Home Assignment 2: **2D Random Walk**

Name: Daniyar Zhakyp

ID: 201774605

Section: 3

The whole strategy of writing the assembly code for the Random Walk process, was to add +1 or -1 in random order, such that we start from predefined initial position ((0,0) in our case), and make **single** consecutive steps. For that purpose, I used the same randomization technique that was used in the manual, however, I stored the value of register AX into DX to make several manipulations with it to get either 1 or -1.

If number of parity bits in line 214 is odd, then AND DX with 0001h, to get either all zeroes (0000h), or 0001h itself. We would like to get 0001h in order to take 2's complement of it and get -1, therefore we compare this value with 0000h, and add 1 if needed. After successfully obtaining -1, we store it in the variable ONE (see line 236 in Figure 1).

```
204 RAND1    PROC NEAR
205
206          PUSH AX      ;to save the registers before the call
207          PUSH BX
208          PUSH CX
209          PUSH DX
210
211          MOV  AX, RANDOM1    ;enter the seed
212          MOV  BL, 2Dh        ;extract the taps
213          AND  BL, AL         ;by masking with AND
214          JNP  odd            ;check the number of ones,
215
216
217          MOV DX, AX          ;use the spare register DX to obtain +1 or -1
218          AND DX, 0001h       ;AND DX with 0001h to get 0001h back or 0000h
219          CMP DX, 00h         ; check if it's 0000h
220          JNE okay            ; if yes, add 1 to it
221          ADD DX, 1
222 okay:
223          MOV ONE, DX          ;move dx to variable ONE
224          SHR  AX, 01h        ;if even just shift
225          MOV  RANDOM1, AX
226          JMP  rend
227
228 odd:     MOV DX, AX
229          AND DX, 0001h         ;AND DX with 0001h to get 0001h back or 0000h
230          CMP DX, 00h           ; check if it's 0000h
231          JNE norm              ;if not equal to 0000h then jump to norm
232          ADD DX, 1             ;if equal to 0000h then add 1
233 norm:
234          NOT DX
235          INC DX               ;take the 2's complement to get -1
236          MOV ONE, DX          ;store -1 in variable ONE
237          SHR  AX, 01h       ;if odd enter one in the shift
238          OR   AX, 8000h
239          MOV  RANDOM1, AX
240
241 rend:    POP  DX
242          POP  CX
243          POP  BX      ;to registers the registers after the call
244          POP  AX
245          RET
246 RAND1    ENDP
```

Figure 1. Obtaining +1 or -1 in the RAND macro

Conversely, if number of parity bits is even, then just AND DX, 0001h, and ADD 1 if needed – to get +1.

After inserting the maximum amount of steps, we directly jump to **posNum** to print the value of STOREV1, which is initially zero – that is our initial X position (see line 67 in Figure 2).

Unfortunately, I could not pass zero to the second column (Y position), so that it starts either from +1 or -1.

My next step was to add this randomized +1 or -1 to BX, and then store to variable STOREV1 in order to save my current position to which I would add +1 or -1 in the next steps (see lines 71-72 in Figure 2). Then I compared STOREV1 with 0 and used conditional flags for signed arithmetic. If the value of the variable is positive or zero, then we directly print it on the screen, but if the value is negative, then we take its 2's complement and append '-' ASCII symbol before it (see lines 75-102 in Figure 2 & lines 105-109 in Figure 3).

```
61  next2:
62      newline
63      writemsg MSG6
64      CALL INPUT_NUM      ;input the number of entries
65      MOV CX, VARIN
66      XOR BX, BX
67      JMP posNum          ;jump to print the first value of the sequence as 0
68  cycle:
69      CALL RAND1
70      MOV BX, STOREV1     ;move stored value to BX
71      ADD  BX, ONE        ;add to BX randomized 1 or -1
72      MOV  STOREV1, BX    ;store this value in variable STOREV1
73      MOV  AX, RANDOM1
74      CMP STOREV1, 0000h ;check if the value is positive or negative or zero
75      JZ   posNum          ;if zero print to DOS in posNum
76      JG   posNum          ;if it is positive print to DOS in posNum
77
78  negNum:
79      newline
80      MOV AH,02h           ;however, if it is negative print the symbol '-' before the value
81      MOV DL, '-'
82      INT 21h
83
84  flow:
85      PUSH CX
86      PUSH DX
87
88
89      XOR CX, CX          ;by use of register CX, take the negative value of STOREV1
90      XOR DX, DX
91      MOV CX, STOREV1     ;and perform the 2's complement to get the positive number
92      NOT CX
93      INC CX
94
95      MOV DX, CX
96      MOV VAROUT, DX      ;print the negative number using '-' and 2's complement
97      CALL PRINT_NUM
98
99
100     POP DX
101     POP CX
102     JMP cycle2      ;jump directly to cycle2
```

