Lets Make A Retro Game

Episode 11 – Scoring

In this episode we are going look at adding scoring to our game.

Our Megablast game we are making is based on Astrosmash for the Intellivision and in that game you can both increase your score by shooting various enemies plus loose score if any of the enemies make it to the ground.

This is another reason it makes a good example game, as we get to cover both increasing and decreasing a player's score.

So to get started let's look at how we are going to keep track of the players score.

To make displaying the score a bit easier we are going to use a special Z80 set of instructions that assist with Binary Coded Decimal (BCD) values.

What is BCD you ask? It is special number storage mode that instead storing the values 0-255 in each byte, it stores a single decimal digit (0-9) in each of the two nibbles (4-bits) in a byte i.e. you can store from 0-99 per byte of memory.

This does seem to be a bit of a waste, but what this allows us to do is easily display a decimal representation of the number with very little work (as we will go through below).

In our game we are going to use three bytes of Ram to hold the players current score, if you always make the last digit of the score zero, this will allow us to have seven digits or scores from 10 to 99 million.

So to make our scoring work we are going to need five things:

- A routine to display the current score to the screen
- A routine to add to the players score
- A routine to subtract from the players score
- Call the display routine when we are allowed to draw to the screen
- Call the routines to add and subtract points to and from the players score at the appropriate places in our game.

Displaying the Current Score

So first we need a routine to display our score on the screen, this is actually already in the supplied code, but I made a couple of slight changes to make it more readable so we will go through it here for completeness.

```
; Display the current score on the 2nd last row
DISPLAYSCORE:
      ; compare the 1st score digit with our lastscore value
      LD A,(SCORE)
      LD HL,LASTSCORE
      CP (HL)
      RET Z
```

This first part of the routine, checks that the 1st digit of our score has changed, by comparing it with the value stored in LASTSCORE. This prevents us from wasting time displaying the score if our score hasn't changed.

```
      ; setup our write to video ram
      LD HL,VRAM_NAME + 707
      CALL SETWRT
```

Next we set the current video ram pointer to be at position 707 (or row 22, column 3).

```
      ; starting at our last byte, write out each of the
      ; two digits per byte
      LD HL,SCORE+2
      LD B,3
SLP:
      ; output the two decimal digits currently in A
      CALL PRINTIT
      DEC HL
      DJNZ SLP
```

Here we loop through each of the three bytes for our score and for each one calling the PRINTIT subroutine that will display the two digits stored in our score value.

Basically PRINTIT takes each nibble in the byte and adds it to our character for zero in our pattern table and outputs that value to our screen name table.

```
      ; save the current score value into lastscore
      LD A,(SCORE)
      LD (LASTSCORE),A
      RET
```

To finish up we get our current score value and store it in the LASTSCORE byte so that we don't keep on writing to the screen if our score has not changed.

Adding to the Current Score

Next we need to create a routine to add to the current score. We will call this routine with the amount to add to the score in the Accumulator (A).

```
; Add A to the current score
; - Score is stored as three two nibble decimal values.
; - Displaying a fixed zero at the end this gives a
;   score range of 7 digits i.e max score is 99 million.
SCOREADD:
    PUSH DE ; save DE
    PUSH HL ; save HL
```

First we save the current values of the DE and HL registers on the stack as they will be changed by this routine, this way we only have to worry about A being changed when the routine is called.

```
    ; add value in A to the current 1st score byte
    LD HL,SCORE
```

The HL register is used to point to the byte of our Score that we are working on.

```
    LD E,A
    LD A,(HL)
    ADD A,E
    ; adjust into a two nibble decimal
    DAA
    ; save to 1st score byte
    LD (HL),A
```

In this section of our code, we put our value to add into E, then get the current value of our 1st score byte, add E to it, then use the BCD command DAA, which will handle changing the value into BCD format. The carry flag will be set if the number is larger than 99.

```
    ; now add any overflow to the 2nd score byte
    INC HL
    LD A,(HL)
    ADC A,0
    DAA
    LD (HL),A
```

In this section we add one to the next score byte if the carry flag was set from the 1st add.

```
    ; now add any overflow to the 3rd score byte
    INC HL
    LD A,(HL)
    ADC A,0
    DAA
    LD (HL),A
```

In this section we do the same for the 3rd score byte, I probably could have used a loop for this as these two sections of code are identical. Shows how there are always multiple ways of doing things.

```
    POP HL ; restore HL
    POP DE ; restore DE
    RET
```

Lastly we restore our HL and DE values from the stack and return to our calling code.

