of multi-level arithmetic interpretation can be seen in the realization of complex arithmetic on contemporary computer systems. This realization has the following structure:

1) the machine code generated by a compiler for complex arithmetic contains repeated references to the floating-point instruction set,
2) each floating-point operation is realized by a set of microinstructions, and
3) there exists hardware, known as a microprocessor, that executes the microinstructions.

The lowest level of interpretation, that is identical to the host machine, is called Level 0. The next level of interpretation, Level 1, is the microcoded realization of floating-point operations, and the last level of interpretation, Level 2, is the set of instructions that "drive" the floating-point unit. The characteristics of

*This work has been partially supported by NATO Grant No. 755, and by research funds provided by the University of Southwestern Louisiana

this multi-level system are:

1) each level is entirely supported by the immediate level below, i.e. each interpreter at Level N, N = 1, 2, 3,..., m, is realized by the instruction set of the interpreter at Level N-1.
2) any level N is finally realized by the host processor at Level 0,
3) each Level N, N = 0, 1, 2,..., m, of interpretation has a decoding procedure, instruction simulation procedure, initialization procedure, sequencing procedure, and stop procedure.

The systems designer needs to recognize from the many examples of multi-level interpretation is also another level of execution overhead. When an increase in system performance is required, significant gains can be made by:

1) decreasing the execution speeds of the lower levels of interpretation,
2) minimizing instruction simulation procedures, or
3) eliminating any of the intermediate levels.

The PAU was specified, designed, and implemented as a multi-level computer system. When approaching the problem of improving performance in arithmetic units, the multi-level interpreter approach may be used to:

1) serve as a unitfying feature so as to create more amenable arithmetic facilities for the user,
2) serve as a unifying structure so that speed increases may be made possible by implementing lower levels of interpretation in lower level technologies, i.e. "harder"; moreover,
3) separate an interpreter into major subdivisions so that the basic functions of an interpreter may be examined with respect to cost/performance.

Since the interpretation process ends at the host level, the realization of higher level applications via multi-level interpreters is only possible in a cost/effective sense when the host is powerful enough to support the interpretation process.

Combining Arithmetic Units

The PAU is composed of seven arithmetic units that have often functioned in the past as independent entities within a computing system. The arithmetic units that function within the PAU in a coordinated fashion are discussed below.

Yohe Arithmetic Unit, YAU

The YAU[4] is a working model for rounded floating-point arithmetic where the user can specify one of the five rounding options. The rounding strategies supported by the YAU are the symmetric rounding 1) truncation, 2) away from zero, 3) closest, and the non-symmetric roundings, 4) upward directed , and 5) downward directed. The floating-point representation is variable base, and has variable field lengths for the exponent and fraction. The YAU also supplies a mathematically consistent treatment for procedures that perform rounded floating-point operations. Each arithmetic operation is a function defined on representable floating-point numbers, and the result of each operation is a representable number. Figure 1 is a unit diagram for the YAU.

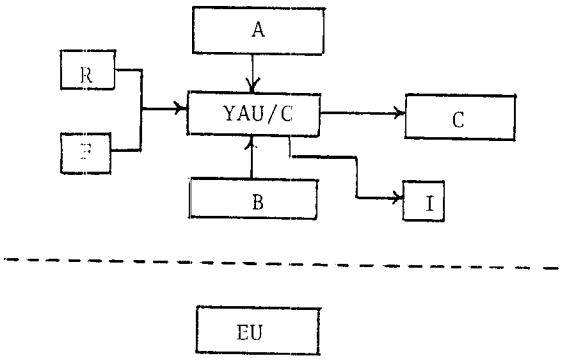


Figure 1

YAU Unit Diagram

where,

R	- a rounding strategy selector,
F	- arithmetic function selector,
A,B	- fixed-length, floating-point operand registers,
C	- a destination register which contains the floating-point result,
YAU/C	- a control unit that uses a subordinate machine (EU) to execute primitive operations that constitute F,
I	- an indicator to denote exceptional conditions,
EU	- an execution unit which supports the YAU/C by serving as the host machine (either real or virtual).