Figure 2. Adding +1 or -1 and printing the results

```
105 posNum:
106     MOV DX, STOREV1
107     MOV  VAROUT, DX
108     newline
109     CALL PRINT_NUM
110     ;JMP posNum2
111
112 cycle2:
113     CALL RAND2
114     MOV AX, RANDOM2    ;take the randomised number from rand_seed 2 and store it in AX
115     MOV BX, STOREV2     ; move the variable STOREV2 corresponding to the 2nd column to BX
116     ADD  BX, ONE         ;add randomised +1 or -1
117     MOV  STOREV2, BX     ;store the value in STOREV2
118
119     CMP STOREV2, 0000h  ;make the same operations by checking if the number is 0, positive or negative
120     JZ   posNum2
121     JG   posNum2
```

Figure 3. Printing pos. numbers and going for *cycle2* to print values of the 2nd column

We perform exactly the same operations with values of the second columns but using different value of clock ticks form RAND_SEED2. By the end of cycle2, we return to cycle by applying some strange succession of jumps as shown in Figure 4.

```
151  done:
152      DEC CX
153      JNZ cond            ;LOOP couldn't reach the beginning of the cycle so we used
154
155
156      MOV  AH,4Ch      ;21h OS(DOS)interrupt.
157      INT  21h         ;AH=4Ch is for terminate the process
158
159  cond:
160      JMP NEAR PTR cycle    ; return to the beginning of cycle and repeat
161      RET               ;return to the OS
162  MAIN     ENDP         ;termination of the procedure
```

Figure 4. Unconditional and conditional jumps to return to the beginning of *cycle*

As we know the LOOP command can jump from -128 to 127 bytes relative to the address of the previous command, but it was not sufficient for my code. So I jumped to the unconditional jump in line 160 that returned me to the beginning of the cycle.

Results

I have successfully done the 2D random walk in Assembly which you can verify by assembling the code attached to the report. It adds either 1 or -1 at each consecutive steps and displays value in 2 columns (see Figure 5). However, I failed to transfer the data to .txt file properly as I couldn't convert two-digit decimals into ASCII codification. I called the macro WRITE_FILE inside the PRINT_NUM procedure that deals with conversion into ASCII, but still it showed me some irrelevant ASCII symbols (see Figure 6). So I haven't handled it and decided to transfer only the first column to the .txt file.

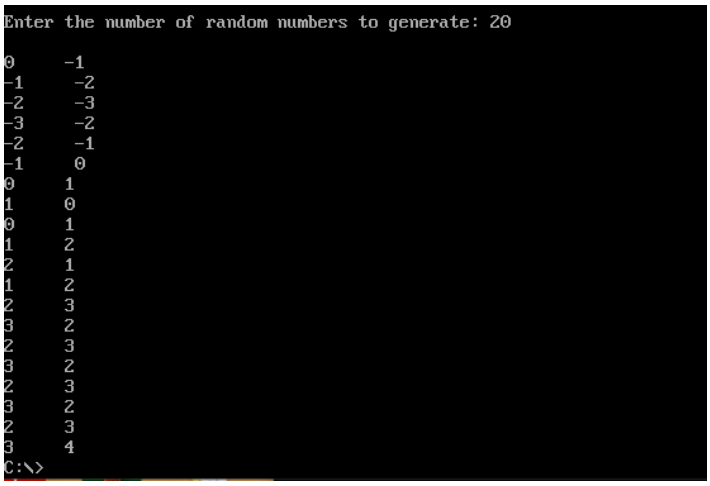

Figure 5. Random walk with 20 numbers

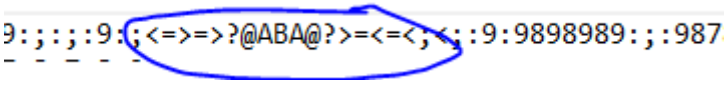

Figure 6. Problem with ASCII