106061151 劉安得 HW2

*1. (Software design) (20%)*

*In some applications, BCD (binary coded decimal) is used instead of binary. Each BCD will use 4 bits to represent. A two digits BCD is usually placed in a single byte. For example, a two digits decimal number 59 would be represented by the byte “0101 1001”, where the upper nibble (b7~b4) 0101 stands for “5” and lower nibble (b3~b0) 1001 stands for “9”.*

*Please write an 8051 assembly program that will add two 6-digit BCD numbers stored in internal data memory location say 30h-32h and 33h-35h and produce a 7 digits BCD result in 36h-39h. You should show at least two results of the BCD addition operation in MCU8051IDE (by print screen).*

MOV A, 30H

ADD A, 33H

DA A

MOV 36H, A

MOV A, 31H

ADDC A, 34H

DA A

MOV 37H, A

MOV A, 32H

ADDC A, 35H

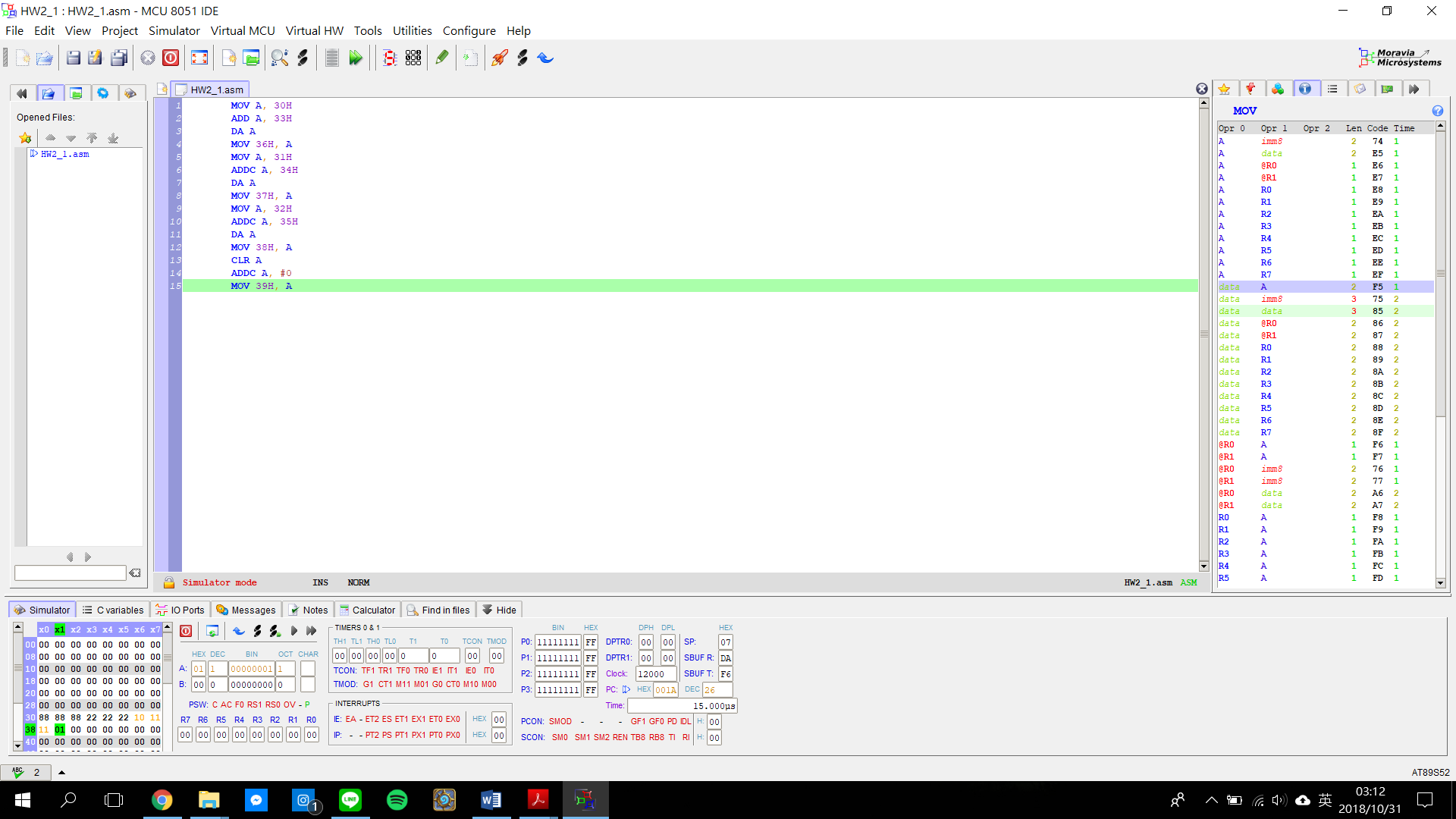
DA A

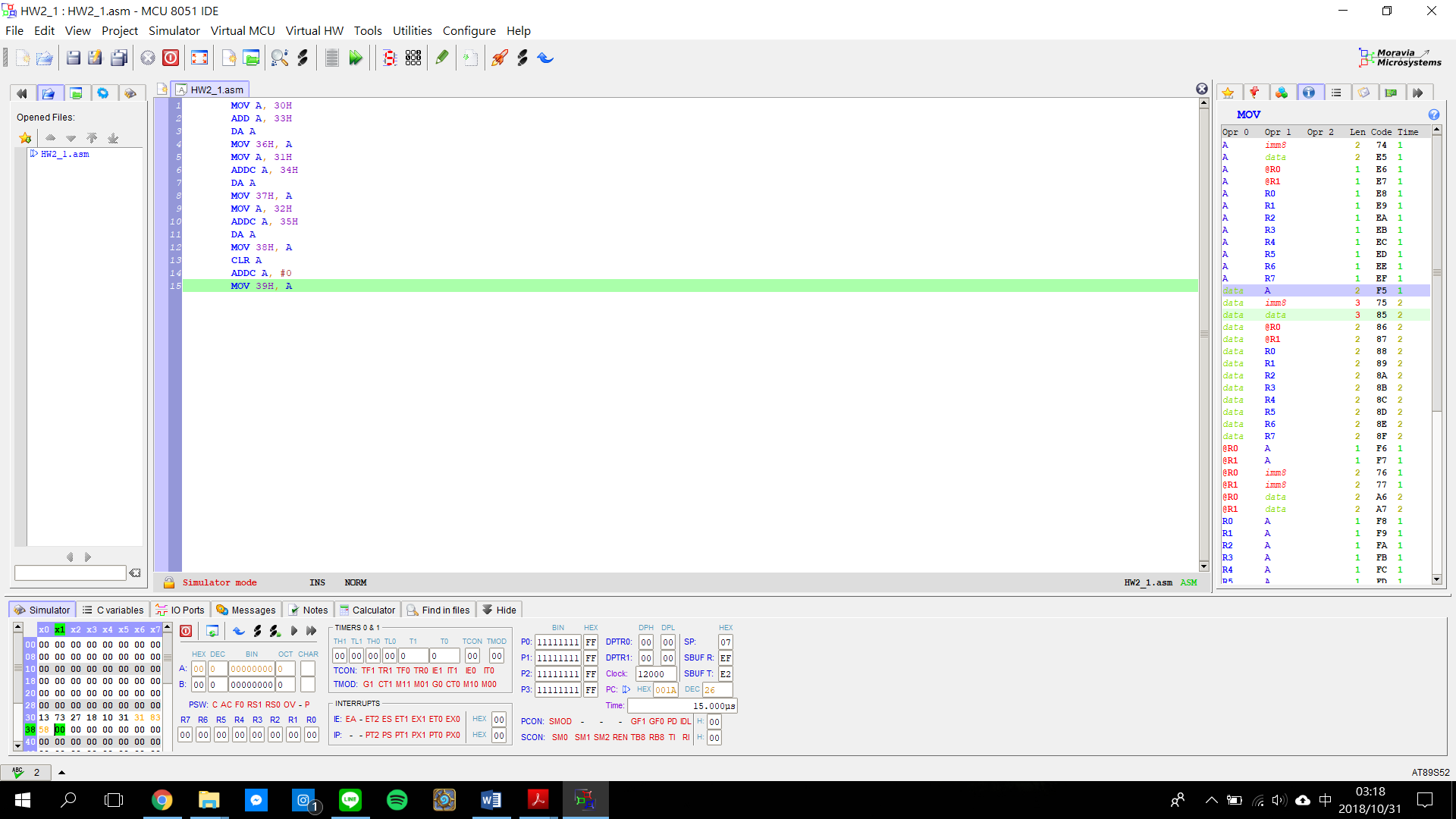
MOV 38H, A

CLR A

ADDC A, #0

MOV 39H, A





*2. (Software design, simulation and instruction cycle counting) (20%)*

*8051 is commonly used as the core of a programmable logic controller (PLC) to perform logical manipulations on various incoming signals and issue proper controls accordingly, because it has Boolean instructions, internal bit-addressable bits for storing Boolean variables, and 4 8-bit I/O ports. The basic idea to implement a logic operation with 8051 is to enclose the logic operations in an infinite loop so that it can react to change of input persistently like hardware does. Assume that we want to implement the following logic operations:*