One extention to the YAU would be the incorporation of non-representable numbers into the definition of YAU arithmetic. This is accomplished in the PAU by the utilization of another arithmetic unit, the Neely Arithmetic Unit.

Neely Arithmetic Unit, NAU

Neely[5] provides a description of an augmented floating-point and integer number system and proposes it as an enhancement for arithmetic units. The augmented number system would enable arithmetic units to provide an interpretation for all possible arithmetic operations on elements in the augmented set, and provide a consistent treatment for the irregularities that occur during arithmetic operations, e.g. under/overflow.

The range of floating-point and integer numbering systems is extended by defining the following concepts:

1) ∞i_j, ∞f_j are intervals which represent numbers that are too large or small to be represented,
2) ϵ_j are intervals which represent floating-point numbers that are too close to zero,
3) 0_{f_j} is the set of floating-point zeros, and
4) U_{i_j}, U_{f_j} are the set of undefined numeric values.

In the above, i denotes integer, and f denotes floating-point; j indicates that each concept has a single and extended precision counterpart.

Operands within the PAU have tag bits associated with them which specify: (a) if the operand is a single or extended precision integer or floating-point number, and (b) if the operand is to be interpreted as a representable number or a member of one of the above augmented sets. Tagged[6,7] data demands more memory space to be allocated so that the hardware can determine the content of data represented in memory. With the descreasing cost of computer memory, tagged architectures are becoming more cost/effective, so in general, the designers of future computing systems (arithmetic units included) have the option of utilizing the tagged data concept. The lack of this option in the past could explain the relative unavailability of Needly type arithmetic units.

Figure 2 is a unit diagram for the NAU:

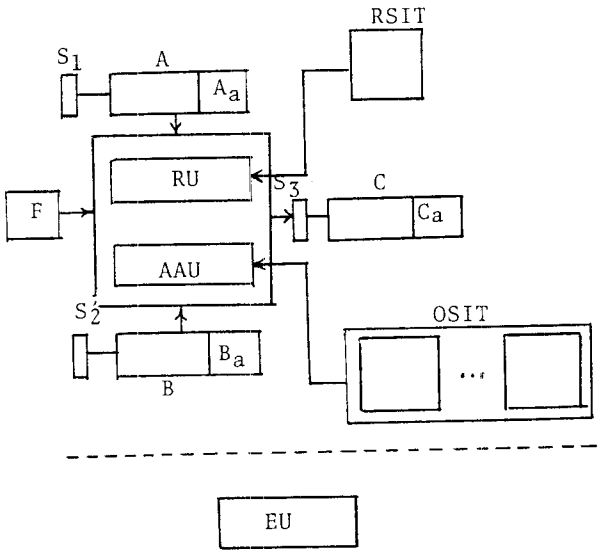


Figure 2

NAU Unit Diagram

where,

F - a function selector that specifies the mode (integer or floating-point), and the arithmetic operation to be performed,
A,B - fixed-length input registers which contain integer or floating-point operands,
C - a destination register which contains the result,
A_a,B_a,C_a - coded operands of the augmented set found in the low order set of contiguous bits,
S_1,S_2,S_3 - indicators that signal the presence of an augmented number (in A_a,B_a,C_a respectively),
OSIT - an Operand Semantic Interpretation Table,
RSIT - a Recovery Semantic Interpretation Table,
RU - a functional unit called the Recovery Unit which specifies a result when normal operation produces an irregularity,
AAU - a functional unit called the Augmented Arithmetic Unit that uses the OSIT to determine the result,
EU - an Execution Unit that realizes the functions of the AAU and RU.

