



TouchMetal Sample



Sample Program 4 is a memory-test game.

Author : Dr Nicholas J. Macias

```
00001 ;
00002 ; memory-test game
00003 ; NJM May 2026
00004 ;
00005 ; Game displays a sequence of numbers, which you try to repeat.
00006 ; Initial sequence has a length of 1; the length increases each round.
00007 ;
00008 ; Press 000 and then Interrupt to begin. OUT register will display a
sequence of numbers, followed by 111.
00009 ; At the end, you repeat them with the numeric keys.
00010 ; While you are entering your numbers, display will count up where you are
in the sequence
00011 ; 377 means you correctly completed the sequence. Press 0 to go on to the
next sequence
00012 ; 000 means you made a mistake, game over. Press 000 and Interrupt to try
again.
00013 ;
00014 ;
00015 ; NOTE: There will be no repeat numbers in the sequence.
00016 ;
00017
00018 ; interrupt handler
00019                                     .org 376
00020 376:004 003                       jmp IntHandler
00021
00022                                     .org 0
00023 000:030                             inte      ; enable interrupts
00024 001:004 005                       jmp NewGame ; go to start of game
00025
00026 ; put interrupt code here
00027             IntHandler:
00028 003:101                     inc R1 ; set flag
00029 004:032                     rti ; and return
00030
00031 ; play a game!
00032             NewGame:
00033 005:127                     clr R7 ; length of sequence
00034 006:071 045                 ldi R1,prevRandNum ; address of previous
random number
00035 010:321                     clr (R1) ; clear it (so we don't gen
a 0 first time)
00036 011:071 151                 ldi R1,lastIR ; last input register
00037 013:321                     clr (R1) ; user shiould start with
000
```



TouchMetal Sample



```
00038
00039          GetInt:                ; wait for INT key
00040 014:121      clr  R1              ; Flag saying we're waiting
00041 015:070 177      ldi  R0,var_c    ; address of var_c (starting seed)
00042          WaitStart:
00043 017:300      inc  (R0)              ; increment starting seed
00044 020:027 011      mov  R1,R1        ; set flag
00045 022:005 017      jz   waitStart    ; still waiting
00046
00047 ; User pressed INT; go on to first (or next) round
00048          NextRound:
00049 024:107      inc  R7                ; new sequence length
00050 025:003 046      call playRound      ; run a sequence of length R7
00051 027:170      out  R0                ; display status (0=fail,
377=success)
00052 030:027 000      mov  R0,R0        ; test status
00053 032:005 005      jz   NewGame        ; game over - try again!
00054 ; success here; have user press 0 to start display of next sequence
00055 034:003 040      call get0            ; wait for 0 key
00056 036:004 024      jmp  NextRound      ; try next sequence...
00057
00058          get0:
00059 040:003 152      call getAKey
00060 042:006 040      jnz  get0            ; getAKey touches Z
00061 044:001      ret                    ; Got a 0!
00062
00063 ; play one round (sequence)
00064 ; R7 is sequence length
00065 045:000      prevRandNum: .byte  0    ; remember prev # so we don't
00066                                           ; require input like say 1111
00067          playRound:
00068 ; First, get a new number and add to end of sequence
00069 046:003 201      call Rand              ; R1 is new part of sequence
00070 050:070 007      ldi  R0,7            ; bitmask
00071 052:024 001      and  R0,R1            ; just keep last octal digit
00072 ;*** bump if same as prior number
00073 054:070 045      ldi  R0,prevRandNum
00074 056:027 202      mov  (R0),R2        ; R2 is prev number
00075 060:021 012      sub  R1,R2          ; dup?
00076 062:006 070      jnz  NoDup          ; nope, okay
00077 ; else need to tweak R1
00078 064:072 001      ldi  R2,1            ; toggle mask
00079 066:026 021      xor  R2,R1          ; toggle lsb
00080          NoDup:
00081                                           ; save R1 for next time
00082 070:070 045      ldi  R0,prevRandNum
00083 072:027 110      mov  R1,(R0)        ; current num is saved :)
00084
00085                                           ; save this number at end of the
```



