A postfix notation for logic circuits

F. G. Duncan,* D. Zissos,† and Maureen Walls†

Reverse Polish (postfix) notation is well known to compiler writers through its relationship to the concepts of the stack and the tree (Randell and Russell, 1969). A development of the ideas of postfix notation is proposed which in principle can be applied to any information structure expressible as a directed graph. In the present paper the developed notation is specialised to the needs of logic circuits. It has already proved itself as a powerful and convenient tool in logic design. Its considerable advantages over the conventional notation of Boolean expressions stem from the immediate one-to-one correspondence between its symbols and the elements of the physical circuit. In addition it has been found most useful for the representation of circuits in computer storage, particularly for programs concerned with automatic circuit design.

(Received July 1973)

1. Introduction

In an earlier paper (Zissos and Duncan, 1972) two of the present authors proposed the introduction of NAND and NOR operators to enable a one-to-one correspondence to be achieved between the symbols of Boolean expressions and equations and the elements of logic circuits. The most serious disadvantage of the notation then proposed has been an excessive need for brackets, resulting in frequent transcription errors, The need for a bracket-free notation was apparent, and it seemed reasonable to begin with a second look at the reverse Polish (postfix) system.

In the usual postfix system, each operator is written after its operands:

$$a + b \text{ appears as } a\ b\ +$$
$$a - b \text{ appears as } a\ b\ -$$
$$b - a \text{ appears as } b\ a\ - \ .$$

The number of operands required by an operator must be fixed and is implicit in the operator itself. The notation does not take account of commutativity except that it respects the actual ordering of operands.

Where an operator in ordinary notation is used with differing numbers of operands (e.g. monadic and dyadic minus), a distinct operator symbol must be admitted into the postfix system for each number of operands ('adicity'). Thus if we use '−' as above for dyadic minus (subtraction) we must have something like 'neg' for monadic minus (negation):

$$\text{e.g. } -a \text{ appears as } a\ \mathbf{neg}\ .$$

Expressions as those given above can be treated as operands in the building up of more complicated expressions. This introduces no ambiguity to be resolved by the use of brackets. Thus

$$(a + b)(c - d) - f/g \text{ appears as } a\ b\ +\ c\ d\ -\ \times\ f\ g\ /\ -$$

while

$$((a + b)(c - d) - f)/g \text{ appears as } a\ b\ +\ c\ d\ -\ \times\ f\ -\ g\ /\ .$$

As was pointed out in Zissos and Duncan (1972), NOR and NAND gates used in logic circuits have differing 'fan-in', and so the corresponding operators were allowed to have variable numbers of operands, that is, variable 'adicity'.

For arithmetic expressions we might do something similar—beginning by writing '−1' and '−2' for monadic and dyadic minus respectively, and, for good measure, writing in the adicity of every operator explicitly.

Thus $(-a + b)(c - d)$ would now appear as

$$a - 1\ b + 2\ c\ d - 2 \times 2\ .$$

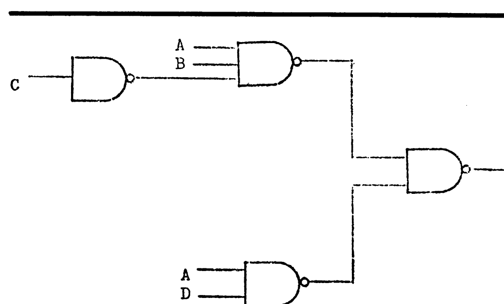


Fig. 1 NAND circuit for ABC + AD

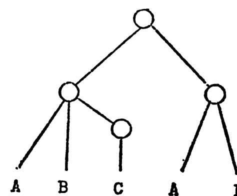


Fig. 2 A tree structure

It should be noted at this stage that $a + b + c$ can be written as

$$a\ b\ c + 3$$

as well as $a\ b\ +2\ c\ +2$ and as $a\ b\ c\ +2\ +2$.

The form $a\ b\ c\ +3$ is much more satisfying than either of these other two forms; it can be read as 'the sum of *a*, *b*, *c*' rather than 'the sum of *a* and the sum of *b*, *c*'. There is, after all, nothing in the nature of addition to suggest it is essentially dyadic.

We now have a basis from which a notation for logic circuits can be developed.

2. NOR/NAND expressions

The circuit in **Fig. 1**, whose output is $AB\bar{C} + AD$, may be expressed in the notation of Zissos and Duncan as

$$\triangle(\triangle(A, B, \triangle C), \triangle(A, D))\ .$$

It may now be expressed as

$$A\ B\ C\triangle 1\triangle 3\ A\ D\triangle 2\triangle 2\ .$$

The *configuration* of the circuit may be even more simply expressed, by dropping the operator symbols, as

$$A\ B\ C\ 1\ 3\ A\ D\ 2\ 2\ .$$

*Department of Computer Science, University of Bristol, School of Mathematics, University Walk, Bristol BS8 1TW.
†Department of Mathematics, Statistics and Computing Science, University of Calgary, Canada.

Downloaded from https://academic.oup.com/comjnl/article/18/1/63/455225 by guest on 19 April 2024