The NAU is not an independent unit; it is actually an extension to a typical arithmetic unit that "smooths over" hardware irregularities that occur in arithmetic units. The Recovery Unit is what provides a user specifiable assignment to irregularities that occur during normal arithmetic operations while the AAU handles irregularities expressed as operands entering the arithmetic unit. For example, if a and b are two positive representable floating-point numbers and c←a*b produces an overflow in the YAU, then the Recovery Unit supplies the special default symbol $+\infty$ as the result for c. Operand c is now tagged and once c re-enters an arithmetic system as an operand for an operation, the NAU is activated to execute the operation which the YAU cannot perform.

The NAU is a table driven unit. An Operand Semantic Interpretation Table (OSIT) is the major input to the operation of the unit. An OSIT for floating-point multiplication is shown in Figure 3. The elements in the OSIT are references to the appropriate semantic procedure to be used for the indicated operand combination. Examples of these semantic procedures are listed in Table 1. Each operation that is supported by an arithmetic unit (i.e. +,-,/,*,<,>,=) has a corresponding operation in the NAU and a separate OSIT.

Selection of the elements (semantic procedures) for the OSIT is the major mathematical issue to be considered in the unit. The major design philosophy for the NAU is to make the OSIT accessible to the user so that the user can specify the appropriate interpretation.

a \ b	-F	+F	$-\infty_f$	$-\varepsilon$	0_f	$+\varepsilon$	$+\infty_f$	U_f
-F	*	*	$+\infty$	$+\varepsilon$	0_f	$-\varepsilon$	$-\infty$	U_f
+F	*	*	$-\infty$	$-\varepsilon$	0_f	$+\varepsilon$	$+\infty$	U_f
$-\infty_f$	$+\infty_f$	$-\infty_f$	U_f	U_f	0_f	U_f	U_f	U_f
$-\varepsilon$	$+\varepsilon$	$+\varepsilon$	U_f	$+\varepsilon$	0_f	0_f	0_f	U_f
0_f	0_f	0_f	0_f	0_f	0_f	0_f	0_f	U_f
$+\varepsilon$	$-\varepsilon$	$+\varepsilon$	$-U_f$	$-\varepsilon$	0_f	$+\varepsilon$	$+U_f$	U_f
$+\infty_f$	$-\infty_f$	$+\infty_f$	U_f	U_f	0_f	$+U_f$	U_f	U_f
U_f	U_f	U_f	U_f	U_f	U_f	U_f	U_f	U_f

-F negative floating-point numbers

+F positive floating-point numbers

* specifies processing to be done by a subordinate arithmetic unit

Figure 3

Operand Semantic Interpretation Table for Augmented Floating-Point Multiplication c←a*b

Table 1

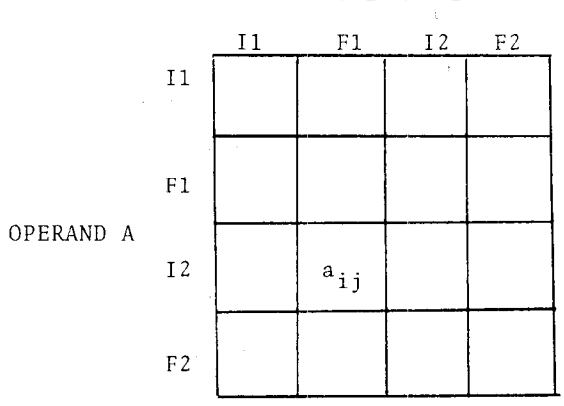
AAU Semantic Interpretations

INTERPRETATION ASSIGNMENT	CODE
C ← OPERAND A	(A)
C ← OPERAND B	(B)
$C_a \leftarrow -\infty$	$(-\infty)$
$C_a \leftarrow -\varepsilon$	$(-\varepsilon)$
$C_a \leftarrow 0_f$	(0_f)
$C_a \leftarrow +\varepsilon$	$(+\varepsilon)$
$C_a \leftarrow +\infty$	$(+\infty)$
$C_a \leftarrow U_f$	(U_f)
I ← TRUE	(T)
I ← FALSE	(F)
I ← STOP MACHINE	(SM)

