

ALGOL 60

Introduced

Formal description

BNF Language Description

Nested structure

Block Structured

Lexical scoping

Multiple types of procedure arguments

"call by name"

"call by value"

Orthogonal design

All variables declared as to type

"own" variables

Recursive procedures

Procedures could be declared in procedures with downward scope

Also defined a publication language so that programs would look nicer in print

But

No IO specification

Popular in Europe - not so much in the US

Many compilers built

Appealed to the researcher but not the business programmer

Not supported by IBM

BNF

The language is described by metalinguistic formulas like

```
<ab> ::= ( | | <ab> ( | <ab> <d>
```

```
<d> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

The idea is that the syntactic entity on the left of the ::= can be replaced by any of the alternatives on the right separated by the | symbol

If the result has more syntactic entities enclosed in < > then further substitutions can be done

When no further substitutions can be done then the result is an ALGOL 60 program

The above example can produce

```
(((1(37(
(12345(
[86
```

Interesting features of ALGOL 60

Used many special symbols not found on the I/O devices of the day (or today)

ALGOL 60 had reserved words that were considered to be a token and not made up of its individual characters

Generally represented as bold underlining

begin end true false

The = sign was reserved for logical comparisons (like == in C++ or Java) and not for assignment

The := symbol was used for assignment

The language had a goto (as did most languages of this time period)

Array subscripts could be declared with lower and upper bounds defined at run time

The semicolon separated statements - it did not terminate statements

Comments looked like statements beginning with the keyword comment and ending with a semicolon

Some expressions had types that could change at runtime

An integer to an integer power was an integer for positive powers and real for negative powers

own variables were like static variables in C++ - there was only one instance of such a variable even though declared in a recursive procedure

procedures without arguments were called by naming them - no following () in the declaration or the call

Designational expressions were expressions whose value was a suitable target of a goto

A switch was an array of designational expressions

Procedures

Types of arguments did not have to be specified unless they were passed by value

There were two ways to pass parameters to procedures (there was nothing like the FORTRAN "call by reference")

Call by value

The local variable in the procedure was initialized by the calling argument

Modification of this local variable would not affect anything external to the procedure

Call by name

The actual argument effectively replaced every occurrence of the variable in the body of the procedure before the body was executed

This meant that the code to evaluate the actual argument was executed every time the argument was referenced

Example

```
begin procedure assign(a, i);
    integer i;
    begin i := i + 1;
        a[i] := 6.5;
    end assign;

    integer j;
    array x[3:12];
    j := 5;
    assign(x, j);
end
```

The call assign(x, j) would increment j to 6 and then assign 6.5 to x[6]

The last assignment to a variable with the same name as that of the procedure was used as the return value of the procedure

Labels could be arguments to a procedure

The for Statement

The for loop allowed concatenating several looping constructs (separated by commas) which were executed in sequence

When one looping construct finished the next one was executed

```
for j := 1, 5 step 3 until 11, j+1 while j < 50 do
xxx;
```

This statement would execute the controlled statement xxx with

```
j = 1
j = 5; j = 8; j = 11;
j = 12; j = 13; ... j = 49;
```


Burroughs 5500 Computer

A "zero address" machine

Operands were on a stack and results returned to the stack

Available with two processors

Designed to facilitate running of ALGOL 60 programs

Improved model was Burroughs 6500/7500

Segment paging

Architecture

48-bit words (plus parity bit)

Maximum memory of 32,768 words

Asynchronous io

6-bit characters (8 per word)

Interrupt system

Processor independent interrupt types

Time interval

Processor B busy

Printer finished

I/O channel busy

I/O channel finished

Keyboard request

Disk file check operation finished

Processor dependent interrupt types

Memory parity

Invalid address

Communication operator

Flag bit

Continuity bit

Invalid index

Exponent underflow

Exponent overflow

Integer overflow

Divide by zero

Program release

Stack overflow

Presence bit

Data Representation

Characters represented in 6-bits

Numbers

Floating-point

Floating-point from 8^{+63} to 8^{-63}

Floating-point was base 8 and normalized by shifting octal digits

The "octal" point of the mantissa was considered to be to the right

Integers

Integers were represented with an exponent of zero

If a floating-point number could be considered an integer it was normalized with an exponent of zero

Logical (boolean)

zero or one

Instructions

The instruction stream of the B 5500 was reverse Polish

The stream

$B C + 7 \times A =$

was interpreted by

1. placing all operands on the stack

2. executing instructions by taking operands from the stack and returning results to the stack

so the above was interpreted as

Symbol	Operation	Stack
	being examined	taking place afterwards

B		B
C		B, C
+	B+C	B+C
7		B+C, 7
x	(B+C) x 7	(B+C) x 7
A		(B+C) x 7, A
=	(B+C) x 7 stored in A	[empty]

Program is converted to reverse Polish for execution

Instructions are 12-bits long (4 per word)

Memory Organization

The Program Reference Table holds constants and other permanent data for the program

The Stack holds temporary data for expression evaluation and local variables

The program segment string holds the program

There is also a P register containing the offset in the program segment of the currently executing instruction

There are hardware registers holding the locations of these three areas of memory

Since all memory references are relative to these registers

Areas can be moved by the executive program

Multiple programs can be in execution simultaneously - by switching these registers the program currently being executed can be changed

All memory is divided into segments - which are the unit for transferring to backup storage

All pointers to a segment contain a bit indicating if the segment is present or must be swapped in

Relative Addressing

(almost) All references to memory are relative to

Base of Program Reference Table

Location of last MSCW in stack

Location of current instruction

PRT+7 containing base of stack

The architecture supports an almost direct mapping from an ALGOL 60 program to internal data structures

Array Addressing

An array descriptor pointed to the contents of the array and also contained the length of the array

Indexing and bounds checking were hardware operations

If an array was not in memory the "presence" bit in the descriptor would cause an interrupt so that the array could be swapped in

Procedure calling

To call a procedure the system would

- push a Mark Stack Control Word onto the stack
- push the arguments to the procedure onto the stack
- call the procedure resulting in pushing a Return Control Word onto the stack

The procedure would leave the return value on top of the stack

- The location of the last MSCW in the stack was held in the F register
- All of the MSCWs were linked together to facilitate stack operations

There was a special procedure descriptor that, if loaded as an operand, would call a procedure with no arguments

Operations of the B 5500

Instructions were called "syllables" and were coded in 12 bits

Literal Call Syllable

- Pushed a number from 0 to 1023 on to the stack

Descriptor Call Syllable

- Pushed an operand fetched from
 - Program Reference table
 - Program segment string [code segment]
 - An array
 - Result of calling a procedure
- Both numbers and descriptors could be fetched to the stack with this syllable

Several relative branching syllables

There was a full set of arithmetical and logical operations

- Unary operations would operate on the top item on the stack and replace it with the result
- Binary operations would combine the top two items on the stack and replace them with the result

Relational operations left a boolean on top of the stack

Store syllables would store the top of stack in a location specified by a descriptor on top of the stack

Some syllables would duplicate/delete/permute the top elements of the stack

Many syllables would convert from integer to real as necessary

Assorted miscellaneous operators

Other comments

Interrupt handling

- Only the A processor could handle interrupts
- If the B processor gets an interrupt the registers of the two processors are swapped and the interrupt is not handled by the A processor

Note that by changing a very few registers the computer could change which program it was executing

Burroughs extended this line of computers with the B 6500 and B 7500