*P1.0 =(P1.1•P1.2+!P1.3) ⊕ (!P1.4•P1.5+P1.3) if P1.6 = 0 and*

*(P1.1+!P1.2•P1.3) ⊕ (!P1.4+!P1.5•P1.3) if P1.6 = 1.*

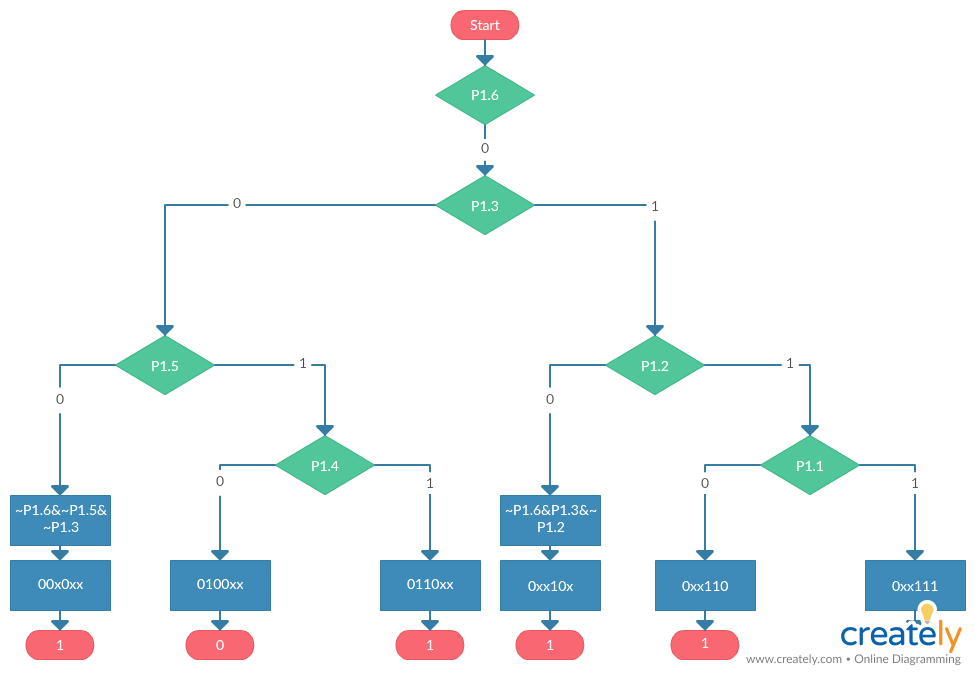
*(Note: ! means NOT, ⊕ means exclusive or)*

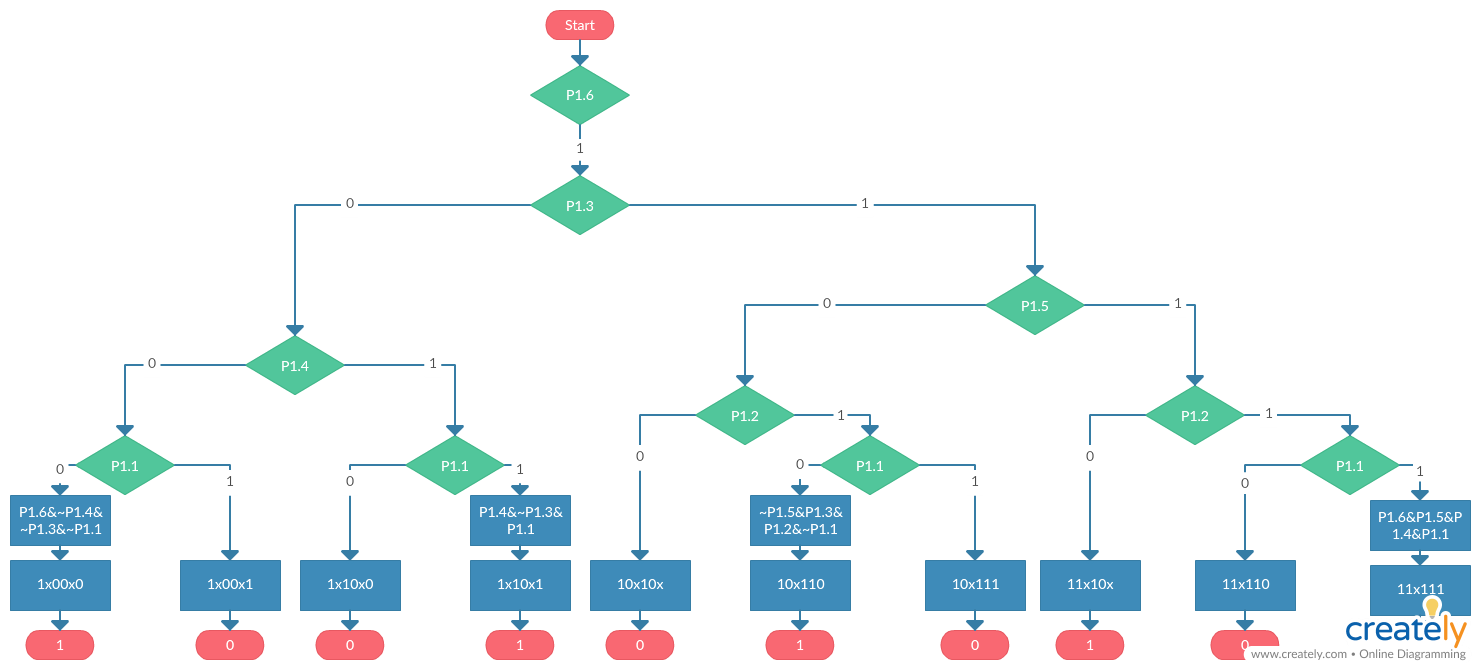
1. *First construct the truth-table. Apply K-map to simplify the logic if you like.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P1.6 | P1.5 | P1.4 | P1.3 | P1.2 | P1.1 | P1.0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