Mixed-Mode Filter Unit, MMFU

Another arithmetic unit that is included within the definition of the PAU is the Mixed-Mode Filter Unit[2]. A MMFU is a unit that exists in a software environment on most arithmetic facilities that support both integer and floating-point operations on mixed operands. It is the function of the MMFU to preprocess operands of different representations so that the standard operations of integer or floating-point arithmetic may be performed. To do this preprocessing, operands are converted to a standard, predetermined type. Problems arise during conversion. For example, how is a floating-point number to be converted to an integer if its magnitude is larger than the maximum representable integer? Again, if a floating-point number is to be converted to an integer where the whole number part of the floating-point number is representable by an integer, but there also exists a fractional part, must one employ an optional rounding strategy? Finally, a zero integer is to be converted to a normalized floating-point number, what value must be chosen?

The sixteen different types of operand combinations that can exist upon entering the unit are shown in Figure 4 which is called a Mixed Mode Semantic Interpretation Table (MMSIT). The table entries for Figure 4 indicate which standard operation is to be performed for the Operand A, Operand B combination. This means that both Operand A and Operand B will be converted to the same data type as specified by the table element.



where $a_{ij} \varepsilon \{I1, F1, I2, F2\}$
$i, j = 1, 2, 3, 4$

Figure 4

A Mixed-Mode Semantic Interpretation Table

Rather than have an operation for each operand combination, i.e. an F1 OP_{12} I2, and an F2 OP_{21} I1, this decision table mechanism may be utilized; hence the need for only one add operator. This type of operator is called a polymorphic operator.

By allowing the user to control the selection of the table element in the MMSIT a general approach must be analysed for the converting process. There are sixteen possible conversions, four of which are the identity conversions. These conversions are called P functions. Each P_i has situations where the mapping must be specified by the user. In the description of each P_i when the user can specify a mapping assignment "user specifiable" is indicated. The possible assignment that can be made, when one of these situations occur, is an element from the set ∞_c, ε_c, 0_f, U_c, where c = I1, I2, F1, or F2; f = F1 or F2; or a STOP-MACHINE indicator may be specified which aborts MMFU execution. When the user specifies part of a mapping, a table element in the Conversion Semantic Interpretation Table (CSIT) is altered to indicate to the PAU the desired interpretation.

When rounding is performed in a P_i, the rounding is in the same context as those strategies discussed for the YAU, and is controlled by the user.

Figure 5 shows how conversion function P_2 is parameterized to allow a flexible interpretation of the P_2 function.

