# Machine Language: The Wonderful Wedge

Jim Butterfield

Adding new commands to Basic seems an impossible task at first glance. The Basic interpreter is frozen forever in ROM chips, and unless you're the adventurous type who can program your own EPROM chips, it seems that there's no way in.

It can be done. A small but important part of the Basic interpreter is located in RAM memory. It's written there during system initialization and is available for you to change.

The subroutine is called **CHRGET** (Character Get), and all 6502 Microsoft Basic implementations use it. Every time the Basic interpreter wants to get a character from the Basic statement it is executing, it calls CHRGET.

Here's where you can find subroutine CHRGET in some 6502 systems:

| | |
|---|---|
| **KIM -** | **C0 to D7 hexadecimal** |
| **SYM -** | **CC to E3** |
| **AIM -** | **BF to D6** |
| **OSI -** | **BC to D3** |
| **Apple -** | **B1 to C8** |
| **Early PET -** | **C2 to D9** |
| **PET/CBM -** | **70 to 87** |

Our description here will refer to the PET/CBM version, Upgrade and subsequent ROMs.

## How it works

Let's look at the CHRGET subroutine in detail.

```
0070 E6 77  CHRGET  INC POINTER
0072 D0 02          BNE CHRGOT      ;skip next instruction
0074 E6 78          INC POINTER + 1
```

Locations 77 and 78 contain the address of the last Basic character obtained. The above coding bumps the pointer to the next address, adjusting the high order address if necessary.

```
0076 AD xx xx CHRGOT LDA xxxx       variable address
```

The address indicated with xxxx above normally points at your Basic program or at a direct Basic statement you have typed in. Note that the address itself has been modified by CHRGET, above.

```
0079 C9 3A          CMP #':         ;ascii colon or higher?
007B B0 0A          BCS EXIT        ;yes, exit subroutine
```

The above coding tests two things. If the new character is a colon, meaning end of Basic statement, we will exit with the Z flag set to one. If the new character is higher than ASCII 9 (hex 39), we will exit with the Carry flag set to one. The meaning of these flags will be discussed in a moment.

```
007D C9 20          CMP #'          ;is it a space?
007F F0 EF          BEQ CHRGET
```

We know that Basic ignores spaces; this is where it happens. If we find a space, we go back and get another character.

```
0081 38             SEC
0082 E9 30          SBC #$30
0084 38             SEC
0085 E9 D0          SBC #$D0
```

This seems to be a curious bit of coding: we subtract 256 from the A register, in two steps, which leaves it with its original value! The point is this: if the A register contains a value less than ASCII zero (30 hex), the Carry flag will be set to one; otherwise, it will be cleared to zero. The Z flag, too, will be affected: it will be set if we have obtained a binary zero.

```
0087 60             RTS
```

## What the flags mean

The flags are often checked by the calling routines. The Z flag will be set on if we have found an ASCII colon (end of statement) or a binary zero (end of Basic line).

The Carry flag will be cleared to off if the character is an ASCII numeric, zero to nine (30 to 39 hex); otherwise it will be set on.

## How the subroutine is called

CHRGET is called many times during the interpretation of a Basic program or a direct statement. It normally obtains data from the active program; but it is also used to obtain information from DATA statements or keyboard input during READ or INPUT activities. In such cases, the pointer at 77 and 78 is swapped out temporarily.

The Basic interpreter also frequently calls CHRGOT (address 0076) to re-obtain and check a previously obtained character.

From time to time, the pointer at 77 and 78 is used as an indirect address by the interpreter; when we start tampering with the coding of the subroutine, we must be sure to leave the pointer intact in its normal place.

Finally, there is a rare call that is made to the subroutine at address 7D (Compare to space); it doesn't happen often, but we must watch for it.

Keep in mind that the subroutine does not affect the X or Y registers.

## Wedging it in

To fit in the extra features, we must "patch" the CHRGET program and connect it to our own code. The patch will destroy some of the existing code, of course, and we must carefully replace it.

There are two places we can insert the patch: at the beginning of CHRGET, or a little distance past CHRGOT. The first location will go into action only when a new character is called up by the interpreter. The second location would be invoked more often, since CHRGOT is called to recheck a previously obtained character.

Let's use the first location; and let's put in a simple do-nothing wedge for starters. Call up the Machine Language Monitor and set up the following memory locations as shown:

```
027A:   E6 77 D0 02 E6 78 4C
0282:   76 00 xx xx xx xx xx
```

The first six locations exactly match the coding at CHRGET. Now we'll put in the patch with:

```
0070:   4C 7A 02 02 E6 78 AD xx
```

Leave the Machine Language Monitor and play with Basic for a moment. Everything still works. It looks like we have found a way to penetrate Basic...but we haven't done anything yet.

## A tiny example

Let's write a very small wedge to recognize an "@" sign and break to the monitor if it is seen. Not much in the way of power, but it will show how the technique is used. We'll continue to use the patch at 0070.

To get the character we plan to analyze, we'll have to use indirect, indexed addressing. The pointer is of course at 77 and 78, and we must set the Y register to zero. Since we must not affect the Y register, we must first save its contents, and restore them before we finish.

So our coding will follow the following pattern: STY WORK, to save Y; LDY #0, LDA (POINTER), Y to get the character; LDY WORK, to restore Y; CMP #'@, to check for the @ character in A; BEQ BREAK if we find it; and JMP CHRGOT to return if not. BREAK will have the BRK instruction to go to the Monitor. Let's do it.

```
027A:   E6 77 D0 02 E6 78 8C A0
0282:   02 A0 00 B1 77 AC A0 02
028A:   C9 40 F0 03 4C 76 00 00
```

We have arbitrarily picked address 02A0 as our Y Save location. Now the patch to implement the wedge:

```
0070:   4C 7A 02 xx xx xx AD xx
```

Return to Basic. Try statements which do not contain the @ sign, and others which do.

## Final remarks

You're ready to try your hand at more ambitious wedge inserts. Be careful: remember to save X and Y if you use them, and restore them later. Keep in mind that the larger your wedge program, the slower Basic will run. Look for quick tests: for example, many wedge programs will exit instantly unless the statement was input as a Direct command...this can save a lot of time on a running program.

Watch that you don't conflict with other wedge programs, like Trace, Toolkit, or the DOS wedge program. It takes a lot of careful coding; but the results can be dramatic. ©