P1.0 = ~P1.6&P1.3&~P1.2 + ~P1.6&~P1.5&~P1.3 + ~P1.6&P1.4&~P1.1 + P1.4&~P1.3&P1.1 + ~P1.5&P1.3&P1.2&~P1.1 + P1.6&~P1.4&~P1.3&~P1.1 + P1.6&P1.5&P1.4&P1.1 + P1.5&P1.4&P1.3&~P1.2 + ~P1.4&P1.3&P1.2&~P1.1

1. *Then draw a flow chart to describe your code for the logic operation.*





1. *Write an 8051 assembly program to implement the above logic operations. You should* ***verify your program in MCU8051IDE*** *by trying several input combinations according to the truth-table.*

start: JB P1.6, b1xxxxx

JB P1.3, b0xx1xx

JB P1.5, b01x0xx

SETB P1.0 ;00x0xx

SJMP start

b01x0xx: JB P1.4, b0110xx

CLR P1.0 ;0100xx

SJMP start

b0110xx: SETB P1.0 ;0110xx

SJMP start

b0xx1xx: JB P1.2, b0xx11x

SETB P1.0 ;0xx10x

SJMP start

b0xx11x: JB P1.1, b0xx111

SETB P1.0 ;0xx110

SJMP start

b0xx111: CLR P1.0 ;0xx111

SJMP start

b1xxxxx: JB P1.3, b1xx1xx

JB P1.4, b1x10xx

JB P1.1, b1x00x1

SETB P1.0 ;1x00x0

SJMP start

b1x00x1: CLR P1.0 ;1x00x1

SJMP start

b1x10xx: JB P1.1, b1x10x1

CLR P1.0 ;1x10x0

SJMP start

b1x10x1: SETB P1.0 ;1x10x1

SJMP start

b1xx1xx: JB P1.5, b11x1xx

JB P1.2, b10x11x

CLR P1.0 ;10x10x

SJMP start

b10x11x: JB P1.1, b10x111

SETB P1.0 ;10x110

SJMP start

b10x111: CLR P1.0 ;10x111

SJMP start

b11x1xx: JB P1.2, b11x11x

SETB P1.0 ;11x10x

SJMP start

b11x11x: JB P1.1, b11x111

CLR P1.0 ;11x110

SJMP start

b11x111: SETB P1.0 ;11x111

SJMP start

1. *Estimate the best case and worst case propagation delay of your program from an input transition to output transition under 12 MHz operation:*

the best case: 9us(00x0xx, 0xx10x)

worst case: 13us(1xx11x)

*3. (Data sheet study and software design/simulation in MCU8051IDE) (20%)*

*A 4-bit DIP switch and a* ***common-anode*** *7-segment LED are connected to an 8051 as shown below. The 4-bit DIP switch is connected to P3.4-P3.7. When the DIP is in up position, the corresponding port pin would read in a “1”, and a “0” for down position. To turn on an LED segment, the corresponding control signal input of the 7-segment, i.e., a~g and dp, should be logic zero. Which means the corresponding port pin should be at logic “1” since a 74LS240 is used in the middle.*

