Program belonging to logic-circuits. Part-09

Test EQU 1FF0H

Blank EQU 0802H

Delay EQU 0002H

Key\_Tab EQU 077BH

Seg\_Tab EQU 07F0H

Message EQU 080DH

Stack\_pointer EQU 000FH

Dis\_Buf EQU 1100H

Wait EQU 0100H

Scan EQU 0200H

Scan-1x EQU 0300H

Exec EQU 0400H

; PROC Main

; Clean displays. All corresponding segments of the displays are controlled parallel. With control lines C0-C5 (from Port-2), will be decided to which digits the information of Port-1, will go.

0000 OUTI P1, 0000 ; all segments off

0001 OUTI P2, 003F ; enable all displays

0002 OUTI P2, 0000 ; disables all displays

; Display initial message. This message is stored in memory-locations 080D-0812H. (6 WORDS). The start-address of this message is loaded into memory. Register-10 is used as an offset.

When R10=0005, the rightmost two digits will be updated with the content of memory location 0812H. The right control line is activated by the content of R11 (0001). Hereafter R10 will be decremented, the next two digits will be updated with the content of location 0811H, R11 is shifted to the left. (0002) and that’s why the content of 0811H goes to the right digits.

This will be repeated until all digits are updated.

0003 IDXA 080DH ; start address of memory location

0004 LDI R11, 0001 ; this bit enables the right control line

0005 LDI R10, 0005 ; loads the right content of the table

0006 start: OTRX P1, R10 ; puts the content of a memory-location in Port-1

0007 OUT P2, R11 ; enables the right two digits

0008 OUTI P2, 0000 ; disables the control line (information is saved)

0009 SHL R11, R11 ; enable next two digits

000A DEC R10, R10 ; selects next characters

000B JR P, start ; repeat until all characters are done

; Initialise Stackpointer

000C LDI Stack\_pointer

;show message some time

000D Call Wait ;display message some time

; initialize display-buffer.

; Fill memory-location 1100H-1105H with zero. When this area is pointed to as output,

all displays are blanked

000E IDXA 1100 ; start-address of buffer

000F LDI R0, 0000 ; content of array

0010 LDI R10, 0005 ; offset to start-address

0011 Store: ORI R11, R10, 1000 ; ALAS, you still have to give up the page-number. (=1)

0012 STRX R0, R11 ; fill last element of array

0013 DEC R10, R10 ; go to next element (to the left)

0014 JR P, Store ; all 6 elements done? 5-0

0015 Main: CALL Scan

0016 CALL Exec

0017 JR Main

; END PROC

; PROC Wait

0100 LDI R1, 03FF ; Load register with delay-time

0101 dec: DEC R1, R1

0102 JR NZ, dec

0103 RET

; END PROC

; PROC Scan

; At first, if there is any key-stroke pending, the buffer will be emptied. The input for the keystrokes (Port-0) must be empty for at least two scans. Then the PROC “Scan\_1x” is called, while no key is hit.

(In reality, this procedure is done, to prevent wrong interpretations, as a result of key-bouncing)

0200 Scpre: LDI R5, Delay ; number of scans, in which no keystroke may occur

0201 Scanx: CALL Scan\_1 ; scan the keyboard

0202 JR C, Scpre ; if there is any keystroke, start all over again

0203 DEC R5, R5 ; count the number of scans

0204 JR NZ, Scanx ; total number of scans done?

0205 Scloop: CALL Scan\_1 ; scan the keyboard,

0206 JR NC, Scloop ; no key: again. Key: go back to calling PROC.

0207 RET

; END PROC

; PROC Scan\_1

This procedure scans the keyboard one time. At the same it controls the output.

0300 SCF ; reset carry (set carry first, then complement it)

0301 CCF

0302 IDXA 1100 ; pointer to start of Display-Buffer

0303 LDI R10, 0000 ; position number of key pressed

0304 LDI R9, 0005 ; number of element in buffer (column)

0305 LDI R11, 0001 ; enable word for control lines

0306 col: ORI R12, R9, 1000 ; together with R9, element number + page-number

0307 OTRX P1, R12 ; information to digits

0308 OUT P2, R11 ; enable the right digit

0309 LDI R8, 0006 ; initialize row-counter

030A IN R6, P0 ; read byte (n)

030B OUTI P2, 0000 ; disable control lines (preventing flicker)

030C test: OR R6, R6, R6 ; test, if any (remaining) bit is high

030D JR NZ, shift ;

030E ADD R10, R10, R8 ; not any bit high, add position with row-counter

030F JR ready ; read next byte

0310 shift SHR R6, R6 ; check every bit

0311 JR NC, next ; when a bit is high (keystroke), next bit, else

0312 LD R0, R10 ; save key-position number in R0

0313 LDI R1, 0001 ; save the occurrence of a key-stroke

0314 next: INC R10, R10 ; next key-position

0315 DEC R8, R8 ; next bit

0316 JR NZ, test ; repeat if not all bits are done (4-6 bits)

0317 ready: SHL R11, R11 ; enable next digits and input-byte

0318 DEC R9, R9 ; load next element of display-buffer

0319 JR P, COL ; repeat if not all 6 bytes are done. (5-0)

031A OR R1, R1, R1 ; test if there was any key-stroke

031B JR Z , end ; no, leave procedure, else

031C SCF ; set carry Flag

031D LDI R1, 0000 ; reset flag for key-strokes

031E end: RET

END PROC

; PROC Exec

; This procedure first uses the key\_map. The key-position is used as an index in the table. Then the value in that memory-location is used, for looking up the 7-segment codes. This codes will be sent to

The displays

0400 decode IDXA 077B ; start-address of key\_map

0401 LDRX R7, R0 ; value of element, index in R0 = keyposition, goes to R7

0402 IDXA 07F0 ; start-address of table with 7-segment-codes

0403 LDRX R0, R7 ; the 7-seg-code, belonging to the key-stroke goes to R0

0404 STA R0, 1105 ; This code is saved in the Display\_Buffer, always at the right

# 0405 RET ; go back to calling procedure

; END PROC