TouchMetal Sample



sequence

```
00086 074:072 240      ldi R2,Sequence      ; address where sequence begins
00087 076:112          dec R2                ; since we start with an offset of 1
:)
00088 077:020 072      add R7,R2            ; R0 is where new number goes
00089 101:027 112      mov R1,(R2)          ; save it!
00090
00091 ; now display sequence, 1 number at a time, 1 sec between each
00092 103:072 240      ldi R2,Sequence      ; number to show
00093 105:027 073      mov R7,R3            ; count down to 0
00094                showSequence:
00095 107:372          out (R2)                ; show next number
00096 110:003 171      call Wait1            ; wait 1 sec
00097 112:102          inc R2                ; move to next number
00098 113:113          dec R3                ; and count down
00099 114:006 107      jnz showSequence      ; show next number
00100
00101 ; sequence is done
00102 116:070 111      ldi R0,111            ; "your turn" flag
00103 120:170          out R0
00104 121:003 171      call Wait1
00105
00106 ; here's where user tries to repeat the sequence
00107 123:072 240      ldi R2,Sequence      ; number to show
00108 125:027 073      mov R7,R3            ; count down to 0
00109 127:125          clr R5                ; this will count where we are in the
sequence
00110
00111                userTurn:
00112 130:105          inc R5
00113 131:175          out R5                ; show where we are in sequence
00114 132:003 152      call getKey           ; get a key (into R1)
00115 134:021 220      sub (R2),R0          ; Z means good answer
00116 136:006 147      jnz fail             ; didn't match :(
00117 ; so far so good...
00118 ;%% show progress...
00119 140:102          inc R2                ; next sequence loc
00120 141:113          dec R3                ; count down
00121 142:006 130      jnz userTurn          ; keep going...
00122 ; DONE!
00123 144:070 377      ldi R0,377            ; success flag
00124 146:001          ret                  ; sequence done
00125
00126 ; user didn't match sequence
00127                fail:
00128 147:120          clr R0                ; failure flag
00129 150:001          ret                  ; sequence done
00130
00131
```



TouchMetal Sample



```
00132 ;***
00133 ;*** Support Functions
00134 ;***
00135
00136 ;
00137 ; getAKey
00138 ; Wait for user to press a key, then return it in R0
00139 ; NOTE: Limited in its ability to detect consecutive
00140 ; presses of the same key...
00141 ;
00142 ; input: none
00143 ; output: R0
00144 ; uses: R0, R1
00145 ;
00146 ; NJM May 2026
00147 ;
00148
00149 151:000      lastIR: .byte 0          ; remember last input reg here
00150      getKey:
00151 152:070 151      ldi R0,lastIR        ; address of last IR
00152      getKey_W1:          ; spin here
00153 154:161          in R1                ; Get current input reg
00154 155:021 201      sub (R0),R1        ; changed?
00155 157:005 154      jz getKey_W1      ; nope
00156          ; input register has changed!
00157 161:161          in R1                ; read it again
00158 162:027 110      mov R1,(R0)        ; save for next time
00159 164:070 007      ldi R0,7          ; mask for low digit
00160 166:024 010      and R1,R0         ; R0=latest digit
00161 170:001          ret
00162
00163 ;
00164 ; Delay(n)
00165 ; delays for n seconds
00166 ; (approx 7% fast on my board...)
00167 ;
00168 ; input: R0
00169 ; output: none
00170 ; changes: R0
00171 ;
00172 ; NJM May 2026
00173 ;
00174
00175 ;Delay:
00176 ;      call Wait1
00177 ;      dec R0
00178 ;      jnz Delay
00179 ;      ret
00180
```



TouchMetal Sample



```
00181          Wait1:          ; 1-second delay
00182 171:000          nop
00183 172:000          nop
00184 173:000          nop
00185 174:000          nop
00186 175:000          nop
00187 176:001          ret
00188
00189 ;
00190 ; rand(r)
00191 ; perturbs r and returns it
00192 ;
00193 ; input: none
00194 ; output: R1
00195 ; uses: R0, R1, R2, R3
00196 ; NJM May 2026
00197 ;
00198 ; "var_c" is an incrementing additional element
00199 ;
00200 177:000          var_c:      .byte  0      ; salt
00201 200:000          lastrandnum: .byte  0      ; save last random number here
00202
00203          Rand:
00204 201:071 200          ldi  R1,lastrandnum    ; address of last random num
00205 203:027 211          mov  (R1),R1          ; R1 is the last random num
00206 205:070 177          ldi  R0,var_c         ; c increments each time we call this
00207 207:300              inc  (R0)            ; do the inc
00208 210:072 137          ldi  R2,0137         ; first mask
00209 212:026 012          xor  R1,R2          ; twiddle those bits
00210 214:142              rol  R2            ; shift left one bit
00211 215:073 376          ldi  R3,0376        ; mask to clear lsb
00212 217:024 032          and  R3,R2          ; clear lsb
00213 221:073 022          ldi  R3,022         ; 2nd mask
00214 223:026 013          xor  R1,R3          ; twiddle that
00215 225:020 023          add  R2,R3          ; sum these
00216 227:020 203          add  (R0),R3        ; and add var_c
00217 231:070 200          ldi  R0,lastrandnum
00218 233:027 130          mov  R3,(R0)         ; save this for next time
00219 235:027 031          mov  R3,R1          ; return value
00220 237:001          ret
00221
00222
00223 ;***
00224 ;*** Sequence Storage
00225 ;***
00226 240:000          Sequence: .byte  0      ; sequence grows from here (without
00227                          bounds!)
00228                          .end
```