THE UNRAU

a Unified Numeric Representation Arithmetic Unit*

by

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

and

Peter Kornerup
University of Aarhus**
Aarhus, Denmark

Abstract

A companion paper entitled[1], "A Unified Numeric Data Type in Pascal", proposes the substitution of the standard data type real of the language Pascal with a unified data representation termed numeric. The numeric data type can represent a variety of arithmetic operands such as integers, normalized floating point numbers, and centered-radius intervals.

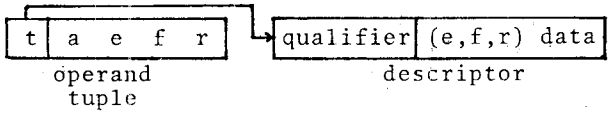
This paper describes an arithmetic unit which is capable of executing the standard arithmetic operations (addition, subtraction, multiplication, and division) on pairs of operands specified to be of the numeric data type. This arithmetic unit, called the UNRAU - Unified Numeric Representation Arithmetic Unit, supports operations on operands externally represented as 5-tuples (t, a, e, f, r). The UNRAU provides for automatic conversion among the various data types and can also be used to perform an explicit conversion on a single operand.

It is intended to implement the UNRAU on a dynamically microprogrammable microprocessor to determine what host facilities are required to efficiently realize such an arithmetic unit and to experiment with the high level language support of such a unit.

1. An External View of the UNRAU Operands

The UNRAU is an arithmetic unit which is capable of executing standard arithmetic operations upon operands which may be integers, normalized or unnormalized floating point numbers and center-radius intervals. The operands are "tagged" to indicate which particular type of number they are to be interpreted as representing. There is only one set of arithmetic operations and not four sets, i.e., integer arithmetic operations, normalized floating point operations, etc. As such, the operations are referred to as "polymorphic" operations.

Operands are represented, external to the UNRAU as 5-tuples, (t,a,e,f,r). The t field (tag-field) is a pointer to a descriptor which contains data concerning the representation. This is shown symbolically as:



The descriptor contains a "qualifier" field which specifies how the operand tuple is to be interpreted and a "(e,f,r) data" field which specifies the position and sizes of the e,f, and r fields of the operand. The (e,f,r) data field will be referred to as the "format data".

The (e,f,r) triple is interpreted in one of the four ways shown in Table 1 as specified by the contents of the qualifier field.

Name of representation (qualifier)	Interpretation of (e,f,r) as a "normal" operand
fixed:	the value represented is f, where the f-field is interpreted as a sign-magnitude integer, the e and r fields are not used.
normalized:	the value represented is $f*2^e$, where f and e fields are interpreted as sign-magnitude integers. The f-field is assumed to be normalized, i.e., the most significant bit of the f-magnitude fields is a 1 if the field is non-zero. The r-field is not used.
unnormalized:	the value represented is $f*2^e$, where the interpretation is the same as for normalized numerics. The f-field is not normalized, and the least significant bit of f is considered as being the last "correct" bit. The r-field is not used.
centered:	the "value" represented is the interval $(f-r, f+r)*2^e$, where e and f are sign-magnitude integers, and r is the magnitude of the unsigned integer value of the r-field. The f-field is not normalized

Table 1

Normal Operand Interpretations

*The work has been partially supported by NATO Grant No. 755 and The Danish Research Council Grant No. 1546/511.
**Currently at the University of Southwestern Louisiana.

A non-representable number may result from the execution of an arithmetic operation or whenever a mapping into the finite precision representation (as specified by the format data) is impossible. The a-field of the operand tuple provides an escape bit to signal the presence of such non-representable numbers. If the a-field has the value "normal", it indicates that the number is representable and should be interpreted as shown in Table 1. However, if the a-field has the value "augmented", indicating that a non-representable value has been introduced, then the (e,f,r) triple will be interpreted as one field which can assume the following values: negmax, negmin, posmin, posmax, undef. These values are interpreted as shown in Table 2 for each of the four types of operands.

qualifier / nrv	fixed	normalized & unnormalized & centered
negmax	too large neg. value	too large neg. value(exp. overflow)
negmin	not applicable	too small neg. value(exp. underflow)
posmin	not applicable	too small pos. value(exp. underflow)
posmax	too large pos. value	too large pos. value(exp. overflow)
undef	undefined val.	undefined val.

