Experiment 4

Bit manipulation, Programming using I/O Ports and Counter programs

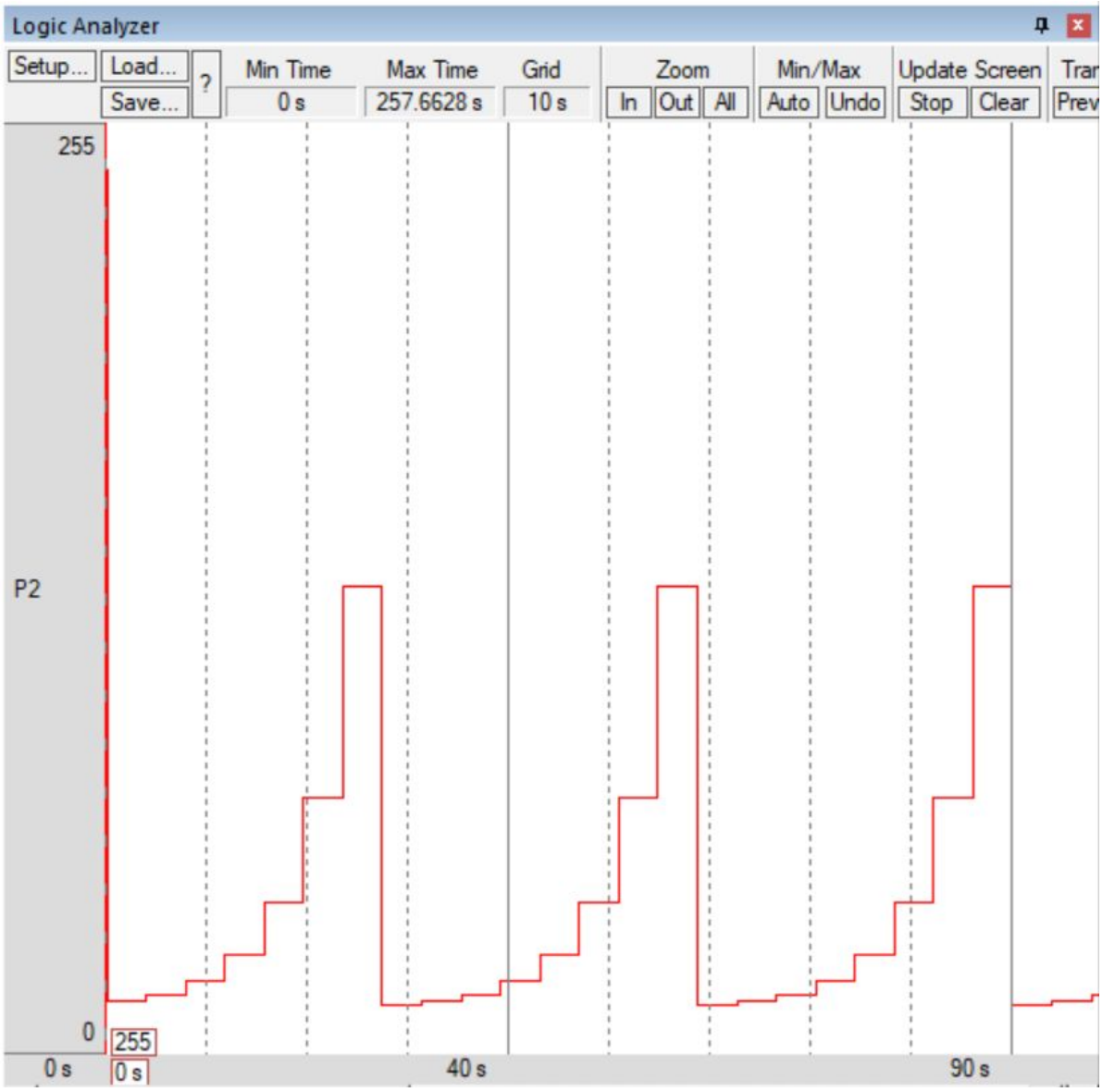
Exercise 1

> Implement an 8 bit ring counter.

Code

```
ORG 0000H               ; ORIGINATE
AJMP START              ; JUMP TO THE LABEL START

START:
    MOV A, #01H         ; INITIALISE

ITER:
    RL A                ; LEFT SHIFT ACCUMULATOR, WITHOUT CARRY
    MOV P2, A           ; SHOW IN PORT-2

    LCALL DELAY         ; AWAIT A DELAY

    SJMP ITER           ; REPEAT IN A LOOP

DELAY:
    ; AWAIT (0FH*0FFH*0FFH*02H) MACHINE CYCLES
    MOV R3, #0FH
    UP1:
        MOV R4, #0FFH
        UP2:
            MOV R5, #0FFH
            UP3:
                NOP
                NOP
            DJNZ R5, UP3
        DJNZ R4, UP2
    DJNZ R3, UP1

    RET                 ; RETURN BACK TO THE MAIN LOOP
    END
```

