Quick and Dirty
Routines for the Sweet-16

BY STEVE WHEELER
P. O. Box 15
U. S. Naval Facility Antigua
FPO Miami 34054

The Apple II computer contains in its firmware a 16-bit pseudo-machine called SWEET-16. A subroutine call to SWEET-16 causes all following code to be interpreted as instructions to the 16-bit pseudo-machine until a return to 6502 instruction (RTN) is encountered. SWEET-16 is a powerful adjunct to the Apple II computer, and several programs have been written for the Apple containing SWEET-16 code, including programs to renumber or to recover integer BASIC programs. Up until now, only Apple has had an assembler which would recognize SWEET-16 mnemonics, which meant that all other programmers have been forced to hand-assemble SWEET-16 code, and insert it into their programs as hex data. The following program is a modification to the cassette version of the S-C Assembler II which gives it the capability to recognize and correctly assemble SWEET-16 mnemonics.

How It Works

The assembler is a line-oriented, fixed-field assembler. Each line in the program is copied from the text file into a buffer at $0200 prior to assembly. This program grabs control from the main assembler at location $1477, by replacing the instruction there with a hook to the SWEET-16 assembly routines. The branch to a jump is necessary to avoid disturbing the instruction at $1479. What this does is cause these routines to take control if the entry in the instruction field of the line being assembled is not recognized as a pseudo-op or a 6502 instruction mnemonic.

The mnemonic is then compared with entries in the instruction table. Spaces are ignored, with one exception which is covered in the section on program limitations. If there is a match, a type byte is used to determine which routine(s) will be used to produce the final opcode, and to return to the main assembler at the proper point.

The instruction table format is as follows: the letters in the mnemonic are in ASCII code with the high bit set on the last character only. No space characters are included. The mnemonic is followed by an opcode byte, and then by a type byte. The end of the table is signified by table entries of $FF.

Routines Used in the Main Assembler

As written, these routines will work only with the cassette version of the S-C Assembler II. The following explanation is for those who wish to adapt the routines to another assembler.

GTCH – This routine loads the accumulator with a character from the line buffer, and converts the character to true ASCII (high bit clear). The Y register is used as the index into the buffer. Location $06 is used to store the value of Y between calls. Before returning, the character in the accumulator is compared to a space, the carry flag is cleared, and Y is incremented and stored into $06.

BLD – This routine updates the location counter, and builds the file of assembled code if called during pass #2. The value in the X register is used to determine which pass is in progress.

BYT1 – This is the entry point for single-byte opcodes.

BYT2 – This is the entry point for branch instructions. This returns to the main assembler just after an opcode would have been obtained, and just before the calculation of the relative displacement.

BYT3 – This is the entry point used for the SET opcode. It is the second instruction of the subroutine which handles the .DA pseudo-op.

NOPE – This is the return point if a match is not found. It prints the error message "BAD OPCODE AT LINE", aborts the assembly, and returns the assembler to command mode.

How To Use the Program

Before assembling this program, one patch must be made to the assembler. Location $1010 must be changed from $1C to $1E to move the start of the symbol table past these routines. If this is not done, the program will be stored on top of its own symbol table, and the assembly will be aborted.

The following mnemonics are recognized:

Non-register operations	Register operations
BC	ADD R
BK	CPR R
BM1	DCR R
BP	INR R
BR	LD R
BS	LD @R
BZ	LDD @ R
RS	POP @R
BM	POPD @R
BNC	SET R
BNZ	ST R
BNM1	ST @R
RTN	STD @R
	STP @R
	SUB R

Spaces within a mnemonic are ignored. The register number is specified by one ot two decimal digits following the 'R' for a register operation.

Limitations of the Program

These routines were designed as a minimal add-on to an existing assembler. As such, there are no error checks written into them. The following are "gotchas":

1. The subroutine REG, which calculates the register number for a register operation, assumes that the first character following the 'R' is a number, a space, or an end of line token. If the second character is not one of those, it is treated as a number, and the previous character is assumed to be a '1'. This means that ADD R 3, ADD R%3, ADD R13, ADD R03, and ADD RQC will all produce the opcode 'AD'.
2. The assembler requires a fixed-format line, and assumes that a particular index into the buffer will point to the first character of the operand. This feature has been sidestepped for the SET instruction, but not for branches. This means that a two-character branch mnemonic must be followed by two spaces before the branch destination, a three-character branch mnemonic must have one space between it and the destination, and the four-character branch mnemonic (BNMI) must be immediately followed by the destination, with no intervening spaces.
3. Do no put a comma between the SET mnemonic and its operand. Just leave a space between them. Example:

```
SET R4 BUFF
```

These are the limitations that I know of. If anyone discovers other flaws, or comes up with a fix for these, I would appreciate hearing about them.

Good Points

1. Since these routines only produce the opcode, the host assembler handles all operands. Therefore, operands in branches and the SET instruction can be decimal, hexadecimal, or labels.
2. Where else can you get an assembler which handles SWEET-16?

Final Comments

If the above limitations are respected, assembly of SWEET-16 code is correct and almost uneventful. I say almost because I nearly knocked my Apple off the table the first time the routines worked correctly, and I still have to stifle a grin whenever I use them.

These routines are basically a "quick and dirty" implementation of a SWEET-16 assembler. Since I have only a few months of experience with assembly language, writing them was anything but quick. My purpose in publishing them is simple: I'm hoping that similar routines, suitably improved, will be incorporated in future assemblers for the Apple II computer. It's time that assemblers for the Apple were capable of recognizing and assembling ALL instructions which the Apple can execute.

I would also like to dedicate this program (Why not? It's done for books.) to Chris Johnson, who gave me my start in assembly language programming, and who provided the original idea for this program.

References

The Apple II computer is produced by the Apple Computer Company of Cupertino, California. The S-C Assembler II is available from S-C Software, P.O. Box 5537, Richardson, Texas 75080. The definitive article on SWEET-16 originally appeared in the November 1977 issue of BYTE magazine.