Table 2.

Interpretation of (, f, r) triple as an "nrv" operand

2. An External View of the UNRAU Operations

A block diagram of the UNRAU is shown in Figure 1. It has an internal stack store of finite length and can be used to directly evaluate arithmetic expressions presented to it in reverse polish form (postfix notation).

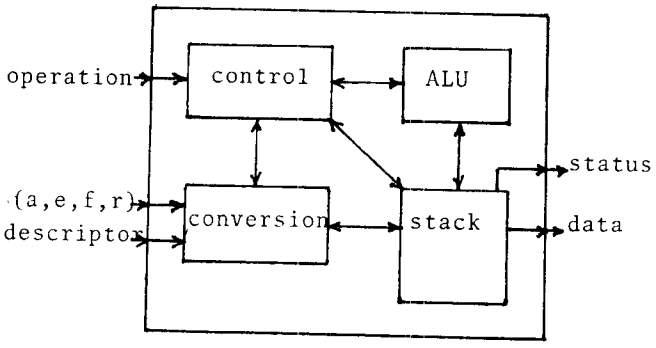


Figure 1

A Block Diagram of the UNRAU

When an operand is to be sent to the UNRAU, its t-field is decoded to fetch the descriptor associated with that operand and the data actually presented to the UNRAU is the (a,e,f, r) tuple and the descriptor. The UNRAU can execute the operations shown in Table 3.

Operations	Input	Output: Data	Output: Status
Push	(a,e,f,r) & descriptor		os,as
Pop	descriptor	(a,e,f,r)	os,as
Store	descriptor	(a,e,f,r)	os,as
Add			os,as
Subtract			os,as
Multiply			os,as
Divide			os,as
Negate			
Absolute			
Compare		R data	os
Characteristics		C data	
Convert	descriptor		as
Duplicate			os
Initialize			
Interchange			

Table 3

UNRAU Operations

The Push, Pop, and Store are the input/output operations associated with the UNRAU. Push is used to load operands onto the stack. Store presents the top of the stack element to the output lines, while Pop is a store followed by a "popping" off of the top of the stack element. The dyadic arithmetic operations operate on the two top stack elements, pop them, and store the result on the top of the stack. Monadic operations operate on the top of the stack. Duplicate pushes a copy of the top stack element onto the stack while interchange swaps the two top stack elements. Convert is discussed in Section 3 and the Compare operation and its associated R data and the Characteristics operation and its associated C data are discussed in Section 4. The Initialize operation is used to set the UNRAU to a specified state: the stack is emptied and all status lines are cleared.

The two classes of status data which are available externally at the end of the execution of each UNRAU operation. These are identified as "os" and "as" in Table 3. "Os" refers to operational status and indicates (1) stack overflow, (2) stack underflow, or (3) the Compare operation was executed on augmented data. Stack overflow can arise as a result of the Push and Duplicate operations, while stack underflow can result from the Pop operation, Store operation, dyadic operations with only 1 element in the stack, or unary operations with no elements in the stack. "As" refers to augment status and indicates (1) an augmented form has been pushed onto the stack, (2) a non-representable value has arisen during the computation, or (3) an augmented form is being stored or popped off the stack. Thus there are 6 independent status lines available at the end of

each UNRAU operation; they are summarized in Table 4.

Status Class	Status
os	so, stack overflow su, stack underflow ra, relational involving augmented numbers
as	pa, push augmented number onto stack nrv, non-representable value has arisen spa, an augmented number has arisen during a Store or Pop Operation

Table 4

UNRAU Status Data

Whenever a non input/output operation has an operand which is an augmented form, the result undefined is placed on the top of the stack.

3. An Internal View of UNRAU Operands

Whenever data is presented to the UNRAU along with its descriptor, it is converted into an internal UNRAU representation before pushing it onto the top of the stack. This internal UNRAU representation corresponds to a bound format "maximal accuracy" representation. The data in the descriptor is used during the conversion process and then disregarded. The operand on the top of the stack is then represented by the 5-tuple (qualifier, a,e,f,r).