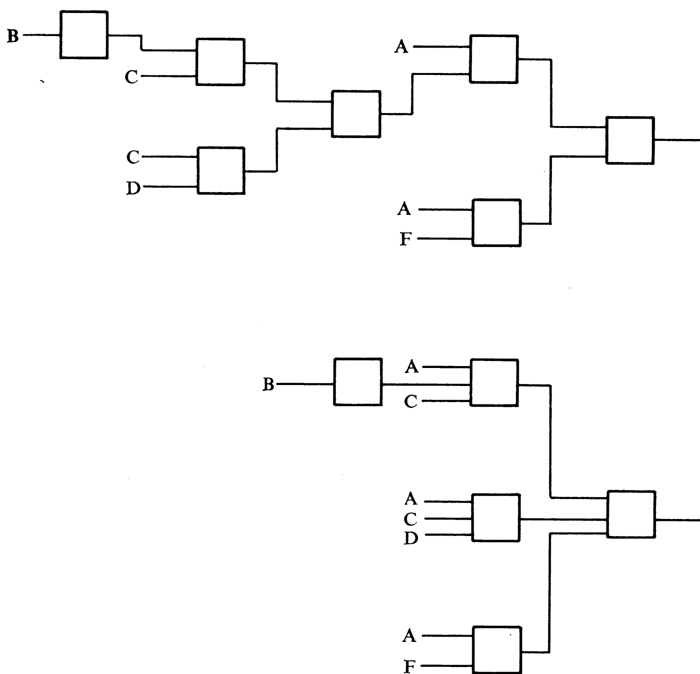


Fig. 3 Two equivalent circuits

(This last form may be regarded as a postfix expression for a tree as in **Fig. 2.**)

To interpret the expression $A\,B\,C\,1\,3\,A\,D\,2\,2$, given that it stands for a logic circuit, we need to know whether it has NAND gates or NOR gates. We can then (notionally) supply the missing operator symbols, and, using the adicity numbers, evaluate the expression from left to right. For NAND the result is $AB\bar{C} + AD$, for NOR it is $(A + B + \bar{C})(A + D)$— these are, of course, dual expressions.

We now consider the manipulation of these postfix expressions as a technique in the design of logic circuits.

Turning now to the manipulation of these postfix expressions, consider the two configurations of **Fig. 3.** Although these two circuits are logically equivalent the second implementation may be preferred insofar as it has fewer gates and shorter signal paths at the expense of higher loading of one of the signal sources and the use of three-input gates.

(If NAND gates are assumed these circuits give

$A(\bar{B}C + CD) + AF$ and $A\bar{B}C + ACD + AF$ respectively;

with NOR gates they give

$$(A + (\bar{B} + C)(C + D))(A + F)$$

and

$$(A + \bar{B} + C)(A + C + D)(A + F).)$$

The problem is to reduce $A\,B\,1\,C\,2\,C\,D\,2\,2\,2\,A\,F\,2\,2$ to $A\,B\,1\,C\,3\,A\,C\,D\,3\,A\,F\,2\,3$ by symbol manipulation only.

We write

$$A\,B\,1\,C\,2\,C\,D\,2\,2\,2\,A\,F\,2\,2 \equiv A\,X\,Y\,2\,2\,A\,F\,2\,2$$

(where $X \equiv B\,1\,C\,2$, $Y \equiv C\,D\,2$)

$$\equiv A\,F\,2\,A\,X\,Y\,2\,2\,2$$

by commutativity.

Now

$$A\,X\,Y\,2\,2 \equiv A\,X\,1\,2\,A\,Y\,1\,2\,2\,1 \qquad (1)$$

(This is equivalent to the identity

$\bar{A} + XY \equiv (\bar{A} + X)(\bar{A} + Y)$ NAND

or $\bar{A}(X + Y) \equiv \bar{A}X + \bar{A}Y$ NOR)

Therefore,

$$A\,B\,1\,C\,2\,C\,D\,2\,2\,2\,A\,F\,2\,2 \equiv A\,F\,2\,A\,X\,1\,2\,A\,Y\,1\,2\,2\,1\,2$$
$$\equiv A\,F\,2\,A\,B\,1\,C\,2\,1\,2\,A\,C\,D\,2\,1\,2\,2\,1\,2\,.$$

Also,

$$P\,Q\,R\,2\,1\,2 \equiv P\,Q\,R\,3 \qquad (2)$$

(This is equivalent to the identity

$\bar{P} + (\overline{\overline{Q + R}}) \equiv \bar{P} + \bar{Q} + \bar{R}$ NAND

or $\bar{P}.(\overline{\overline{Q.R}}) \equiv \bar{P}.\bar{Q}.\bar{R}$ NOR)

Therefore,

$$A\,B\,1\,C\,2\,C\,D\,2\,2\,2\,A\,F\,2\,2 \equiv A\,F\,2\,A\,B\,1\,C\,3\,A\,C\,D\,3\,3$$
$$\equiv A\,B\,1\,C\,3\,A\,C\,D\,3\,A\,F\,2\,3$$

by commutativity.

From this example it is clear that, as in the manipulation of conventional expressions, it will be useful to memorise a number of elementary results for use in proceeding from one step to another. The two such results applied here are analogous to statements of the distributive (1) and associative (2) laws in arithmetic expressions.

Boolean algebra has two other special results of practical importance—the 'absorption rule' and the 'optional product rule'. In conventional terms these are:

Absorption:

$$A + AB \equiv A$$

or

$$A(A + B) \equiv A$$

Optional product (or factor):

$$AB + \bar{A}C \equiv AB + \bar{A}C + BC$$

or

$$(A + B)(\bar{A} + C) \equiv (A + B)(\bar{A} + C)(B + C)\;.$$

In our notation:

Absorption:

$$A\,1\,A\,B\,2\,2 \equiv A \qquad (3)$$

Optional product (or factor):

$$A\,B\,2\,A\,1\,C\,2\,2 \equiv A\,B\,2\,A\,1\,C\,2\,B\,C\,2\,3\;. \qquad (4)$$

The next example illustrates the use of (3).

$A\,B\,2\,C\,D\,2\,A\,B\,C\,3\,3 \equiv A\,B\,2\,C\,D\,2\,C\,A\,B\,3\,3$ by commutativity

$\equiv A\,B\,2\,C\,D\,2\,C\,A\,B\,2\,1\,2\,3$ by (2)

$\equiv X\,C\,D\,2\,C\,X\,1\,2\,3$ where $X = A\,B\,2$

$\equiv C\,D\,2\,X\,X\,1\,C\,2\,3$ by commutativity

$\equiv C\,D\,2\,X\,X\,1\,C\,2\,2\,1\,2$ by (2)

$\equiv C\,D\,2\,X\,1\,1\,X\,1\,C\,2\,2\,1\,2$ since $X\,1\,1 = X$

$\equiv C\,D\,2\,X\,1\,1\,2$ by (3)

$\equiv C\,D\,2\,X\,2$

$\equiv C\,D\,2\,A\,B\,2\,2$

$\equiv A\,B\,2\,C\,D\,2\,2$ by commutativity.