1. *What is common-anode 7-segment LED? Why do we need the 220 resistor and the 74LS240? What’s the function and purpose of 74LS240 in this design? You should search the web to find out relevant data sheets of 74LS240 and common-anode 7-segment LED, read them, and then try to answer the question.*

common-anode 7-segment LED : consist 7 segment and a decimal point. All anode connections of LED are joined together to logic “1”. The segments are illuminated by applying a ground. 0 represent “on”, 1 represent “off”

220 resistor: Current limiting

74LS240: Octal buffer and line driver

|  |  |  |
| --- | --- | --- |
| Input | | Output |
| 1G, 2G | D |
| L | L | H |
| L | H | L |
| H | X | Z |

1. *Design an 8051 program which can continuously read the DIP position and display the 7-segment LED correspondingly. Test your program* ***in MCU8051IDE*** *and use the* ***simple keypad*** *and* ***LED display*** *virtual HW to demonstrate your program. You should capture at least two examples of the MCU8051IDE simulation results by print screen and put them in your written report.*

dp

P1.0

P1.1

P1.2

P1.3

P1.4

P1.5

P1.6

P1.7

P3.4

P3.5

P3.6

P3.7

74LS240

8051

a

a

b

b

c

c

d

d

e

e

f

f

g

g

dp

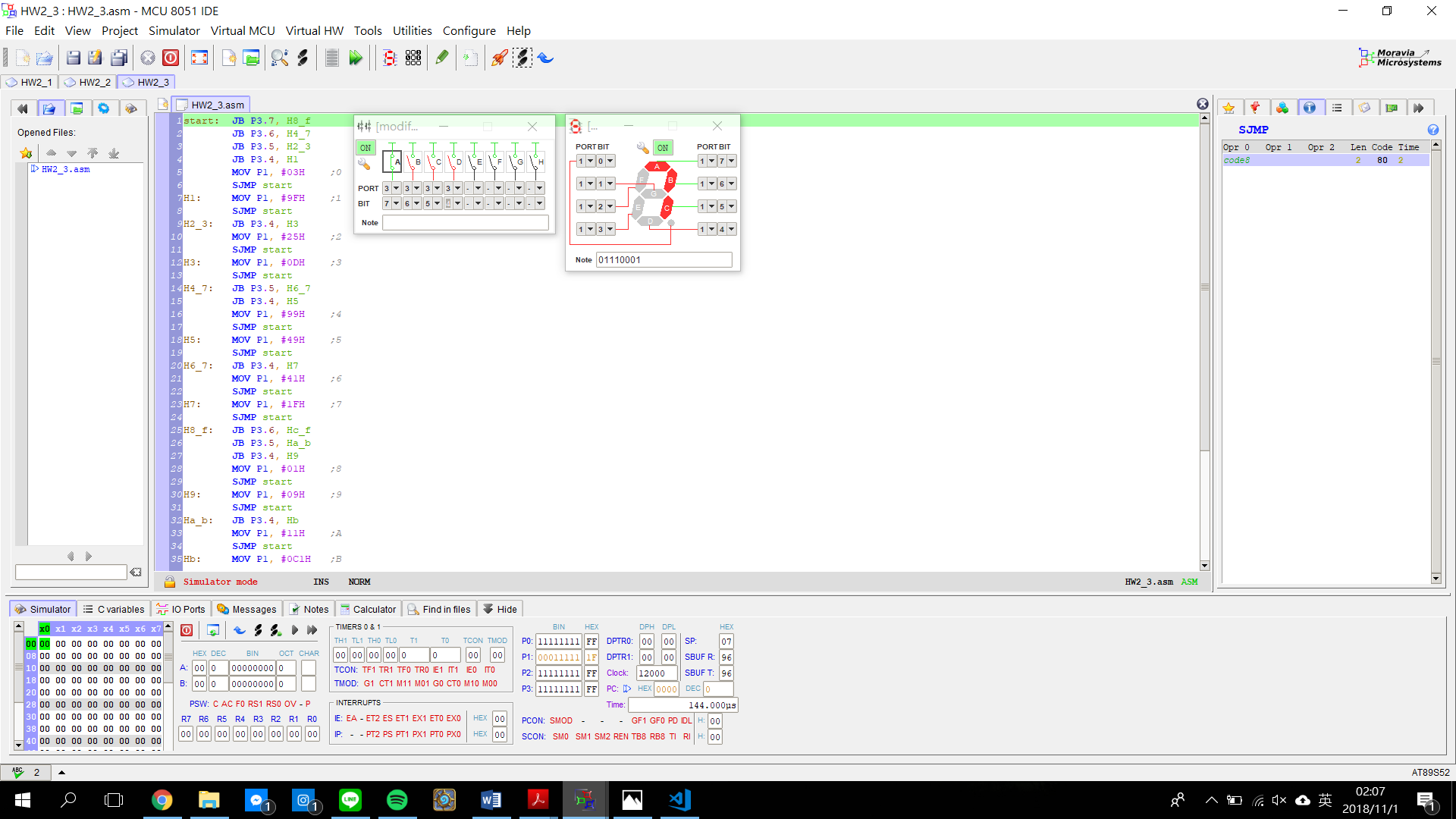
8x220

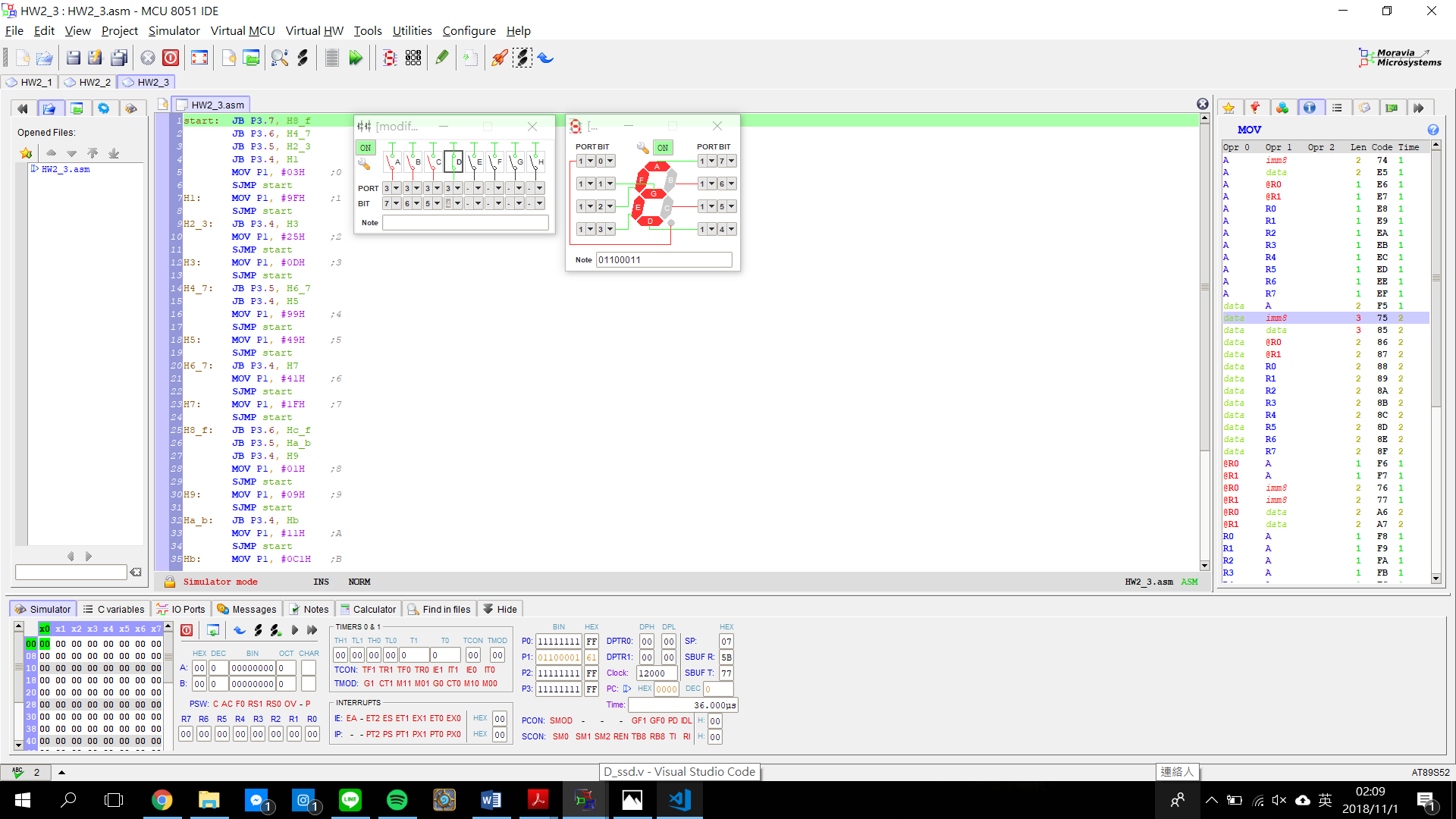
7-segment

LED

+5V

DIP





start: JB P3.7, H8\_f

JB P3.6, H4\_7

JB P3.5, H2\_3

JB P3.4, H1

MOV P1, #03H ;0

SJMP start

H1: MOV P1, #9FH ;1

SJMP start

