Appendix F: wAx 2.1 Addendum

wAx 2.1 includes the following changes:

- Go (the G tool) improves stack integrity for post-BRK continuation.
- The Register Editor can now be used to change the PC, to aid the aforementioned post-BRK continuation.
- The Register Editor now provides syntax to keep any register as it was, without providing a value.
- The Register display now includes visual representation of processor flags, in addition to the processor flag byte value.
- Variable substitution for addresses now allows hexadecimal strings, and substitution in arithmetic.
- Two "illegal" instructions' mnemonics and byte sizes have been changed.
- A buffer overwrite bug, which could prevent partial filename search from working in emulated environments (e.g., VICE) has been remediated.
- Operand arithmetic (+/-) now allows 8-bit operands.
- The File tool has several enhancements.
- The Character Helper plug-in has been replaced with a Cycle Counter plug-in.
- The Code Search now allows wildcard matching.
- Two subroutines (SyntaxErr, SetUserVal) have been added to the API jump table.

Be Excellent To Each Other,

Jason
October 2023

Go

.G

When the G tool is invoked with no address, execution continues at the address specified in the Register Editor. The return address to BASIC is not put on the stack with this syntax; rather the stack is left as it was after the last BRK point.

Example

```
.A 1800 LDA #$55
.A 1802 PHA
.A 1803 LDA #$44
.A 1805 BRK
.A 1806 NOP
.A 1807 PLA
.A 1808 BRK
.G 1800
```

This will push $55 onto the stack and change A to $44, at which point, execution will break, showing A to be $44 and PC to be $1807 (two bytes after the BRK).

If you then continue execution with G alone, the $55 is pulled from the stack into A.

Register Editor

Registers may now be set with

```
.;ac [xr] [yr] [pr] [sp] [pc]
```

where:

ac is the Accumulator as a valid hexadecimal byte, or ==

xr is the X Register as a valid hexadecimal byte, or ==

yr is the Y Register as a valid hexadecimal byte, or ==

pr is the Processor Status Register as a valid hexadecimal byte, or ==

sp is the Stack Pointer as a valid hexadecimal byte, or ==, but has no effect on the Stack Pointer

pc is the Program Counter as a valid 16-bit hexadecimal address

For any register, you may enter == instead of a hex byte. Doing this will leave the register's value unchanged.

You may change the Program Counter before issuing G alone (*see Go, p. 2*).

Note: The Stack Pointer is not changeable because there is no corresponding pre-SYS byte available for it. Arguments in the *sp* position are ignored.

Processor Status Display

The Register display now includes on/off annunciators for five processor flags:

- The Negative flag (N)
- The Overflow flag (V)
- The Decimal flag (D)
- The Zero flag (Z)
- The Carry flag (C)

When an annunciator is in reverse text, the corresponding flag was set at the end of execution. Otherwise, that flag was clear.

Note: wAx's post-execution code clears the Decimal flag because the BASIC environment cannot function if the Decimal flag is set. If the Decimal flag's annunciator is "on" in the Register display, it will be set at the next invocation of the G tool.

BASIC Variable Substitution

BASIC string variables may now be used in tools where only numeric variables were previously permitted, provided they are valid hexadecimal addresses, and provided they are the correct length for the context.

For example:

```
S$ = "F27A"
E$ = "F28A"
.D `S$` `E$`

A$ = "033C"
.* `A$`
```

Note: This type of substitution *can not* be used within the A tool (Assembly or Memory Editors).

Variable Substitution in Arithmetic

Additionally, variable substitution is now permitted in arithmetic operands:

```
10 INPUT "LETTER NUM";N
20 .A 1800 LDA #"@"+`N`
30 .A *       JMP $FFD2
40 SYS 6144
```

"Illegal" Instruction Changes

Two of the illegal instructions have changed mnemonics and lengths. These are:

- **$34** DOP zero page,x is now SKB (*skip byte)*
- **$3C** TOP absolute,x is now SKW (*skip word)*

Although the 6502 treats these as two- and three-byte instructions, respectively, wAx will assemble and disassemble (via the E tool) them as implied mode (one-byte) instructions.

This allows SKB and SKW to be used in selectors. For example:

```
.A * @P LDA #"%"
.A *    SKW
.A * @C LDA #","
.A *    SKW
.A * @A LDA #"&"
.A *    JSR $FFD2
```

Operand Arithmetic

Arithmetic operands now have a range of -FF to +FF. For example:

```
.A 1800 STA $9000+5
.A 1803 LDA ($FF-30),Y
.A 1805 STA @V+0F
.A 1808 LDX #"R"+80
```

File Tool

Changing Device Number

The current device number can be changed with

```
.F #device
```

Where *device* is a four- or eight-bit hexadecimal number between 8 and B (11). Subsequent disk operations (of the File tool, the Save tool, and the Load tool) will use this device number. Using another device number with a BASIC command will also change the device number that these tools use.

Running Disk Commands

A command can be sent to the current device with

```
.F command
```

Where *command* is a valid command for the device. For example:

```
.F S:BAD-FILE
.F R:NEW-FILE=OLD-FILE
.F CD:../
```

After each command, the status line reported by the device is displayed. Devices will have different capabilities and status messages, so check the documentation.

Viewing Directories

For devices that support directories (SD2IEC, etc.), directories are shown in reverse text followed by /. Find directories in the current path with

```
.F "/"
```

Plug-In: Cycle Counter

Cycle Counter shows how many processor cycles were executing during the subroutine.The Cycle Counter plug-in replaces the Character Helper plug-in in wAx 2.

Installation

```
.P "CY"
```

Usage

```
.U addr
```

where *addr* is a valid 16-bit hexadecimal address.

Cycle Counter will count the number of cycles in the routine at the specified address by running the routine. The count does not include the RTS at the end of the routine. Routines assessed with Cycle Counter should end in RTS. After 65535 cycles, the count rolls back to 0.

When executed in direct mode, Cycle Counter will print the number of cycles on the screen *as a decimal number*.

When the plug-in is invoked in a program, the number of cycles is stored in the numeric UU variable, instead of being printed to the screen.

```
.A 1800 LDY #100
.A 1802 @@ DEY
.A 1803 BNE @@
.A 1805 RTS
.P "CY"
.U 1800
 501
```

Code Search Wildcards

Code search strings may now contain any number of wildcard characters (=). The wildcard with match any character. For example:

```
.H D000 LDA $6=
.H E000 JS= $FF==
.H D000 ==A ($==),Y
```

Additional API Subroutines

SyntaxErr

Display context-sensitive syntax error

Call Address: **$A039**

Affected Registers: N/A (ends execution)

If a user provides invalid input, you may call SyntaxErr to inform the user of this condition.

If SyntaxErr is called in direct mode, wAx will display a question mark, and then return to the wAx prompt.

If SyntaxErr is called during program execution, a BASIC ?SYNTAX ERROR IN LINE nnnn is thrown.

SetUserVar

Set FAC1 and/or BASIC variable UU

Call Address: **$A03C**

Affected Registers: Accumulator, X, Y

Set Accumulator (high byte) and Y (low byte) before calling SetUserVar. SetUserVar will set FAC1 to the floating-point representation of this 16-bit value, regardless of operating mode.

If SetUserVar is called in direct mode, only FAC1 will be set. If SetUserVar is called during program execution, the numeric variable UU will also be set to this value.