Thus the equivalence of the configurations in **Fig. 4** is shown. As an example of the application of (4) above we have:

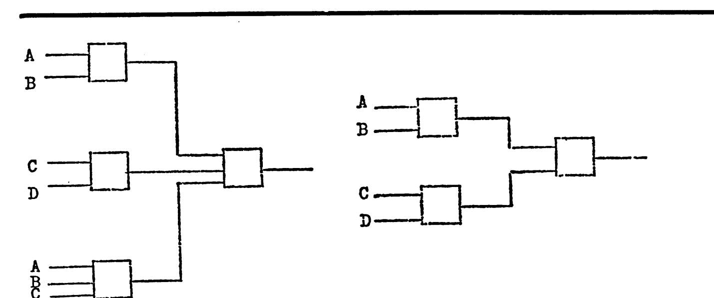


Fig. 4 Example of absorption rule

Downloaded from https://academic.oup.com/comjnl/article/18/1/63/455225 by guest on 19 April 2024

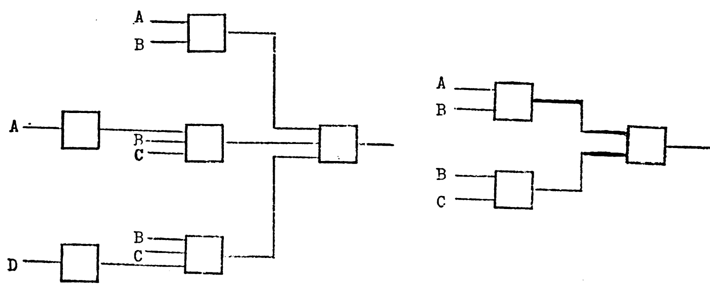


Fig. 5 Example of optional product rule

$A\,B\,2\,A\,1\,B\,C\,3\,B\,C\,D\,1\,3\,3$

$\equiv A\,B\,2\,A\,1\,B\,C\,2\,1\,2\,B\,C\,D\,1\,3\,3$ by (2)

$\equiv A\,B\,2\,A\,1\,B\,C\,2\,1\,2\,D\,1\,B\,C\,2\,1\,2\,3$ by commutativity and (2)

$\equiv D\,1\,B\,C\,2\,1\,2\,A\,B\,2\,A\,1\,B\,C\,2\,1\,2\,2\,1\,2$ by commutativity and (2)

$\equiv D\,1\,B\,C\,2\,1\,2\,A\,B\,2\,A\,1\,B\,C\,2\,1\,2\,B\,B\,C\,2\,1\,2\,3\,1\,2$ by (4)

$\equiv D\,1\,B\,C\,2\,1\,2\,A\,B\,2\,A\,1\,B\,C\,2\,1\,2\,B\,C\,2\,3\,1\,2$ since $B\,B\,2 = B\,1$

$\equiv D\,1\,X\,1\,2\,A\,B\,2\,A\,1\,X\,1\,2\,X\,3\,1\,2$ where $X = B\,C\,2$

$\equiv D\,1\,X\,1\,2\,A\,B\,2\,A\,1\,X\,1\,2\,X\,2\,1\,2\,1\,2$ by (2).

$\equiv D\,1\,X\,1\,2\,A\,B\,2\,X\,1\,1\,2\,1\,2$ by (3)

$\equiv D\,1\,X\,1\,2\,A\,B\,2\,X\,2\,1\,2$

$\equiv A\,B\,2\,X\,X\,1\,D\,1\,2\,2\,1\,2$ by commutativity

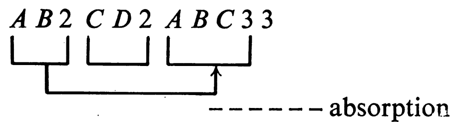
$\equiv A\,B\,2\,X\,1\,1\,2$ by (3)

$\equiv A\,B\,2\,X\,2$

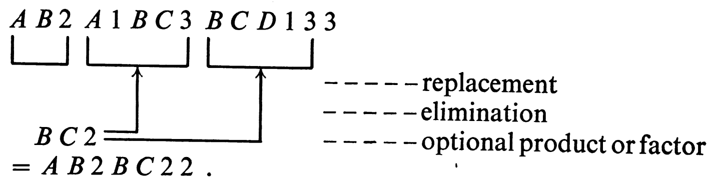
$\equiv A\,B\,2\,B\,C\,2\,2$.

This shows the equivalence of the configurations in **Fig. 5.**

In the working given above every step is shown in detail. However many short cuts become possible with experience, and the overall method described in Zissos and Duncan (1973a) can be followed exactly. The first of these examples is in practice worked simply as:



The second becomes:



$= A\,B\,2\,B\,C\,2\,2$.

The optional product of $A\,B\,2$ and $A\,1\,B\,C\,3$ is $B\,C\,2$, and this can replace the parent term $A\,1\,B\,C\,3$ (by absorption) and eliminate the non-parent term $B\,C\,D\,1\,3$ (also by absorption).

