INTERRUPTS

Take a moment to pretend that it's your turn to fix dinner for your family of nine people. You need to start dinner at 3, and have it ready by 5. So you decide to fix a tasty gourmet dinner that needs to be stirred every ten minutes.

Now, are you going to just stare at the food, stir it after 10 minutes, stare at it again, stir it, stare at it? Probably not. You may grab a book, type some emails, play some Xbox, or do something else while the 10 minute timer on your stove is running.

When the timer rings, are you going to finish what you were doing before stirring the dinner? Even if you had one hour before you finished playing an Xbox game? Chances are the food would burn to a crisp if you waited to stir it. Nope, you would **interrupt** whatever you were doing to stir the food, and then you would return to your relaxing until ten more minutes passed.

Sometimes, when you design a calculator program, you have something that needs to happen on a consistent, timely basis, no matter where in the program you are. In Axe, this is different from using subroutines with sub(Lbl). A subroutine will only run when the axe program **asks** it to run. Thus, a subroutine will only run as many times and as fast as it is called by sub(). But an interrupt routine will run after a certain amount of time has passed, so no matter what happens in your program, your interrupt routine will run the same number of times every minute.

Now that you're excited about using interrupts, let's put them into practice. Remember that when you use interrupts, you cannot use L_2 for data storage.

Hopefully you remember the fun pong game that you created as an example Axe program. At the end of the game you saw how many times you hit the ball. We are going to edit this game so that you can also see how many seconds you survived.

Below is the source code for your new PONG game. Anything highlighted in blue is new, and will be commented on.

```
.PONG WITH INTERRUPTS
"PONG"→Str1
"SCORE:" →Str2
"SURVIVED:" →Str3        ; We want the game to display how many seconds long we
                         ; survived, so Str3 will hold the word “SURVIVED” to display
                         ; at the score screen.

[000000000000FFFF] →Pic1
[0000182C3C180000] →Pic2
DiagnosticOff
ClrHome
Output(6,3,Str1)
Pause 1000
0→S-1→D

0→T
                         ; T, the amount of time, needs to start at Zero.
44→Z*256→X
10→Y
sub(HT)

FnInt(TI,6)              ; Our interrupt routine is coded at Label TI.  Turn on the
                         ; interrupt routine at the slowest speed.
Repeat getKey(15)
If getKey(2) and (Z≠0)
Z-2→Z
End
If getKey(3) and (Z≠88)
Z+2→Z
End
X+V→X
Y+D→Y
If Y>70
Goto D
End
If Y=0
sub(HT)
End
If Y=54 and (abs(X/256-Z)<8)
sub(HT)
S+1→S
End
If X/256=0 or (X/256=88)
-V→V+X→X
End
ClrDraw
```

```
Pt-On(Z,54,Pic1)
Pt-On(X/256,Y,Pic2)
DispGraph
End
Lbl D
ClrHome
Disp Str2,S→Dec,i

Disp Str3,(T/118) →Dec,i  ; Our interrupt routine ran at 118 times a second, and each
                          ; time it ran, T was increased by 1.  Divide by 118 to get
                          ; the number of times each second the routine ran.
                          ; IF YOU ARE USING A TI-83+ SE OR A TI-84+, change
                          ; the value 118 to 108.

                          ; Be aware that if you happen to survive for more than nine
                          ; minutes, the number of seconds you survived will be
                          ; inaccurate, because T went over 65535 and had to reset
                          ; itself to 0.

LnReg                     ; Return interrupts to normal.
Return
Lbl HT
rand^512-256→V
-D→D
Return                    ; Before, this line was not needed because the line before it
                          ; was the end of the program.

Lbl TI                    ; Our interrupt routine increases T by 1.
T+1→T
```

FUN FACTS

- Depending on what speed you choose, here's how many times your interrupt routine will run per second:

Speed	Standard TI-83+	TI-83+ SE or TI-84+
0	560 Times a Second	512 Times a Second
2	248 Times a Second	228 Times a Second
4	170 Times a Second	146 Times a Second
6	118 Times a Second	108 Times a Second

- Why is L_2 needed for interrupts? Well, the way the processor works, 257 bytes of RAM are needed to tell the processor where your interrupt routine is. Furthermore, there's only 256 possible locations that this table can start at, and one of those locations is inside of L_2. So L_2 is big enough—and in the right place—to handle interrupts correctly.

- Axe uses IM 2 (called interrupt mode 2) for interrupts. There actually was a way that one could run interrupts without L_2, by using interrupt mode 1, IM 1. However, TI called dibs on IM 1.

- Can you have more than 1 interrupt routine? It's possible, but whether it is supported by Axe or not, it's not an easy feat. (Actually, it would be easier for ASM programmers than for Axe programmers) Whatever the case, having more than one interrupt routine means each routine is called at random.

- IM 1 is necessary for your calculator to perform normally when your game is finished. LnReg returns from IM 2 to IM 1, which is why you must use the command before your program exits.

- Many calculator games use interrupts to produce sound and music. In this way, no matter if the game speeds up or slows down, music and sound will always play at a consistent speed. It's also a good way to avoid a worst-case scenario: stutters between notes