Output



Exercise 2

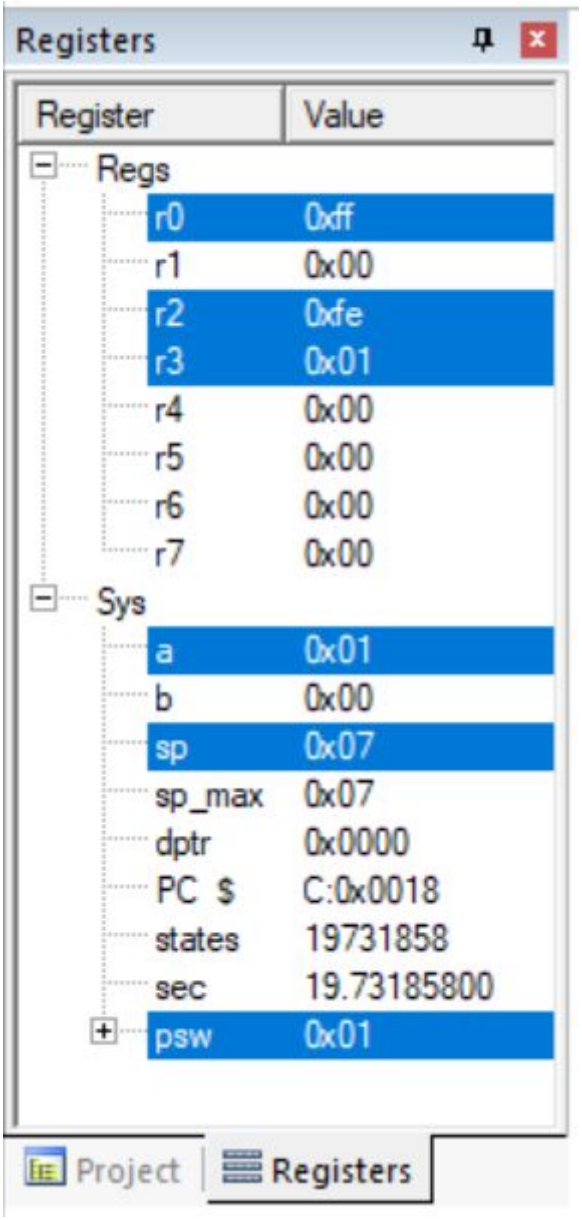
> Assuming R1:R0 registers as an 16 bit register rotate the content of this register left such that bit-7 of R0 becomes bit 0 of R1 and bit-7 of R1 becomes bit-0 of R0.

Code

```
ORG 0000H               ; ORIGINATE
AJMP START              ; JUMP TO THE LABEL START

START:
    MOV R0, #0FFH       ; INPUT LOWER BYTE
    MOV R1, #00H        ; INPUT HIGHER BYTE
    CLR C               ; CLEAR CARRY

    MOV A, R1           ; GET HIGHER BYTE IN ACCUMULATOR

    ; MASK AND GET THE FIRST BIT OF THE HIGHER BYTE, AND STORE IN ACCUMULATOR
    ANL A, #80H         ; (80H) IS (10000000B)

    ; IF ACCUMULATOR IS ZERO => BIT-7 OF HIGHER BYTE IS 0
    JZ UP1
```

```
    ; IF ACCUMULATOR IS NON-ZERO => BIT-7 OF HIGHER BYTE IS 1
    SETB C              ; SET THE CARRY BIT

UP2:
    MOV A, R0           ; GET LOWER BYTE IN ACCUMULATOR
    RLC A               ; ROTATE LEFT ACCUMULATOR, WHICH HAS THE LOWER BYTE
    MOV R2, A           ; STORE LOWER BYTE IN (R2)

    ; THE CARRY BIT IS NOW THE BIT-7 OF THE LOWER BYTE

    MOV A, R1           ; GET HIGHER BYTE IN THE ACCUMULATOR AGAIN
    RLC A               ; ROTATE LEFT ACCUMULATOR, WHICH HAS THE HIGHER BYTE
    MOV R3, A           ; STORE HIGHER BYTE IN (R3)

    SJMP HERE           ; END

UP1:
    CLR C               ; CLEAR CARRY
    SJMP UP2            ; RETURN TO THE FLOW

HERE:
    SJMP HERE           ; LOGICAL END
    END
```