The final step necessary for the minimisation of a Boolean expression, as described in Zissos and Duncan (1973a), is also adaptable to this notation. The step is typified by the irredundant expression

$$AB + \bar{A}\bar{B} + \bar{A}\bar{C} + AC \tag{5}$$

which has a minimal form

$$AC + B\bar{C} + \bar{A}\bar{B} \text{ .} \tag{6}$$

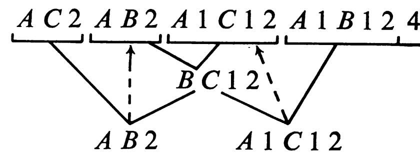
The expression (5) translates directly into

$$A\,B\,2\,A\,1\,B\,1\,2\,A\,1\,C\,1\,2\,A\,C\,2\,4 \tag{7}$$

which by commutativity is

$$A\,C\,2\,A\,B\,2\,A\,1\,C\,1\,2\,A\,1\,B\,1\,2\,4 \text{ .}$$

Relevant optional products and their use with the absorption rule can be seen as follows:



As long as $A\,C\,2$ and $A\,1\,B\,1\,2$ are present, $B\,C\,1\,2$ can replace both its parent terms $A\,B\,2$ and $A\,1\,C\,1\,2$.

Thus we have

$A\,C\,2\,A\,B\,2\,A\,1\,C\,1\,2\,A\,1\,B\,1\,2\,4 \equiv A\,C\,2\,B\,C\,1\,2\,A\,1\,B\,1\,2\,3$ (**Fig. 6**), the latter expression corresponding to the minimal form (6).

The expression (7) has nine operators, but the circuit in Fig. 6 (top) has only eight gates. The reason is that $A\,1$ occurs twice in the expression, and the inverter for A has been allowed a fan-out of two.

For a number of reasons including the desire to maintain one-to-one correspondence between the symbols of an expression and the elements of the circuit, we introduce a 'naming operator'. It is denoted by '=' and is regarded essentially as

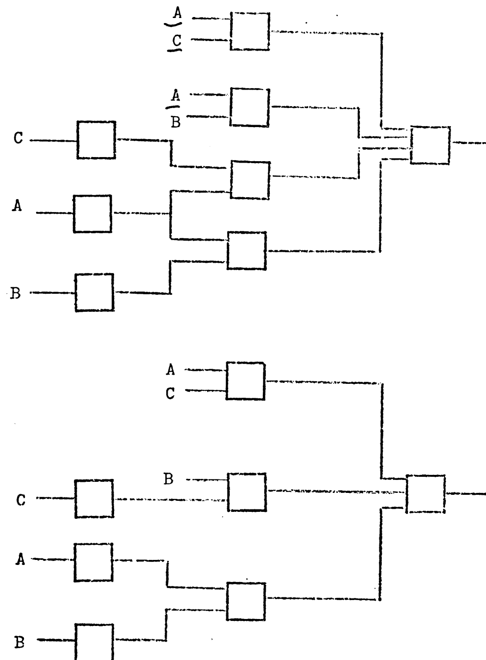


Fig. 6 Example of minimisation

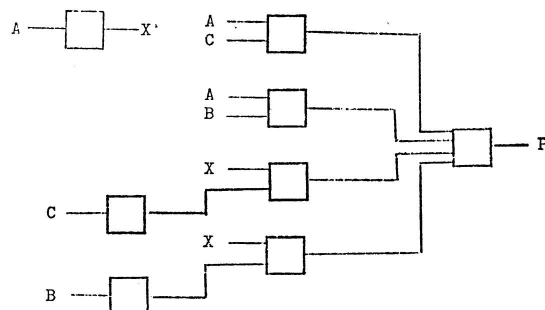


Fig. 7 Use of naming operator

Downloaded from https://academic.oup.com/comjnl/article/18/1/63/455225 by guest on 19 April 2024

dyadic, associating a name (its second operand) with a value (its first operand). (Later, in Section 5, we consider a multiple naming operator.)

Expression (7) can now be named and one-to-one correspondence restored as follows.

$$A\,1\,X = A\,C\,2\,A\,B\,2\,X\,C\,1\,2\,X\,B\,1\,2\,4\,P =$$

This should be compared with **Fig. 7.**

As a final example in this section, the reader may like to verify that the expression given below, when realised with NAND gates, gives a simple excess-3 encoder (simple in the sense that it does not check against invalid inputs). The inputs are A, B, C, D; the outputs are P, Q, R, S; inputs 0, 1, 1, 1 give outputs 1, 0, 1, 0 (7 + 3 = 10), etc.

$$A\,1\,B\,C\,2\,B\,D\,2\,3\,P = D\,1\,S = C\,S\,2\,X = C\,X\,2\,X\,S\,2\,2\,R = C\,1\,S\,2\,Y = B\,Y\,2\,Z = B\,Z\,2\,Z\,Y\,2\,2\,Q = .$$

3. Combinational circuits—special considerations

A number of useful pieces of information about a NAND or NOR combinational circuit may be read from the corresponding postfix expression.

Loading of inputs

The number of occurrences of the name of an input signal is the number of unit loads (in the usual sense) imposed by the circuit on the source of that signal. Thus the excess-3 encoder of the previous section imposes loads of 1, 4, 4, 2 units respectively on the sources of A, B, C, D.

Weakening of outputs

The effective fan-out from an output gate is reduced by the number of unit loads required internally to the circuit. In the encoder, P, Q, R can bear the full fan-out, but the fan-out from S is reduced by 3 since that is the number of occurrences of S (other than its naming occurrence) in the expression.

The fan-in of each gate is explicit in the expression.

Signal delays

Assuming a nominal switching time, these can be determined by counting the number of levels through which a signal has to pass from input to output. This is most conveniently done as part of the investigation of hazards, below.

Hazards

Circuits can be designed hazard-free (Zissos, 1972), but it may be required to investigate the possibility of hazards in a given circuit. Consider, for example, the expression

$$A\,C\,2\,B\,C\,1\,2\,A\,1\,B\,1\,2\,3$$

corresponding to Fig. 6 (bottom).

We first trace signal paths by writing down the given expression and placing underneath it expressions derived by suppressing all inputs except one.

It is in practice useful, both as an aid to clarity and to avoid a possible source of error, to indicate the association of operands and operators by means of horizontal ties as shown.

