```
;
; Kill the bit for PLC14500-Nano.
;
; NOTE! This game requires a REV.C/D (or a modified REV.B). The game is playable
;   on the standard clock (R30=1K, 4.8KHz) as opposed to the version for REV.B
;   that required the clock to be dropped to make the game playable.
;   This is one example of the power of the IEN/OEN instructions that allow us
;   to execute conditionally the bit shift block only every 4 rounds of the
;   program, thus introducing the delay.
;
; The game idea itself is from a game for the Altair by Dean McDaniel in 1975.
; Original Description:
;   Object: Kill the rotating bit. If you miss the lit bit, another
;   bit turns on leaving two bits to destroy. Quickly
;   toggle the switch, don't leave the switch in the up
;   position. Before starting, make sure all the switches
;   are in the down position.
;
; The idea to implement it on the MC14500 came to me after watching this awesome
;   build by Usagi Electric: https://www.youtube.com/watch?v=md_cPxVDqeM
;   Also the code was heavily adapted from this one:
;   https://github.com/veremenko-y/mc14500-programs/blob/main/sbc1/killthebit.s
;

.board=PLC14500-Nano

.io_GAME_BIT0=SPR0  ; Game bit 0
.io_GAME_BIT1=SPR1  ; Game bit 1
.io_GAME_BIT2=SPR2  ; Game bit 2
.io_GAME_BIT3=SPR3  ; Game bit 3
.io_GAME_DLY0=SPR4  ; Game delay bits used as a 2-bit counter
.io_GAME_DLY1=SPR5  ;   to shift the bit only once every 4 rounds.
.io_SWAP=SPR6       ; Swap bit used for temporary storage
.io_GAME_LED0=OUT0  ; Game LED 0
.io_GAME_LED1=OUT1  ; Game LED 1
.io_GAME_LED2=OUT2  ; Game LED 2
.io_GAME_LED3=OUT3  ; Game LED 3
.io_GAME_LED4=OUT4  ; Game LED 4
.io_GAME_BUTTON=IN0 ; Game only button

STO     SWAP        ; Save RR
ORC     RR          ; RR=RR|!RR (always 1)
IEN     RR          ; Enable inputs
OEN     RR          ; Enable outputs
LD      SWAP        ; Restore RR

; This STO is executed only once because after the first loop we
;   set RR=1 (see last line of the whole program), so when we come here
;   RR is 1 and LDC will load a 0.
LDC     RR          ; This is 1 on reset (RR is initialised to 0)
SKZ
STO     GAME_BIT0   ; initialise memory with 1 initial bit

; Kill (or set!) the first bit.
; The bit will be killed if the button is pressed while it's high.
LD      GAME_BUTTON
XNOR    GAME_BIT0
STOC    GAME_BIT0
```

```
; Rotate all bits forward (and last back to first).
;   But only every 4 rounds to slow down the action.
LD      GAME_DLY0   ; Negate GAME_DLY0
STOC    GAME_DLY0
OEN     RR          ; If GAME_DLY0 transitioned to 0
LD      GAME_DLY1   ; negate GAME_DLY1
STOC    GAME_DLY1
AND     GAME_DLY0
OEN     RR          ; endif if GAME_DLY1 and GAME_DLY0 are 1

LD      GAME_BIT3
STO     SWAP
LD      GAME_BIT2
STO     GAME_BIT3
LD      GAME_BIT1
STO     GAME_BIT2
LD      GAME_BIT0
STO     GAME_BIT1
LD      SWAP
STO     GAME_BIT0

ORC     RR          ; RR=RR|!RR (always 1)
OEN     RR          ; endif always

; Display the game status by showing on the outputs the values stored in SPR.
; Note: we don't play directly on SPR as that would be confusing as some bits
;   in SPR are used to store temporary values.
LD      GAME_BIT0
STO     GAME_LED0
LD      GAME_BIT1
STO     GAME_LED1
LD      GAME_BIT2
STO     GAME_LED2
LD      GAME_BIT3
STO     GAME_LED3

ORC     RR         ; RR=RR|!RR (always 1)
                   ; This will cause the code that initialises the game bit
                   ;   to be skipped.

; Don't JMP 0 here, the rest of the code will be
;   filled with NOPF and act as a delay, otherwise the
;   game will be too fast.
```