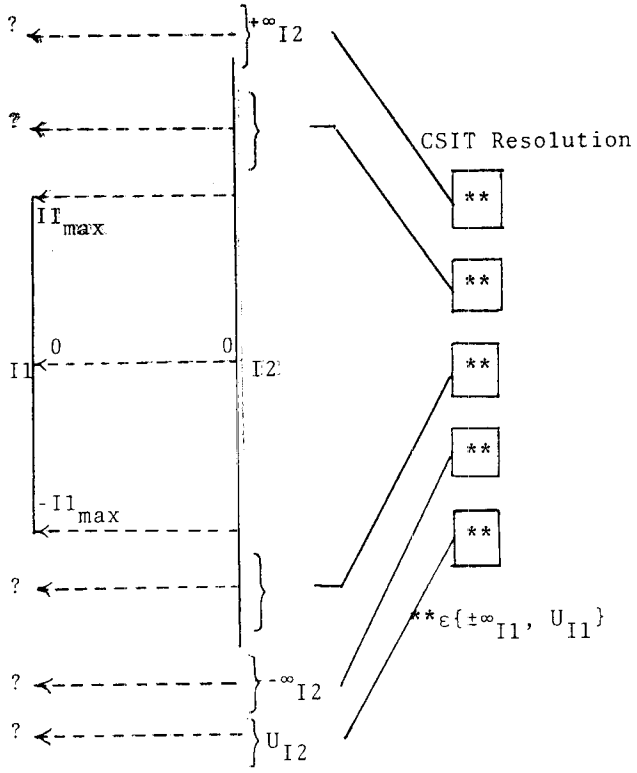


Figure 5

CSIT Specification for P_2: I2→I1

Figure 6 is a unit diagram for the MMFU,

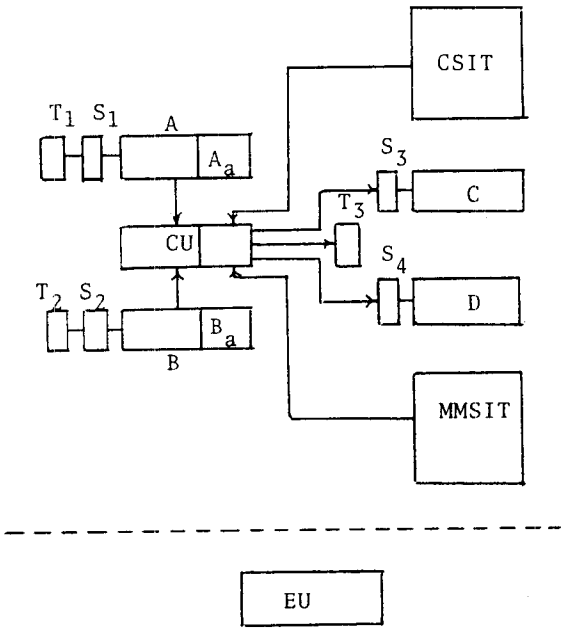


Figure 6

MMFU Unit Diagram

where,

A,B,C,D	- operand storage registers,
T_1,T_2,T_3	- tag information denoting the operand type,
S_1,S_2,S_3,S_4	- signal information denoting that an operand (A,B,C,D) is in either representable form or augmented form,
CU	- a functional unit that performs the conversion function as indicated by the functions P_i (i = 1, 2,..., 16)
EU	- the Execution Unit which executes instructions specified by the CU,
MMSIT	- the semantic interpretation table which specifies the arithmetic type that A and B are to be converted to, and
SCIT	- the semantic interpretation table which specifies assignments for nonstandard mappings in a P function.

The Integer Unit, IU

The Integer Unit[2] executes standard operations on sign-magnitude operands that are defined in the AOD by the i_1, i_2, and β parameters. The IU is able to perform variable length integer operations by utilizing a subordinate machine, the Basic Integer Unit which executes fixed-length integer operations.

Interpretation and the PAU

The PAU is composed of three levels o interpretation. Figure 7 shows these level

and identifies the functional units associated with each.