Full expression	$A\,C\,2$	$B\,C\,1\,2$	$A\,1\,B\,1\,2\,3$
Suppress all inputs except A	$A\;\;1$		$A\,1\;\;\;\;1\,2$
Suppress all inputs except B		$B\;\;\;1$	$B\,1\,1\,2$
Suppress all inputs except C	$C\,1$	$C\,1\,1$	2

These residual expressions show that each signal has two paths to the output of lengths 2 and 3 gates respectively. If the other inputs (if any) to all the gates on these paths are held at 1, a change to the input concerned may cause a spike at the output.

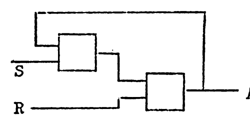


Fig. 8 A simple sequential circuit

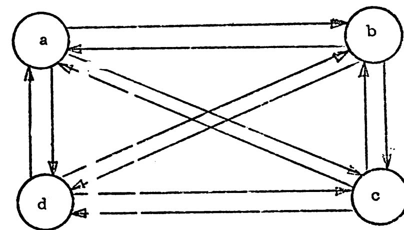


Fig. 9 A directed graph

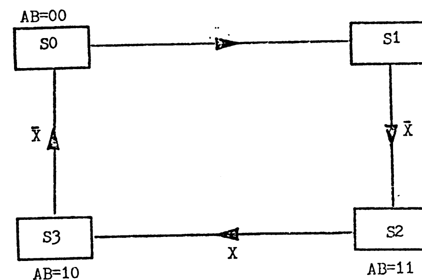


Fig. 10 T-type flip-flop

Consider first the input A. Assume the gates are numbered serially from left to right in the full expression. A goes through gate number 1; the other input to this gate is C. Gate number 4 is merely an inverter for A. Gate number 6 has an input of $B\,1$. Gate number 7 has one input which is not on the paths of A, namely $B\,C\,1\,2$.

Therefore, necessary conditions for a spike at the output are to have C at 1, $B\,1$ at 1, and $B\,C\,1\,2$ at 1. These conditions are all satisfied if $C = 1$, $B = 0$.

The residual expression $A\,1\,A\,1\,1\,2$ shows that a change in A reaches the final gate after one gate delay time on one path (gate number 1) and after two gate delay times on the other path (gate numbers 4 and 6). Consequently, if A is switched from 1 to 0 while $B = 0$ and $C = 1$, there will be a spike at the output; the output, which should remain constant at 1 will temporarily assume the value 0 between one and two gate delay times after the change in A.

Hazards due to changes in B and C can be found by a similar process.

4. Sequential circuits—representation of feedback

No further developments in the notation are needed to represent the configurations encountered in NAND or NOR sequential circuits.

The simplest illustration is a set-reset flip-flop made from two NOR gates (**Fig. 8**).

This configuration can be expressed as

$$A\,S\,2\,R\,2\,A = .$$

The new feature of this expression is that we have in effect a recursive definition for A. This embodies the 'feedback' or 'memory' aspect of the sequential circuit.

(The notation is in fact adequate to describe any directed graph, not only trees. For example, the complete directed graph of four vertices in **Fig. 9** is

$$B\,C\,D\,3\,A = A\,C\,D\,3\,B = A\,B\,D\,3\,C = A\,B\,C\,3\,D =$$

where A means 'an arc (edge) from vertex a' etc.

Downloaded from https://academic.oup.com/comjnl/article/18/1/63/... by guest on ... April 2024

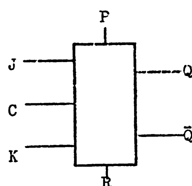


Fig. 11 **JK flip-flop**

It is hoped to describe the application of the notation to basic graph theory in a later paper.)

A rather more interesting example of a sequential circuit is the T-type flip-flop whose state diagram is shown in **Fig. 10.** A direct realisation of this with NOR gates is

$$X\,1\,x = A\,b\,X\,2\,2\,a = B\,A\,x\,2\,2\,b = a\,B\,X\,2\,2\,A = b\,a\,x\,2\,2\,B = \quad (8)$$

However, by a systematic application of a design algorithm given in Zissos (1972) this 9-gate circuit can be reduced to the following 6-gate circuit.

$$X\,p\,q\,3\,r = B\,X\,2\,p = r\,s\,2\,q = A\,r\,2\,s = s\,p\,2\,A = p\,q\,2\,B = \quad (9)$$

The manipulative techniques are similar to those described in Section 2. The basic knowledge required is the result

$$P\,1\,Q\,1\,2\,R\,2\,1 \equiv P\,R\,2\,Q\,R\,2\,2 \quad (10)$$

and the fact that in certain circumstances a is the inverse of A and b is the inverse of B (notation of (8)).

The primitive sequential equations, which follow from the state diagram, are:

$$A\,B\,1\,X\,2\,2\,B\,X\,2\,2\,A = B\,A\,X\,1\,2\,2\,A\,1\,X\,1\,2\,2\,B = \quad (11)$$

Substitution of a for $A\,1$, and b for $B\,1$, and the introduction of x for $X\,1$, leads immediately to (8).

The application of (10) to the second half of (11) transforms (11) into:

$$A\,B\,1\,X\,2\,2\,B\,X\,2\,2\,A = B\,X\,2\,A\,1\,B\,1\,X\,2\,2\,2\,B = .$$

Noting repeated sub-expressions, we write:

$$B\,1\,X\,2\,r = B\,X\,2\,p = A\,r\,2\,p\,2\,A = p\,A\,1\,r\,2\,2\,B = \quad (12)$$

We try to eliminate the indicated inverters for A, B.

$$B\,1\,X\,2 \equiv p\,A\,1\,r\,2\,2\,1\,X\,2 \equiv X\,p\,A\,1\,r\,2\,2\,1\,2 \equiv X\,p\,A\,1\,r\,2\,3 \equiv X\,p\,q\,3 \quad \text{where } q \equiv A\,1\,r\,2 \quad (13)$$

$$A\,1\,r\,2 \equiv A\,r\,2\,r\,2 \equiv r\,s\,2 \quad \text{where } s \equiv A\,r\,2\,. \quad (14)$$

Substituting (13), (14) in (12) we have:

$$X\,p\,q\,3\,r = r\,s\,2\,q = B\,X\,2\,p = A\,r\,2\,s = s\,p\,2\,A = p\,q\,2\,B =$$

which is equivalent to (9).