Subtracting from the Current Score

Next we need to add a routine so we can subtract from the current Players score, this is a little bit more complicated, but is a good example of subtraction with values larger than can be stored in a single byte of memory.

```
; Subtract A from the current score
; - Score is stored as three two nibble decimal values.
; - Displaying a fixed zero at the end this gives a
;   score range of 7 digits i.e max score is 99 million.
SCORESUB:
    PUSH DE ; save DE
    PUSH HL ; save HL
```

First we save the values in the DE and HL registers as we will be changing them.

```
    ; subtract value in A from the 1st score byte
    LD E,A
    LD HL,SCORE
    LD A,(HL)
    SUB E
    ; adjust into a two nibble decimal
    DAA
    ; save to 1st score byte
    LD (HL),A
```

Next we point HL to our 1st score byte of memory and then subtract the value passed in the Accumulator (A) from it, use the DAA command to adjust it back into a BCD value and finally we store the value back into our 1st score byte.

```
    ; now add any overflow to the 2nd score byte
    INC HL
    LD A,(HL)
    SBC A,0
    DAA
    LD (HL),A
```

Next if we overflowed i.e. made our 1st score byte less than zero, we need to subtract one from the 2nd score byte, adjust it back to a BCD value and store it back in the 2nd byte.

```
    ; now add any overflow to the 3rd score byte
    INC HL
    LD A,(HL)
    SBC A,0
    DAA
    LD (HL),A
```

Next we repeat the same code for the 3rd score byte. Once again a loop could have been used here.

```
    JR NC, SCORESUB2
    ; we have overflowed - set score to zero
    XOR A
    LD (HL),A
    DEC HL
    LD (HL),A
    DEC HL
    LD (HL),A
```

Now if we got an overflow on our 3rd byte then we have either made the score larger than 99 million (unlikely as we subtracted a number) or we have made our score negative. We don't want negative scores, so we just reset all three score bytes back to zero.

```
SCORESUB2:
    POP HL ; restore HL
    POP DE ; restore DE
    RET
```

Finally we restore the HL and DE registers so they are not affected by what happens inside this routine.

Calling the Display Routine

Another thing we need to do is call the Score Display routine when we are allowed to write to the screen (and not cause graphical issues). So the best place for that is our VDP blank routine. The sample code already includes this call as follows:

```
; This is our routine called every VDP interrupt during normal game play
; - Do all VDP writes here to avoid corruption
VDU_WRITES:
    CALL DISPLAYSCORE
    RET
```

Adding Scoring to our Game

Right so now we have all our routines ready, all we need to do is call our SCOREADD and SCORESUB routines at some appropriate places in our code.

So 1st we will subtract from the score when an enemy reaches the bottom of the screen.

In the MOVE_ENEMIES routine find the comment:

```
; decrease score
```

And after that add the following two lines:

```
LD A,1
CALL SCORESUB
```

So this is going to decrease our effective score by 10 points every time one of the enemy objects reaches the bottom of the screen.

Next we will increase the score when we shoot one of the enemies.

Further down in the MOVE_ENEMIES routine find the comment:

```
; later we will:
```

And replace it with the following code:

```
; increase our score for hitting the asteroid
; Note: later we will vary the score by type of enemy
LD A,2 ; 20 points per asteroid hit
CALL SCOREADD

; later we will:
; - explosion sound
; - animate enemy
```

So each time we shoot one of the enemies we will add 20 points to the players score. Later on we will add on a different amount of points based on the type of enemy that has been destroyed.

That's all for this episode, in the next episode we will handle the player getting killed and displaying the number of remaining lives on screen.