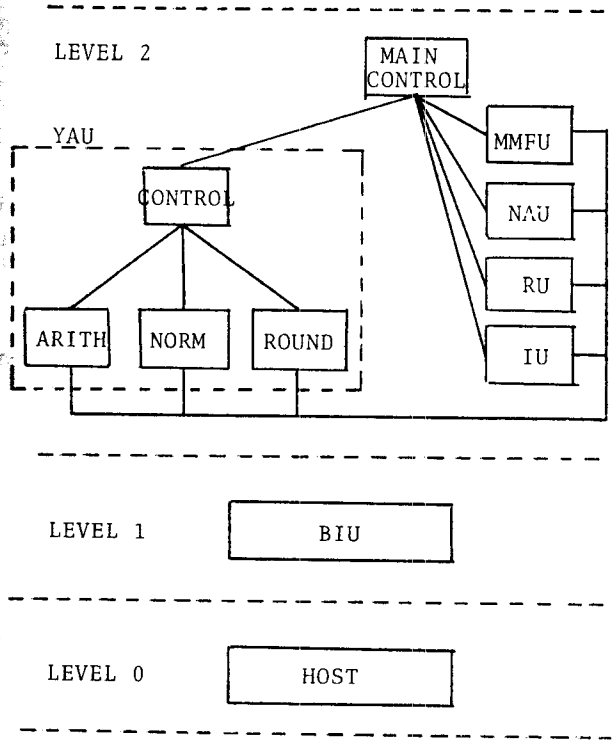


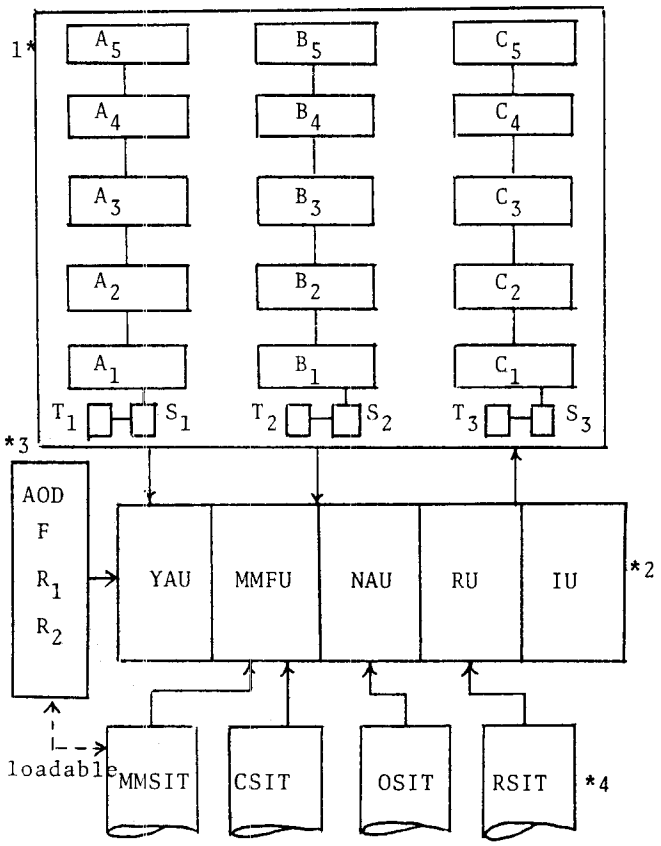
Figure 7

PAU Data Paths
Between Functional Units and Levels

The flow of data is directed by the MAIN CONTROL. The MAIN CONTROL is passed the data that it needs by the user or higher level arithmetic unit. The MAIN CONTROL first passes the input data to the MMFU that performs the necessary conversion (if one is needed) transforming the input data to a common data format. When control is released by the MMFU, MAIN CONTROL proceeds to send the input data to the NAU if one or more of the flags (S_1,S_2) are set. If no flags are set, then MAIN CONTROL sends the input data to the YAU where a standard operation takes place. If the YAU cannot perform the operation due to the result being unable to fit in the predetermined format, then the RU is invoked to supply an interpretation for the YAU error. If the standard operation to be performed is an integer one, then the input data is sent to the IU, where the operation is performed, and again, if an integer error occurs the RU is invoked. The result from the YAU, NAU, or IU whichever may be the case is returned to MAIN CONTROL.

Using the PAU

Figure 8 shows the virtual arithmetic unit that is presented to the user. This is the arithmetic unit that a high level language translator could utilize by generating arithmetic and control instructions.



