Amoral

Written by Paul Robson November 2020

# Introduction

Amoral is a 16-bit programming language designed for the 6502, though it is not limited to it. It is based around an extremely simple conceptual virtual machine that has one 16-bit register; operations are all on this 16-bit register. So, one may add to it, save or load it, and so on. This register is commonly referred to as R.

This has two direct consequences. It maps very easily onto the 6502, which has very few registers, and it is trivial to compile into machine code if required. In the 6502 implementation the 16-bit register is XA.

Amoral can generate either a p-code or 6502 code directly. This gets round the difficulties of the processor; p-code is compact but too slow ; 16-bit 6502 code is fast but space inefficient.

# Language

Syntactically the language is very simple. A statement is either an operation on the accumulator, or ‘something else’ – a procedure call, a structure, whatever. Multiple statements are grouped in curly brackets.

Outside quoted strings and characters, case is ignored. Commenting is like Python, anything after a ‘#’ character is ignored in a line.

## Terms

The following terms are supported.

|  |  |  |
| --- | --- | --- |
| Type | Example | Description |
| Constant | 42 | Integer constant |
| Hexadecimal | $F7 | Hexadecimal constant |
| Character | ‘C’ | ASCII value (32-127 only, not ‘) |
| String | “hello world” | ASCIIZ string (value is the address) |
| Variable | Count hello\_world.42 | A 16-bit static variable. |
| Address | @count | Variable address |

## Identifiers

Identifiers must begin with a letter and after that can be alphanumeric, underscores or a full stop. It is advised, but not enforced, that the full stop be used for modular code.

## Binary Accumulator operations

Accumulator operations all involve a term, which is a variable or a constant, all 16 bits. It is not possible to work with 8-bit data directly. The operators are as follows:

|  |  |  |
| --- | --- | --- |
| Operator | Examples | Function |
| None | 42 count | Loads value into accumulator |
| ! | !count | Stores value into variable. (!constant does not work) |
| + | +5 | Add |
| - | -count | Subtract |
| \* | \*$2A | Multiply |
| / | /4 | Divide |
| % | %9 | Modulus |
| & | & count | Binary and |
| | | | ‘X’ | Binary or |
| ^ | ^ 72 | Binary exclusive or. |

There is no concept of precedence in the language. If you write a+2\*3 it is not a+6 it is (a+2)\*3, because there is no concept of an expression, merely terms and operators.

## Unary Accumulator operations

Unary operations have no term, and simply operate on the 16-bit register

|  |  |
| --- | --- |
| Operator | Description |
| >> | Logical Shift Right (bit 15 is cleared) |
| << | Logical Shift Left |
| ++ | Increment |
| -- | Decrement |

## Variables

Variables, which are all 16-bit single items, can be declared at any point. All variables are static, so if you want to do recursion that you will have to handle yourself (the procedure call will be fine).

Variables are either global, or local when defined inside a procedure. Local variables cannot have the same identifier as globals (because I say so!) or procedures.

Variable are initialised to zero at the start of the program.

A variable declaration looks like this.

var count,name,hello\_world\_42

## Procedures and Functions

Procedures and functions are the same thing. This is achieved using the 16-bit register. When a procedure exits, the value currently in the 16-bit register stays there, so a function can be created by simply putting the return value in that register

.

Entry is slightly different. If there are no parameters, then the procedure is called as a straight subroutine, but R is unchanged. So, you could write a unary function which doubles R as

proc double() { \*2 }

When there are parameters, the current value of R is lost, as it is used to copy parameters into their respective arguments (which, again, are static and unique like variables). The last parameter is passed in R, however, and the first thing the implementation of the procedure does is to store this last parameter.

Note you can use ‘proc’ or ‘func’ in the definition interchangeably, purely for readability purposes.

So, a simple procedure that adds three numbers.

Proc addup(a,b,c) { a+b+c }

called with

addup(42,count,’A’)

proceeds as follows (green code is the procedure executing itself)

1. copy the constant 42 to the local parameter a
2. copy the variable count to the local parameter b
3. load the constant ‘a’ (97) into R (as it’s the last parameter)
4. call the routine. ‘addup’
5. the first thing addup does is to store R in c, completing the parameter load
6. load a into R
7. add b to it
8. add c to it
9. return with the total of a,b and c in R.

Procedure parameters can be sequences of operators, but not function calls. So you can write :

addup(42,count\*2,”Hello world”)

though it’d be more sensible to use count<< because a left shift is far quicker than a 16 bit multiply. This may be a later optimisation (for +1,-1,\*2,/2) but don’t rely on it.

Procedures parameters can be ~ which means ‘use the current value of R’.

## Structures

Structures again resemble C.

Conditional structures have a sequence which is surrounded in brackets, which can be empty, ended by a conditional comparison to zero. e.g.

while (>=0) if (a==0) while (a+2<0) if (count<>0)

these can contain function calls unlike parameters. Note there are only four operators and at present all comparisons are against zero, though this may change to allow other constants later.

So, your loop may resemble

while (a >= 0) { print.constant(a) ; a--!a }

There is no direct equivalent of the for loop. Amoral has a structure ‘times’ which executes a code a given number of times, counting down until zero. At the start of each loop the R register contains the current count. On exit R is -1.

So, the following

times(6) { print.constant(~); }

would print 5,4,3,2,1,0. Note the use of ~ as a parameter to indicate the current value of R.

# System structure

– base library, var memory, free memory,

# P-Code.

every procedure is a 6502 one

no bank switching in code.