The systematic application of the steps involved, and their engineering justification, is given in Zissos (1972).

5. Abbreviations, and notation for repeated sub-circuits

It is convenient to introduce the abbreviation $\bar{A}$ for $A\,1$, or rather the convention that the name $\bar{A}$ stands for a quantity which is always necessarily the complement of the quantity denoted by the name A. This enables one to deal with those situations where two outputs from a circuit (e.g. the two outputs of a JK flip-flop) are complementary. The use of $\bar{A}$ as an abbreviation is a convenience fraught with a little danger in cases where hazards are involved; otherwise it is perfectly safe.

A further convenience is a multiple naming operator. The operator $=$ associates a name and a value; the operator $=2$ associates a pair of names with a pair of values. Thus

$$X\,Y\,2\,Y\,Z\,2\,A\,B = 2$$

is equivalent to

$$X\,Y\,2\,A = Y\,Z\,2\,B = .$$

In general if $=$ is followed by an integer n, it is preceded by $2n$ operands; the first n are values which are given the names written as the second n, matched one to one.

The encoder in Section 2 above is a circuit with four inputs and four outputs which can now be represented by an operator with four operands;

$$A\,B\,C\,D\,\textbf{encoder}\,4\ .$$

This expression stands for four values (the outputs) which can be named by a multiple naming operator:

$$A\,B\,C\,D\,\textbf{encoder}\,4\,P\,Q\,R\,S = 4\ .$$

A JK flip-flop with preset and reset inputs (Zissos and Duncan, 1973*b*) can be represented as

$$J\,K\,C\,R\,P\,\textbf{JK}\,5\,Q\,\bar{Q} = 2 \ (cf.\ \textbf{Fig. 11})\ .$$

This expression can be successively abbreviated as follows:

$J\,K\,C\,R\,\textbf{JK}\,4\,Q\,\bar{Q} = 2$	(no preset)
$J\,K\,C\,\textbf{JK}\,3\,Q\,\bar{Q} = 2$	(no preset or reset; or preset/reset conditions understood)
$J\,K\,\textbf{JK}\,2\,Q\,\bar{Q} = 2$	(as before, and clock conditions understood, as in synchronous circuits).

Some of the possible operators, like **JK,** are sufficiently well known for them to be regarded as 'standard functions, available without explicit declaration'. Others, like **encoder,** need to be defined within a particular circuit description.

The following is a definition for the operator **encoder** adapted from the expression given in section 2 for the excess-3 encoder.

$$A\,B\,C\,D\,\textbf{in}\,4\,A\,1\,B\,C\,2\,B\,D\,2\,3\,P = D\,1\,S = C\,S\,2\,X = C\,X\,2\,X\,S\,2\,2\,R = C\,1\,S\,2\,Y = B\,Y\,2\,Z = B\,Z\,2\,Z\,Y\,2\,2\,Q = P\,Q\,R\,S\,\textbf{out}\,4\ \textbf{encoder op.}$$

The operator-defining operator **op** has three operands which are respectively an input list, an output list, and an operator symbol. The input list is defined here by '$A\,B\,C\,D$ **in** 4'; **in** is an operator of variable adicity whose operands are names. The output list is defined by the operator **out**, also of variable adicity, whose operands are values. In the example these values arise from the expression defining P, Q, R, S. The operator symbol is **encoder,** which, in the present state of the notation, must be 'unique'.

It will be observed that in this example we have introduced 'working variables' X, Y, Z which are required only inside the expression for P, Q, R, S. There seems no reason not to regard these as purely local to the operator definition; the names X, Y, Z can thus be used 'outside' in the normal way.

In the case of the **JK** operator, the user of the notation needs to know the ordering of the output values, but not necessarily how they are derived internally from the inputs. Thus the following declaration of **JK** is adequate, and it is consistent.

$$J\,K\,C\,R\,P\,\textbf{in}\,5\,Q\,\bar{Q}\,\textbf{out}\,2\,\textbf{JK}\,\textbf{op}\ .$$

Two points arise:

(*a*) Although a number of operators have been introduced to be represented in their full form (operator symbol followed by adicity), it is still possible to omit the NOR or NAND symbol (leaving it 'understood') if this is convenient.

(*b*) In all the examples names of one letter and adicities of one digit have been used. If longer names and integers are required, it will be necessary to use a separator—preferably a comma—between the elements of an expression. It is convenient to reserve the names I and O for the Boolean values 1 and 0 respectively.

6. Interpretation of postfix expressions

Expressions in the proposed notation are evaluated according to the normal rules for postfix expressions:

Downloaded from https://academic.oup.com/comjnl/article/18/1/63/455225 by guest on 19 April 2024

(*a*) Each name is stacked as it is encountered during the left to right scan.
(*b*) Each operator is applied to the top entries in the stack, deleting these entries and leaving its result, if any, on the top of the stack.

It should be noted that a defined operator symbol must be treated as an operand of **op** on its defining occurrence and as an operator elsewhere.

