COMPUTE! ISSUE 36 / MAY 1983 / PAGE 244

Versatile Data Acquisition with VIC

Doug Homer and Stan Klein

This simple method of adjusting the VIC's internal jiffy dock can slow it down to match your timing needs making possible "variable speed" machine language subroutines. You can save a good amount of money by transforming a VIC into this special-purpose tool. You can even use this to speed up games.

Home computers are finding their "homes" in labs, more and more frequently. Their flexibility and low cost make them excellent substitutes for more expensive special equipment. One common use is as a data acquisition device. Data acquisition systems monitor and record information on experiments in progress. For example, a chemist may use a special electrode to measure the concentration of a particular component in a chemical solution. As the concentration changes, the electrode sends a varying voltage to an analog-to-digital converter. The converter changes the voltage signal to binary data which can be recorded and stored for later analysis.

To log the data, the chemist could use a special-purpose data acquisition system perhaps costing thousands of dollars and useful only for a particular type of experiment. On the other hand, a microcomputer could be programmed to perform the same function. Moreover, to perform another type of experiment, the chemist need only modify the program instead of buying new equipment. When the data is stored, the computer might also be useful in analyzing it.

Surprisingly Simple

There is a surprisingly simple method for converting the VIC into a data acquisition system. A good acquisition system is based on a clock which uses interrupts to sample the user port at adjustable, fixed intervals. Data acquisition software is usually complicated because you must worry about interrupts generated from the jiffy clock.

A simpler scheme is to append the data acquisition routine to the front of the interrupt service routine which is already functioning in connection with the jiffy clock. Every 16.667 milliseconds, VIC interrupts whatever it is doing to look at the keyboard and update the jiffy timer. Here's how to attach your own program to the jiffy service routine and how to set the jiffy clock to any rate of data acquisition.

To change the number of interrupts per second, just POKE different numbers into the low timer latch (37158) and the high timer latch (37159). Under normal operating conditions, these bytes are loaded with 137 in the low latch and 66 in the high latch. An interrupt is generated and the latches are reloaded into the counters whenever the counters are

decremented to zero. The number of cycles between interrupts is two cycles greater than the number in the latches.

You might expect the counter to be loaded with 16667 less two, since the normal interrupts are every 1/60 of a second; but 66*256 +137= 17033 rather than 16665. This means simply that the "1 MHz" counter decrements at $1.022*10^6$ Hz, not at an even rate of $1.00*10^6$ Hz. So, to make the jiffy clock interrupt at a rate different than the normal 1/60 per second, just multiply the desired number of microseconds per interrupt by 1.022 and subtract two from that number. Example: for a millisecond interrupt (1000*1.022)-2 = 1020, so you would POKE 3 into the high byte at location 37159, and 252 into the low byte at location 37158 (3*256 + 252 = 1020) - and now you have an interrupt every millisecond.

There are limits to this method of changing the jiffy clock to produce varied interrupts. At the slow end, the largest number that could be loaded is $FFFF, or 65535. For the longest time interval between interrupts, the number of microseconds would be (65535 + 2)/l .022 = 64126. The fast end limit is set by the percent of time remaining for BASIC. This percent is derived by (L-IR)/(L + 2), where L is the number POKEd in the timer latch described above, and IR is the number of cycles taken up by the unmodified interrupt service routine.

Interruptions Can Make Your Games Run Faster

Ottis Cowper, Technical Editor

This is a very powerful programming technique, *the interrupt driven subroutine,* which has a much wider range of applications than merely gathering data from the user port. For example, how would you like your computer to handle two jobs at once? Actually, the 6502 microprocessor is a sequential device and can only do one operation at a time, but the VIC's hardware interrupts occur so frequently (60 times per second) that a machine language interrupt routine can appear to work concurrently with BASIC.

A Demonstration

As a demonstration, make the additions and changes shown in Program 1 to the program in the article. (This demonstration is for the *unexpended* VIC and requires a joystick. Remove or disable any expansion modules.) Since the DATA statements contain a machine language routine, they *must* be typed in exactly as shown. Be sure to save a copy of the program before you RUN since an error in an interrupt routine almost always causes your system to lock you out. For those interested in the operation of the routine, a disassembly of the code is provided in Program 2.

When you RUN the program, you should see a bar appear in the center of the screen. Try moving your joystick left and right and notice how smoothly the bar moves. Type in a new value for the high and low bytes of the timer. Higher timer values slow down the bar movement; lower values speed it up. Compare this to the slow and jerky movement

you're used to in BASIC, and imagine how an interrupt joystick or character movement routine would improve your favorite game.

The main point is that the joystick reading and bar movement are totally independent of BASIC. To prove this to yourself, hit the STOP key. You'll see the message BREAK IN 35. The BASIC program has ended, but the interrupt routine is not affected. The bar movement continues as before. To disable the routine, hit the RUN/ STOP and RESTORE keys at the same time.