*1 operand registers
*2 processing units
*3 processing parameters
*4 semantic interpretation tables

Figure 8

PAU Unit Diagram

The block diagram for the PAU is divided into four groups:

1) operand registers,
2) processing units,
3) processing parameters, and
4) semantic interpretation tables

There are several tasks that must be completed before PAU execution can begin. These tasks mainly concern the loading of needed information located in the operand registers, processing parameters, and semantic interpretation tables. The following is a sequence of data transfer that occurs before PAU execution:

1) load the AOD elements - this specifies to the PAU the operand field lengths and radix.
2) load the F register - this specifies one of the standard PAU functions (-,-,*,/,<,>,=),
3) load R_1, R_2 which is the rounding strategies for the YAU and MMFU, respectively,
4) load the two input operand register stacks (A, B), indicating operand

type by separately loading the tag registers (T), and indicating the presence of a Neely representation by loading the signal registers (S),

5) load each of the Semantic Interpretation Tables (MMSIT, CSIT, OSIT, and RSIT); note, since there are four different arithmetic types, and seven different operations, there are 28 different interpretation tables for the OSIT.

PAU Operational Definition

The operational definition of the PAU is given in its entirety in reference 3. The PAU is composed of several functional units. Each of these functional units possess three major organizational characteristics. They are:

1) a unit diagram - the operational definition is composed of elementary functions defined on fixed-length resources that are defined in the unit diagram. Each register is a binary register that can be read or written to by other functional units or by internal procedures defined within the same functional unit. (See Figure 9)
2) a unit control procedure - each functional unit has a separate procedure called the CONTROL procedure. There are three tasks associated with this procedure 1) loading registers, 2) selecting the appropriate instruction execution procedure, and 3) storing registers. (See Figure 10)
3) instruction execution procedures - this procedure utilizes functional units at the next lowest level of interpretation to perform a virtual instruction. (See Figure 11)

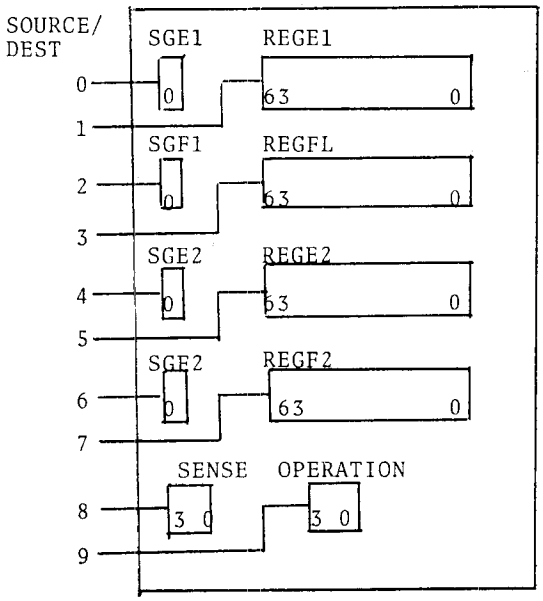


Figure 9

YAU Unit Diagram

A brief introduction to the technique, which was used to describe the PAU, can be given by considering an overview of a description of one PAU component, the YAU. Consider the unit diagram of the YAU as shown in Figure 9.

One instruction within the YAU is floating-point multiply. To realize this instruction the YAU must be activated, and then, once activated, select the multiply procedure (MUL). Figure 10 shows the Control procedure that activates floating-point multiplication. Note that the description technique implies control via an ALGOL-like language structure. Figure 11 demonstrates how two primitive functions within multiplication are simulated with the use of the Basic Integer Unit (BIU). Figure 11 shows how arguments are sent to the BIU, and how the result is returned. Of course, this example is over-simplified, but does illustrate the needed detail for arithmetic unit specification.

```
procedure YAU-ARITHMETIC CONTROL;
 begin
      fetch SGE1,
      fetch REGE1,
      fetch SGF1,
      fetch REGF1,
      fetch SGE2,
      fetch REGE2,
      fetch SGF2,
      fetch REGF2,
      fetch OPERATION;
 end;
 begin
      if OPERATION (2:0) = 0 then ADD else
      if OPERATION (2:0) = 1 then SUB else
      if OPERATION (2:0) = 2 then MUL else
      if OPERATION (2:0) = 3 then DIV else
      SYSTEM-ERROR;
 comment on SYSTEM-ERROR post error and
         return control;
 end;
 begin
      send SGE1,
      send REGE1,
      send SGF1,
      send REGF1,
      send SENSE;
 end;
end YAU-ARITHMETIC CONTROL;
```