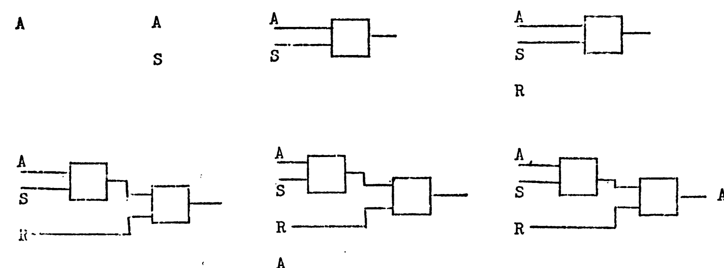
The reader who is familiar with logic circuit diagrams but not with postfix notations may find the following rules helpful for drawing a diagram from a given expression.

(*a*) Take the elements of the expression in order from left to right.
(*b*) For each name encountered, add it to a column on the left of the paper.
(*c*) For a gate operator, draw a gate symbol to the right of the signals to be gated. These are the signals at the 'bottom' of the diagram, and their number is the adicity of the operator.
(*d*) For a naming operator '=', if the adicity is 1 or omitted, transfer the last name from the left hand column to the signal immediately above it. If the adicity is more than 1, the group of the given number of names must be transferred from the bottom of the left hand column to the same number of signals immediately above.

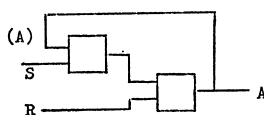
Example:

$$A\,S\,2\,R\,2\,A\,=$$

The seven stages corresponding to the seven symbols are:



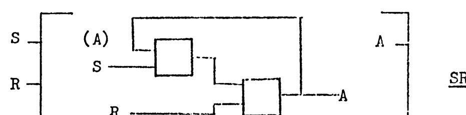
Finally any links representing feedback should be drawn in if required.



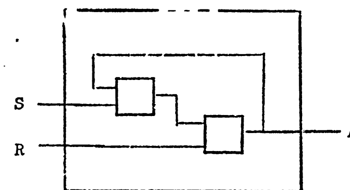
(*e*) For the **in** operator, note that we are drawing a sub-circuit which is to be used at several points later. Draw a large left-hand square bracket next to the affected names of the left hand column.
(*f*) Treat the **out** operator like **in**, but with a right-hand square bracket.
(*g*) A new operator being defined is followed by **op**; it should label the circuit 'enclosed' by the square brackets.

Example:
Set-reset flip-flop.

$S\,R$ **in** $2\,A\,S\,2\,R\,2\,A = A$ **out** 1 **SR op** .



Now tidy up the diagram according to taste:

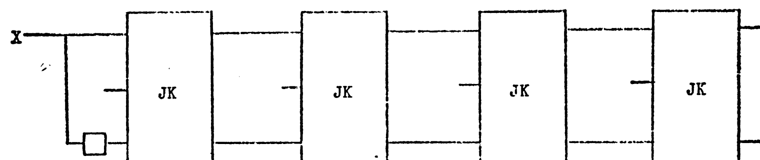


(*h*) For any other operator, proceed as in (*e*) above, using either a standard symbol (for a 'built in' function) or a skeleton drawing based on that produced by (*e*)–(*g*) above.

Examples:

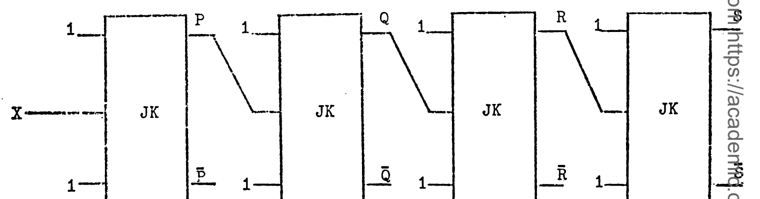
Four-bit synchronous **JK** shift register.

$X\,X\,1$ **JK** 2 **JK** 2 **JK** 2 **JK** 2 .



Four-bit asynchronous binary counter.

$I\,I\,X$ **JK** $3\,P\,\overline{P} = 2\,I\,I\,P$ **JK** $3\,Q\,\overline{Q} = 2\,I\,I\,Q$ **JK** $3\,R\,\overline{R} =$
$2\,I\,I\,R$ **JK** $3\,S\,\overline{S} = 2$



In this example I stands for logic 1. The device counts pulses at S. The counter reading is

$$P.2^0 + Q.2^1 + R.2^2 + S.2^3\ .$$

$\overline{P}, \overline{Q}, \overline{R}, \overline{S}$ are also available.

7. Further considerations

The problem of mechanical translation between conventional infix notation and the proposed notation should not present difficulties, and programs for this are being written. Programs for manipulating postfix expressions as part of the logic design process are also under consideration. One of us (MW) has made a detailed study of the manipulations which are required.

Some further developments of the notation itself are clearly desirable. Each of the last two examples of Section 6 involves the repetition of a string of symbols for each of the bits in the shift register or counter. To avoid this repetition in larger, more realistic, cases, a form of subscripting might be introduced.

A possibility is

$X\,X\,1$ [**JK** $2]^n$ for a synchronous n-bit shift register

and $[I\,I\,X_i$ **JK** $3\ X_{i+1}\ \overline{X}_{i+1}]_{i=0}^{i=n-1}$ for an asynchronous (ripple-through) n-bit counter, where X_0 is the input (pulses to be counted) and $X_i\ (i = 1(1)n)$ represents the coefficient of 2^{i-1}.

Although this notation is no longer bracket-free it seems to offer some advantages over a pure postfix scheme.

Conclusion

It is already clear that after only a short period of practice the proposed notation becomes a very useful and powerful means of handling the algebraic aspects of the logic design process. Indeed, the notation is capable of replacing the existing means of expression. This is true both in written work and in computer programs; postfix expressions are easily manipulated as strings and the established translation methods, including Djikstra's

Downloaded from https://academic.oup.com/comjnl/article/18/1/62/445225 by guest on 19 April 2024

algorithm, are readily adapted for this purpose. At all stages in the work, the fact that the symbols stand in immediate one-to-one correspondence with actual circuit elements removes one of the greatest inconveniences of the conventional notation. This applies, moreover, to components at higher levels than that of gates, and up to system level if required. In this respect it will be desirable to develop 'standard declarations' for such components as common integrated circuits, standard interfaces (as regards their logic aspects), and so on. Using a one-line declaration for a 4-bit full adder (e.g. TTL 7483), a circuit involving 15 such devices has been notated in eight lines; the postfix expression clearly showed up the point where a subsequent modification should be made.