H2\_3: JB P3.4, H3

MOV P1, #25H ;2

SJMP start

H3: MOV P1, #0DH ;3

SJMP start

H4\_7: JB P3.5, H6\_7

JB P3.4, H5

MOV P1, #99H ;4

SJMP start

H5: MOV P1, #49H ;5

SJMP start

H6\_7: JB P3.4, H7

MOV P1, #41H ;6

SJMP start

H7: MOV P1, #1FH ;7

SJMP start

H8\_f: JB P3.6, Hc\_f

JB P3.5, Ha\_b

JB P3.4, H9

MOV P1, #01H ;8

SJMP start

H9: MOV P1, #09H ;9

SJMP start

Ha\_b: JB P3.4, Hb

MOV P1, #11H ;A

SJMP start

Hb: MOV P1, #0C1H ;B

SJMP start

Hc\_f: JB P3.5, He\_f

JB P3.4, Hd

MOV P1, #63H ;C

SJMP start

Hd: MOV P1, #85H ;D

SJMP start

He\_f: JB P3.4, Hf

MOV P1, #61H ;E

SJMP start

Hf: MOV P1, #71H ;F

SJMP start

*4. (Software Design) (20%)*

*Square wave/pulse generation*

1. *Design an 8051 program that can create a square wave on P1.0 with a frequency of 100 kHz, 10 kHz, 1 kHz, and 100 Hz, assuming a 12 MHz 8051 is used. Analyze the accuracy of your design for these 4 square waves.*

100kHz:

ORG 0000H

LJMP Main

ORG 000BH

T0ISR: CPL P1.0

RETI

ORG 0030H

Main: MOV TMOD, #02H ;T0 MODE 2

MOV TH0, #0FBH ;5us DELAY

SETB TR0 ;START TIMER 0

MOV IE, #82H ;ENABLE T0 INT

LOOP: SJMP $ ;DO NOTHING

Accuracy: roughly 100% because it is auto-reload mode

10kHz:

ORG 0000H

LJMP Main

ORG 000BH

T0ISR: CPL P1.0

RETI

ORG 0030H

Main: MOV TMOD, #02H ;T0 MODE 2

MOV TH0, #0CEH ;50us DELAY

SETB TR0 ;START TIMER 0

MOV IE, #82H ;ENABLE T0 INT

LOOP: SJMP $ ;DO NOTHING

Accuracy: roughly 100% because it is auto-reload mode

1kHZ:

ORG 0000H

LJMP Main

ORG 000BH ;2us

LJMP T0ISR ;2us

ORG 0030H

Main: MOV TMOD, #01H ;T0 MODE 1

MOV IE, #82H ;ENABLE T0 INT

SETB TF0 ;force T0 interrupt

LOOP: SJMP $ ;2us, DO NOTHING

T0ISR: CLR TR0 ;1us

MOV TH0, #0FEH ;2us, 488us DELAY

MOV TL0, #18H ;2us

SETB TR0 ;1us

CPL P1.0

RETI

Accuracy: roughly 100%, the reloading of TH0,TL0 and overhead in setting timer 0 is roughly 12us.

100Hz:

ORG 0000H

LJMP Main

ORG 000BH ;2us

LJMP T0ISR ;2us

ORG 0030H

Main: MOV TMOD, #01H ;T0 MODE 1

MOV IE, #82H ;ENABLE T0 INT

SETB TF0 ;force T0 interrupt

LOOP: SJMP $ ;2us, DO NOTHING

T0ISR: CLR TR0 ;1us

MOV TH0, #0ECH ;2us, 4988us DELAY

MOV TL0, #84H ;2us

SETB TR0 ;1us

CPL P1.0

RETI

Accuracy: roughly 100%, the reloading of TH0,TL0 and overhead in setting timer 0 is roughly 12us.

1. *Design an 8051 program to generate a 5 kHz 60/40 duty cycle pulse wave on P1.4. (60/40 means 60% high and 40% low in one cycle)*

ORG 0000H

LJMP Main

ORG 0030H

Main: MOV TMOD, #02H ;T0 MODE 2

MOV TH0, #0ECH ;20us DELAY

SETB TR0 ;START TIMER 0

LOOP: SETB P1.4

JNB TF0, $ ;DO NOTHING

CLR TF0

JNB TF0, $ ;DO NOTHING

CLR TF0

JNB TF0, $ ;DO NOTHING

CLR TF0

CLR P1.4

JNB TF0, $ ;DO NOTHING

CLR TF0

JNB TF0, $ ;DO NOTHING

CLR TF0

SJMP LOOP

*5. (Software Design) (20%)*

*Music playing*

1. *Refer to the C example in the class for playing Do, Re, Mi, Fa, So, La, Ti, Do-H, redesign the program in 8051 assembly language.*

ORG 0000H ;RAM entry points for...