All intermediate results are kept internally on the stack in the "maximal accuracy" format which means that no format specifications are needed for temporary results. Since the operations are polymorphic, any necessary implicit conversions are delt with by the UNRAU. If, on the other hand, the user wishes to force stack contents into specific format restrictions, an explicit conversion operation is provided. When the Convert operation is to be executed, a descriptor must be presented to the UNRAU as data. Whenever a Pop or Store instruction is issued, descriptor information is needed so that the top of stack is converted from its internal type and form, to the type and form required.

The internal representations have been chosen such that all fields are expanded compared to the maximal external representations. Hence arithmetic operations can take place, and intermediate results be placed on the stack, in an extended precision compared to that of the external representation. Furthermore, the internal representation carries a number of guard digits, in the planned implementation two binary digits.

Results of the arithmetic operations are truncated when placed on the stack, if the result is of normalized or unnormalized types. When numbers are packed into an external representation, a standard rounding takes place (rounding upwards when the leading cut-off bit is a one). Because of the guard-digits provided, this double rounding strategy will only under very rare circumstances give an incorrect rounding.

Center-radius represented intervals are treated differently since any mapping into this representation has to guarantee that the actual "value" lies within the interval represented. Hence, any change in the value representing the center has to be reflected in the value representing the radius. The center and the radius carry guard-digits which means that only during quite long expressions can accumulation of such rounding effects add up to significant increases of the radius.

4. An Internal View of the UNRAU Operations

The dyadic operators operate on the two top stack elements. In order for the operation to be performed, both operands must be of the same type. The internal representations and interpretations of the operands are different only if their qualifiers are different. If the two do not have the same qualifier, one of the operands is converted to the type of the other and this will also be the type of the result. The types are ranked in a priority sequence corresponding to their qualifiers as follows:

(lowest) 1) fixed
2) normalized
3) unnormalized
(highest) 4) centered

The type of highest priority is chosen to be the common operand type and the type of the result.

The principles of conversion between internal representations of representable values are given below:

From fixed into all other representations:
The f-field is copied, e and r fields are set to zero. If the conversion is into normalized, the f-field is left shifted until it is normalized, and the e-field is adjusted accordingly.

From normalized into fixed:
The nearest integer value is chosen (if representable, otherwise an augmented form is introduced).

From normalized into unnormalized and centered:
The e and f-fields are copied, the r-field is set to zero.

From unnormalized into fixed:
The nearest integer value is chosen (if representable, otherwise an augmented form is introduced).

From unnormalized into normalized:
By standard normalization.

From unnormalized into centered:
The new fractional part will be 2 * f if possible, otherwise f; and the r-field will be set to 1.

From centered into fixed or normalized:
Gives the result of the center interpreted as an exact quantity.

From centered into unnormalized:
The f-field is shifted entier $(\log_2 r)+1$ places to the right, the e-field is adjusted accordingly, and the r-field is set to zero.

The standard dyadic arithmetic operations can then, after the initial conversions, operate on stack elements of the same type and representation. The implementation of these is then straightforward[+], but it is worth noticing that substantial savings in the operation time for the multiplication and division can be achieved by operating only on the significant digits. The sign magnitude representation allows for an easy determination of the part of a given field which contains the essential parts of the number represented, which then are the parts to be used in standard sequential implementation of those operations. Especially when realizing the interval arithmetic these savings may be essential, since the radius usually only will contain a few significant bits. The arithmetic operations may cause augmented forms to be introduced, which will be delivered on the stack in a special representation, and information about this will be presented in the "as" status.

Any arithmetic operation involving an augmented form as an operand will give as the result an augmented form with the non-representable value "undef". This implies that other non-representable values can only be the immediate result of underflow or overflow situations either in arithmetic operations or in conversions in connection with assignments or explicit conversions.

The language support of UNRAU provides for an explicit type conversion operator, which is implemented as a monadic, top of stack, operator convert. Given the descriptor of an external representation, the operator takes the top of stack number and converts it into the corresponding external representation (including the rounding) and then expands it into an internal representation, of the type requested.

The Compare operation computes all the relations $<$, $\leq$, $=$, $\neq$, $\geq$, $>$ between the two top elements of the stack, and then pops these two elements off the stack. The result of the Compare operation, the R-data, is a 6-tuple (lt,le,eq,ne,ge,gt) where each element has a value true if the (top stack) R (top stack-1) holds, else the value is false. The environment external to the UNRAU can, of course, test any elements of the R-data 6-tuple at the end of the Compare operation. Implicit type conversions may be required to complete the Compare operation and are performed in the same manner as a dyadic operation requires.