Input & Output



Exercise 3

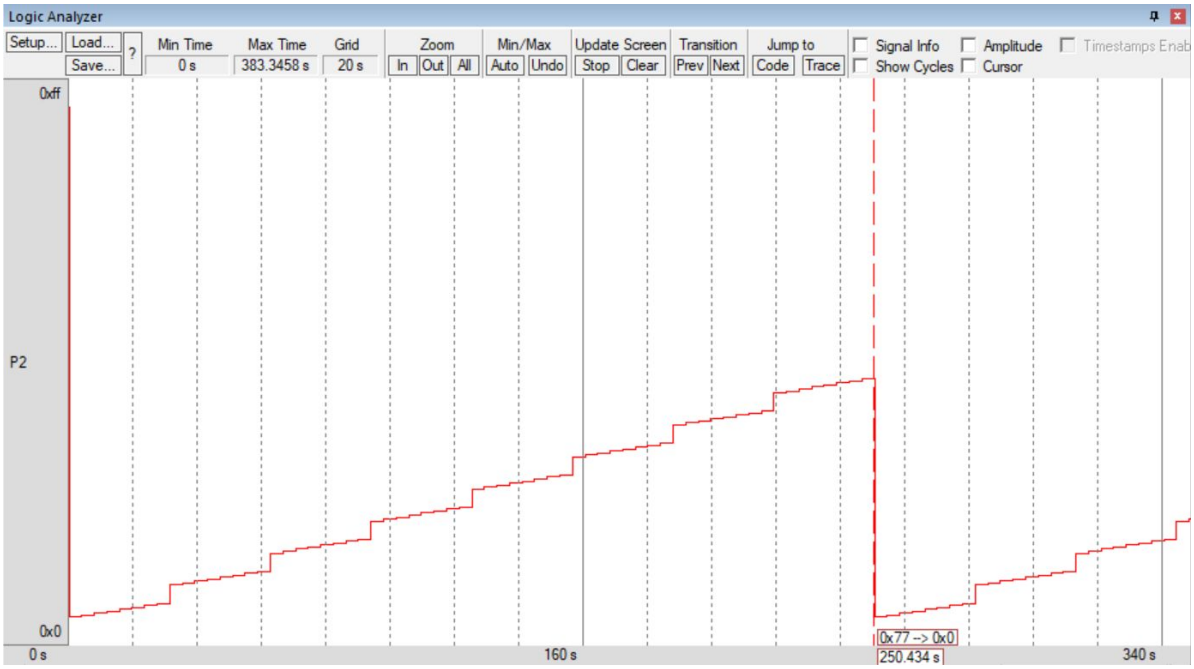
> Implement an 8-bit octal up counter.

Code

```
ORG 0000H                       ; ORIGINATE
AJMP START                      ; JUMP TO THE LABEL START

START:
    MOV R0, #00H                ; INITIALISE COUNTER

    MOV R2, #08H                ; UPPER NIBBLE; (OCTAL BASE - 1)
    U_ITER:
        MOV R1, #08H            ; LOWER NIBBLE; (OCTAL BASE - 1)
        L_ITER:
            MOV A, R0           ; LOAD THE COUNTER
            MOV P2, A           ; SHOW IN PORT-2
            LCALL DELAY         ; AWAIT

            INC A               ; INCREMENT THE ACCUMULATOR
            MOV R0, A           ; SAVE THE COUNTER
```

```
            DJNZ R1, L_ITER      ; REPEAT FOR LOWER NIBBLE

        ADD A, #08H              ; ADD TO GO TO THE NEXT VALID OCTAL NUMBER
        MOV R0, A                ; SAVE THE COUNTER

        DJNZ R2, U_ITER          ; REPEAT FOR UPPER NIBBLE
    SJMP START                   ; REPEAT IN A LOOP

DELAY:
    ; AWAIT (0FH*0FFH*0FFH*02H) MACHINE CYCLES
    MOV R3, #0FH
    UP1:
        MOV R4, #0FFH
        UP2:
            MOV R5, #0FFH
            UP3:
                NOP
                NOP
            DJNZ R5, UP3
        DJNZ R4, UP2
    DJNZ R3, UP1

    RET                          ; RETURN BACK TO THE MAIN LOOP
    END
```

Output



Exercise 4

> On Port-0 of the MC, generate a sawtooth wave.

Code

```
ORG 0000H                ; ORIGINATE
AJMP START               ; JUMP TO THE LABEL START

START:
    MOV R0, #00H         ; INITIALISE

ITER:
    MOV A, R0            ; GET CURRENT DATA
    INC A                ; INCREMENT THE ACCUMULATOR

    MOV P0, A            ; SHOW IN PORT-0
    MOV R0, A            ; SAVE IN (R0)

    LCALL DELAY          ; AWAIT (0FFH) MACHINE CYCLES
    SJMP ITER            ; REPEAT IN A LOOP

DELAY:
    ; AWAIT (0FFH) MACHINE CYCLES
    MOV R1, #0FFH
    UP1:
        NOP
    DJNZ R1, UP1

    RET                  ; RETURN BACK TO THE MAIN LOOP
    END
```

Output