Current work is concerned with the systematisation of the manipulation of postfix expressions, the construction of computer programs to assist in the logic design process, and the development of 'operator declarations' for higher-level components.

Acknowledgements

One of us (FGD) is particularly indebted to Dr. J. Mülbacher of the Hochschule für Sozial und Wirtschaftswissenschaften, Linz, Austria for much useful discussion on general directed graphs.

The work has been supported in part by a research grant of the National Research Council of Canada.

References

RANDELL, B., and RUSSELL, L. J. (1969). *ALGOL 60 Implementation*, Academic Press.

ZISSOS, D. (1972) *Logic Design Algorithms*, Oxford University Press.

ZISSOS, D., and DUNCAN, F. G. (1972). NOR and NAND Operators in Boolean algebra applied to switching circuit design, *The Computer Journal*, Vol. 14, No. 4, pp. 413-417.

ZISSOS, D., and DUNCAN, F. G. (1973*a*). Boolean minimisation, *The Computer Journal*, Vol. 16, No. 2, pp. 174-179.

ZISSOS, D., and DUNCAN, F. G. (1973*b*). *Digital Interface Design*, Oxford University Press.

Book review

Digital Computing and Numerical Methods, by B. Carnahan and J. O. Wilkes, 1973; 477 pages. (*John Wiley and Sons*, £7·80.)

The authors of this large book are both Professors of Chemical Engineering at the University of Michigan. They suggest in their preface that the book is '. . . suitable for the early years at university and the orientation is toward engineering and applied mathematics'.

The first two chapters give a conventional introduction to digital computers, flow diagrams and algorithms, with the IBM 360/370 being used to illustrate internal number and character representations. The full title of the book also includes the phrase 'with FORTRAN IV, WATFOR and WATFIV programming'. This is covered in the 146 pages of Chapter 3, with an apology for the existence of many dialects of FORTRAN IV but the assertion that 'the variations from one dialect of FORTRAN to another tend to be rather small'. The authors therefore describe IBM 360/370 Level G or Level H FORTRAN IV as FORTRAN IV and, since the University of Michigan also uses WATFOR and WATFIV, the further features of these variants are also discussed. Grey stripes in the margins are used to indicate the variations between FORTRAN IV, WATFOR and WATFIV but there are no indications of the 'rather small' variations of the FORTRAN described from standard FORTRAN IV. Many a FORTRAN IV compiler would fail to appreciate

```
      REAL FUNCTION PROF*8 (DELTA, *, MAX, *, N)
      IMPLICIT REAL*8 (A – M)
      READ (5, 100, ERR=99)X
  100 FORMAT (G10.3)
      $=N/X
      Y=DELTA=ARSIN (MAX*$)
      PRINT 200, Y
  200 FORMAT (' THE RESULT IS', F12.3)
      RETURN 2
   99 CALL EXIT
      END
```

but the text does not explicitly warn the reader of the peculiarities of such statements. The authors do suggest that they would expect an instructor to supplement the text with additional short examples and that for self-study *A FORTRAN IV Primer* by E. I. Organick would be a useful auxiliary text—though the only statements in the above subprogram which are mentioned by Organick are RETURN and END.

Although the material is well presented and clearly explained, the reviewer would not recommend this introduction to FORTRAN to anyone who was not going to use an IBM 360/370; who else would appreciate the 16 pages of appendices giving the diagnostic error messages from the three compilers?

Chapter 4 gives a 50 page introduction to operating systems with a clear exposition of the development of a modern operating system through the stages of batch processing, multiprogramming, time-sharing and multiprocessing. Excellent detailed diagrams supplement the text which goes as far as considering strategies for implementing paging. (One is pleased to see that the introduction of virtual memory is attributed to the Atlas computer!) The chapter concludes with a detailed account of an on-line session on the MTS terminal system.

The remaining 185 pages cover numerical methods in five chapters headed Solution of single equations, Numerical approximation, Numerical integration, Solution of simultaneous equations, Solution of ordinary differential equations and the Introduction to optimization methods. The approach in these chapters is generally to introduce the problem and develop a simple method of solution, e.g. Euler's method for ordinary differential equations. The weaknesses of the method are then discussed before proceeding to more satisfactory methods, e.g. Runge-Kutta or predictor-corrector, for that type of problem. The treatment is to derive some results and present others without being too rigorous. Indeed the authors recommend that the reader requiring greater depth should consult *Applied Numerical Methods* of which they are joint authors (but see review in *The Computer Bulletin*, Vol. 15, No. 5, p. 186). Each chapter contains one or two typical real problems for which an analysis, flowchart, documented program, results and discussion are given. As training in how to tackle a problem this is good, but the methods are not the most efficient and the programs are therefore only useful for illustration. One notes the comment on page 291, in relation to Horner's rule, that the user should attempt to write efficient programs which could save 20% on total execution time. In the chapter on simultaneous equations, however, the Gauss-Jordan method is presented without mention of the 30% faster Gaussian elimination, nor mention of the existence of indirect methods.

The opinion of the reviewer is thus that the book is limited in its usefulness to students with access to an IBM 360/370 being taught Fortran by someone who would steer them through the idiosyncracies of the dialect of Fortran presented in the book and who would also be aware of the shortcomings of the numerical methods. Each chapter, except the one on operating systems, contains a large number of problems and those for the numerical methods are well-chosen problems in science and engineering fields, generally with suggested test data.

E. W. HADDON (Norwich)

Downloaded from https://academic.oup.com/comjnl/article/18/1/63/455225 by guest on 19 April 2024