[+]An implementation scheme for the arithmetic operations on center-radius represented intervals is presented in the appendix.

The Compare operation is also defined when the operands are intervals. The interpretation of the R-data elements is as follows, where A and B are intervals:

lt: $A \cap B \neq \phi \wedge (\forall x \in A \wedge \forall y \in B \Rightarrow x<y)$
le: $A \cap B \neq \phi \wedge (\exists x \in A \wedge \forall y \in B \Rightarrow x<y)$
eq: $A \subseteq B$
ne: $A \cap B = \phi$
ge: $A \cap B \neq \phi \wedge (\exists x \in A \wedge \forall y \in B \Rightarrow x>y)$
gt: $A \cap B \neq \phi \wedge (\forall x \in A \wedge \forall y \in B \Rightarrow x>y)$

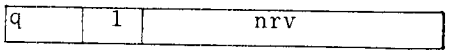
The Characteristics operation yields data concerning the nature of the top of the stack element. The Characteristics operation examines the stack top element and presents a 12-tuple called the C-data. The elements of the C-data, <f, u, n, c, -max, -x, -min, 0, +min, +x, +max, und>, give the type of the number, i. e., fixed, unnormalized, normalized, or centered, and sets one of the other flags to indicate if the number is negative (-x) positive (+x), undefined, etc. The environment external to the UNRAU can test any elements of the C-data tuple at the end of the Characteristics operation.

5. Summary and Some Related Issues

There are two major developments which are currently taking place concerning the UNRAU. First, a firmware implementation of the UNRAU itself and secondly, the development of language support which will give user accessibility to the UNRAU. The latter issue is dealt with in the companion paper[1], "A Unified Numeric Data Type in Pascal" and will not be discussed here.

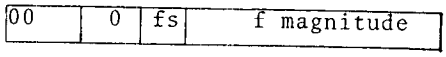
The implementation will be on the dynamically microprogrammable MATHILDA[2] machine. In the planned implementation on MATHILDA the stack will be kept in a 64-bit wide local store. One possibility under consideration for operand representation is to have each entry in the stack occupy 1, 2 or 3 words, depending on the values of q and a:

Augmented forms:

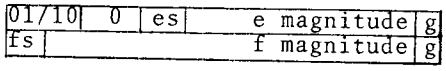


Normal forms:

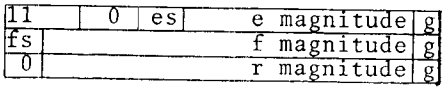
fixed:



normalized & unnormalized:



centered:



In the above diagrams, g represents guard digits. The arithmetic operations and type conversion operators use operands on the stack in these representation, and deliver their result in the same representation. The Push

instruction packs an internal representation into the external representation, except for the tag field, which is supplied by the environment.

It is anticipated that the interface between the UNRAU and the language (e.g. the prefetching of descriptors to deliver to the UNRAU) will also be implemented in firmware on MATHILDA.

References

Kornerup, P., "A Unified Numeric Data Type in Pascal", this conference.

Kornerup, P., and Shriver, B. D., "An Overview of the MATHILDA System," SIGMICRO Newsletter, January 1975, Vol. 5, No. 4.

Appendix

A. The implementation of arithmetic operations on interval numbers

Since the center radius representation is not the most commonly known among interval representations a description of the algorithms will be given.

It is assumed that the arithmetic operations compute their results as a tuple (γ,α,ρ) representing the interval as:

$$[\alpha-\rho,\ \alpha+\rho]*2^{\gamma}$$

where (γ,α,ρ) then has to be mapped into the interval representation (e,f,r).

e: sign magnitude int, $0\le |e| \le 2^{m-1}-1$
f: sign magnitude int, $0\le |f| \le 2^{m-1}-1$
r: unsigned integer, $0\le |r| \le 2^{k}-1$