Figure 10

YAU-Unit Control

```
procedure MUL;
 begin
 comment add exponents;
      store SGE1  in   SGN1          of BIU;
      store REGE1 in   OPERAND1A     of BIU;
      store SGE2  in   SGN2          of BIU;
      store REGE2 in   OPERAND2      of BIU;
      store 0     in   OPERATION     of BIU;
      start-up BIU wait "+" cycles;  of BIU;
      load  REGE1 from OPERAND1A     of BIU;
      load  SGE1  from SIGN1         of BIU;
 end;
```

```
begin
comment multiply fractions;
     store SGF1   in    SGN1        of BIU;
     store REGF1  in    OPERAND1A   of BIU;
     store SGF2   in    SGN2        of BIU;
     store REGF2  in    OPERAND2    of BIU;
     store 2      in    OPERATION   of BIU;
     start-up BIU wait  "*" cycles;of BIU;
     load  SGF1   from  SGN1        of BIU;
     load  REGF1  from  OPERAND1B   of BIU;
end;
end MUL;
```

Figure 11

YAU-ARITHMETIC MULTIPLY

Once the arguments reach the BIU, definitions are given for sign-magnitude integer operations, and these definitions are applied to fixed-length registers in the unit diagram for the BIU.

Simulation

The entire PAU was simulated[7] in PL/1 within a period of about six months. The simulator length is approximately 70% is directly related to realizing the PAU operational definition.

The testing phase of this study completed a necessary feedback inspection. Each of the following was experienced as a result of the testing phase:

1) errors were found and corrected in the PAU operational definition,
2) the operational definition was reduced 20% by finding commonly used functions, and realizing them in lower levels of interpretation, and
3) the resulting simulator proved to be a useful tool which one could use to become familiar with the PAU operational definition, and concepts.

The simulator is an interactive program that allows for sample data to be run through the system, and permits the display of the contents of registers in the system from any level.

Cost/performance studies are supported by the simulator.[8] The simulator possesses a logging feature which traces data flow throughout the system. This feature allows the researcher to analyze the set of instructions executed at each level of interpretation. Status information can indicate bottle necks, or frequently used instructions whose performance is critical to the attainment of a desired cost/performance ratio.

References

1. Shriver, Bruce D. (1973). A Small Group of Research Projects in Machine Design for Scientific Computation. Computer Science Department, University of Aarhus, Aarhus, Denmark, DAIMI PB-14.

2. Lang, Allan L. (1975). The Design of a Polymorphic Arithmetic Unit: A Case Study of a Multi-Level Interpretive Computing System. Department of Computer Science, University of Southwestern Louisiana, Lafayette, Louisiana.

3. Lang, Allan L. (1975). The Polymorphic Arithmetic Unit: An Operational Definition. Department of Computer Science, University of Southwestern Louisiana, Lafayette, Louisiana.

4. Yohe, J. M. (1973). Roundings in floating-point arithmetic. IEEE Transactions on Computers. C-22, 6.

5. Neely, Peter M. (1972). On conventions for systems of numerical representations. Proceedings ACM 72. Association for Computing Machinery, New York.

6. Feustel, E. A. (1972). The Rice Research Computer -- a tagged architecture. AFIPS Spring Joint 1972 Conference Proceedings. AFIPS Press, Montvale, New Jersey.

7. Reuter, Eric, Lang, Allan. (1975). A Users Manual for the Simulated Polymorphic Arithmetic Unit. University of Southwestern Louisiana, Lafayette Louisiana.

8. Shriver, Bruce D., Lange, Allan L., Reuter, Eric. (1975). A Case Study of a Simulator for a Multi-Level Interpretive Computing System. University of Southwestern Louisiana, Lafayette Louisiana. (forthcoming)