How To Add It To Your Programs

Here is the procedure for adding an interrupt driven routine to your BASIC program (example lines from the program given in the article are noted in parentheses):

1. Reserve room for the new routine somewhere in memory (line 10).
2. Load the machine language code into the protected area (line 15).
3. Disable interrupts, load the address (known as the "interrupt vector") of the new routine into locations 788 and 789, and re-enable interrupts (line 20).
4. If necessary, modify the speed of the interrupt routine by adjusting the rate of the jiffy clock (line 30).
5. It is *absolutely* essential that the appended interrupt routine end with a JuMP to the normal ROM interrupt handling routine (for the VIC, this would be JMP $EABF).

Program 1: Demonstration Program

```
11 PRINT"{CLEAR}"
12 FORI=38400TO38905 : POKEI, 0 :NEXT
13 POKE 1,8:POKE2,10
14 FORI=0TO2:POKE7909+I,160:NEXT
15 FORZ=0TO69:READQ:POKE(28*256+Z),Q:NEXTZ
22 DATA 166,1,164,2,169,127,141,34,145,173
23 DATA 31,145,41,16,240,26,173,32,145,41
24 DATA 128,208,35,192,21,240,31,169,32,157
25 DATA 220,30,232,200,169,160,153,220,30,24
26 DATA 144,16,224,0,240,12,169,32,153,220
27 DATA 30,202,136,169,160,157,220,30,134,1
28 DATA 132,2,169,255,141,34,145,76,191,234
35 GOTO35
```

Program 2: Disassembly Of Machine Language; Routine In Program 1

```
1C00 A6 01      LDX $01
1C02 A4 02      LDY $02
1C04 A9 7F      LDA #$7F
1C06 8D 22 91   STA $9122
1C09 AD IF 91   LDA $911F
1C0C 29 10      AND #$10
1C0E F0 1A      BEQ $1C2A
1C10 AD 20 91   LDA $9120
1C13 29 80      AND #$80
1C15 D0 23      BNE $1C3A
1C17 C0 15      CPY #$15
1C19 F0 IF      BEQ $1C3A
1C1B A9 20      LDA #$20
1C1D 9D DC IE   STA $1EDC,X
1C20 E8         INX
1C21 C8         INY
1C22 A9 A0      LDA #$A0
1C24 99 DC IE   STA $1EDC,Y
1C27 18         CLC
1C28 90 10      BCC $1C3A
1C2A E0 00      CPX #$00
1C2C F0 0C      BEQ $1C3A
1C2E A9 20      LDA #$20
1C30 99 DC IE   STA $1EDC,Y
1C33 CA         DEX
1C34 88         DEY
1C35 A9 A0      LDA #$A0
1C37 9D DC IE   STA $1EDC,X
1C3A 86 01      STX $01
1C3C 84 02      STY $02
1C3E A9 FF      LDA #$FF
1C40 8D 22 91   STA $9122
1C43 4C BF EA   JMP $EABF
```

There are approximately 220 cycles in the unmodified interrupt service routine; thus, if the number POKEd into the timer approaches 220, there will be no time available for anything other than attending to the interrupt service routine.

Here's how to add your own machine language routine to the jiffy clock service routine. Normally, when the decrementing counter hits zero, the operation is transferred to the interrupt service routine whose beginning address ($EABF) is stored in 788 and 789 ($0314 and $0315). By changing the address in 788 and 789, you can tell VIC to do additional instructions in machine language and then go to $EABF to run the normal service routine.

To change the address in 788 and 789, you must disable the interrupt enable register for the jiffy clock to allow the number in these locations to be changed. POKEing location 37166 with 128 will disable the interrupt; after the addresses in 788 and 789 have been changed, POKEing location 37166 with 192 will enable the interrupts again. Here's a sample program:

```
10 POKE52, 28:POKE56, 28:REM SETTING UPPER ~
   BOUNDARY FOR BASIC
15 FOR Z=0 TO 9:READ Q : P0KE( 28*256+Z ), Q :N
   EXT Z:REM MACHINE PROGRAM IN PAGE 28
20 POKE37166, 128 :POKE788,0 :POKE789, 28:POK
   E37166.192
21 REM LINE 20 CAUSES THE INTERRUPT TO NO
   W GO TO PAGE 28
25 DATA 173,16,145,157,0,29,232,76,191,23
   4
30 INPUT"LOW";N1:INPUT"HIGH";N2:POKE37I58,
   N1:POKE37159,N2
31 REM LINE 30 CHANGES THE TIMING OF THE ~
   INTERRUPT
```

The machine language program in line 25 disassembles to:

```
1C00 LDA $9110;   Get data from user port
1C03 STA $lD00,X; Store data in page 29 ring buffer
1C06INX;          Increment pointer for ring buffer
1C07 JMP $EABF;   Jump to normal jiffy service routine
```

This program can be used as a guide for setting up the jiffy clock for timed data acquisition. One additional consideration in terms of the percent of time left for BASIC: the above program has added an additional fourteen cycles which must be added to the IR variable. Exercise caution if data is to be gathered at faster than half-millisecond intervals.