A.1 Mapping into the representation

The mapping of an interval $[\alpha-\rho,\alpha+\rho]2^{\gamma}$ where α and ρ are binary numbers, possibly with a fractional part, satisfying:

$$\min(|\alpha|,\rho) \ge 1 \quad \text{or} \quad \alpha=\rho=0$$

is given by the following algorithm:

```
s := sign (α); α:= abs(α);
while   α ≥ 2^(n-1) ∧ ρ ≥ 2^(k-1) do
begin   ρ := ρ/2; α:= α/2; γ := γ+1   end;

ρ :=  ρ+(α-⌊α); α := ⌊α;
while   ρ ≥ 2^(k-1) do
begin   ρ := (ρ+αmod2)/2; α:=  α/2;  α := α+1
end;

r:= ⌈ρ; f:=s*(⌊α); e:=γ;
```

A.2 Addition and Subtraction

For these and the following arithmetic operations we will assume that the operands are (e_1,f_1,r_1) and (e_2,f_2,r_2) respectively. The results of the operation will be described in terms of (γ,α,ρ) which then has to be mapped into the representation.

Assuming that $e_1 \ge e_2$ then the result of the addition is:

$$\alpha=f_1+f_2 2^{e_1-e_2},\ \rho=r_1+r_2 2^{e_1-e_2},\ \gamma=e_1.$$

Similarly for subtraction:

$$\alpha=f_1-f_2 2^{e_1-e_2},\ \rho=r_1+r_2 2^{e_1-e_2},\ \gamma=e_1.$$

Notice that α and ρ may be numbers containing a fractional part.

A.3 Division

Assume that the numerator is (e_1,f_1,r_1) and the demoninator is (e_2,f_2,r_2). If $|f_2|\le r_2$ the result will be "undefined", otherwise:

$$\alpha=\text{sign}(f_1 f_2)\ \frac{|f_1 f_2| + r_1 r_2}{f_2^2 - r_2^2},$$

$$\alpha=\frac{|f_1| r_2 + |f_2| r_1}{f_2^2 - r_2^2},\quad \gamma=e_1-e_2$$

In general α and ρ will not be exact representable, as they may contain infinite fractions. If the division algorithms for α and ρ are stopped whenever the quotients of α and ρ contains as many binary digits as can be represented (n-1 or k respectively), then the addition of a one to the least significant bit of ρ will compensate for the fractions of α.

A.4 Multiplication

There are four different cases to consider, depending on the signs of f_1 and f_2, and the relations $|f_1|\ge r_1$ and $|f_2|\ge r_2$. The cases may be considered two by two:

Case 1 & 2 (The origin is not in any of the intervals)

1) $f_1 f_2 > 0$, $|f_1|\ge r_1$, $|f_2|\ge r_2$:

$\alpha=f_1 f_2+r_1 r_2$, $\rho=|f_1| r_2+|f_2| r_1$, $\gamma=e_1+e_2$

2) $f_1 f_2 < 0$, $|f_1|\ge r_1$, $|f_2|\ge r_2$:

$\alpha=f_1 f_2-r_1 r_2$, $\rho=|f_1| r_2+|f_2| r_1$, $\gamma=e_1+e_2$

Which implies that both cases are covered by the expressions:

$\alpha=\text{sign}(f_1 f_2)(|f_1||f_2| + r_1 r_2)$,

$\rho=|f_1| r_2 + |f_2| r_1$, $\gamma=e_1+e_2$

Cases 3 & 4 (The origin is in either or both intervals)

Both cases are covered by the following expressions:

$$\alpha=f_1f_2 - \frac{|d| - |s|}{2}, \quad \rho=r_1r_2 + \frac{|d| + |s|}{2},$$

$$\gamma=e_1+e_2,$$

where $d=f_1r_2-f_2r_1$ and $s=f_1r_2 + f_2r_1$. Since these expressions involve unnecessary computations, they may be split into the following two cases:

3) $f_1r_2 + f_2r_1 \geq 0$:

$\alpha=f_1f_2 + \min(f_1r_2,f_2r_1)$,

$\rho=r_1r_2 + \max(f_1r_2,f_2r_1)$,

$\gamma=e_1 + e_2$

4) $f_1r_2 + f_2r_1 < 0$:

$\alpha=f_1f_2 - \max(f_1r_2,f_2r_1)$,

$\rho=r_1r_2 - \min(f_1r_2,f_2r_1)$,

$\gamma=e_1 + e_2$

Notice that multiplication results in both α and ρ being integer numbers.