LJMP MAIN ;main program

ORG 000BH

LJMP T0ISR ;Timer 0 interrupt

ORG 001BH ;2us

LJMP T1ISR ;2us, Timer 1 interrupt

MAIN: MOV TMOD,#11H ;both timers 16-bit mode

MOV R7,#0 ;use R7 as note counter

MOV R6,#5 ;use R6 as timeout counter

MOV IE,#8AH ;Timer 0 & 1 interrupts on

SETB TF1 ;force Timer 1 interrupt

SETB TF0 ;force Timer 0 interrupt

SJMP $ ;2us, ZzZzZzZztime for a nap

T0ISR: CLR TR0 ;stop timer

MOV TH0,#3CH ;reload

MOV TL0,#0B0H

DJNZ R6,EXIT ;if not 5th int, exit

MOV R6,#5 ;if 5th, reset

INC R7 ;increment note

CJNE R7,#LENGTH,EXIT ;beyond last note?

MOV R7,#0 ;yes: reset, C=523 Hz

EXIT: SETB TR0 ;no: start timer, go

RETI ;back to ZzZzZzZ

T1ISR: CPL P1.7 ;1us, music maestro!

CLR TR1 ;1us, stop timer

MOV A,R7 ;1us, get note counter

RL A ;1us, multiply (2 bytes/note)

CALL GETBYTE ;2us, get high-byte of count

MOV TH1,A ;1us, put in timer high register

MOV A,R7 ;1us, get note counter again

RL A ;1us, align on word boundary

INC A ;1us, past high-byte (whew!)

CALL GETBYTE ;2us, get low-byte of count

MOV TL1,A ;1us, put in timer low register

SETB TR1 ;1us, start timer

RETI ;time for a rest

GETBYTE: INC A ;1us, table look-up subroutine

MOVC A,@A+PC ;2us

RET ;2us

TABLE: DW 0FC62H ;C, Timer reload values have been adjusted

DW 0FCCAH ;D (quarter note, etc.)

DW 0FD28H ;E -major third

DW 0FD52H ;F

DW 0FDA1H ;G -perfect fifth

DW 0FDE6H ;A

DW 0FE26H ;B

DW 0FE41H ;C'

temp: LENGTH EQU (temp -TABLE) / 2 ;LENGTH = # of notes

END

1. *Modify the program in (a) so that the time duration for each note can be varying. (You might need to use interrupt in your design if you like. Please study the class material and technical references posted in the website)*

ORG 0000H ;RAM entry points for...

LJMP MAIN ;main program

ORG 000BH

LJMP T0ISR ;Timer 0 interrupt

ORG 001BH ;2us

LJMP T1ISR ;2us, Timer 1 interrupt

MAIN: MOV TMOD,#11H ;both timers 16-bit mode

MOV R7,#0 ;use R7 as note counter

MOV R6,#5 ;use R6 as timeout counter

MOV IE,#8AH ;Timer 0 & 1 interrupts on

SETB TF1 ;force Timer 1 interrupt

SETB TF0 ;force Timer 0 interrupt

SJMP $ ;2us, ZzZzZzZztime for a nap

T0ISR: CLR TR0 ;stop timer

MOV TH0,#3CH ;reload

MOV TL0,#0B0H

DJNZ R6,EXIT ;if not 5th int, exit

MOV R6,#5 ;if 5th, reset

INC R7 ;increment note

CJNE R7,#LENGTH,EXIT ;beyond last note?

MOV R7,#0 ;yes: reset, C=523 Hz

EXIT: SETB TR0 ;no: start timer, go

RETI ;back to ZzZzZzZ

T1ISR: CPL P1.7 ;1us, music maestro!

CLR TR1 ;1us, stop timer

MOV A,R7 ;1us, get note counter

RL A ;1us, multiply (2 bytes/note)

CALL GETBYTE ;2us, get high-byte of count

MOV TH1,A ;1us, put in timer high register

MOV A,R7 ;1us, get note counter again

RL A ;1us, align on word boundary

INC A ;1us, past high-byte (whew!)

CALL GETBYTE ;2us, get low-byte of count

MOV TL1,A ;1us, put in timer low register

SETB TR1 ;1us, start timer

RETI ;time for a rest

GETBYTE: INC A ;1us, table look-up subroutine

MOVC A,@A+PC ;2us

RET ;2us

TABLE: DW 0FC62H ;C, Timer reload values have been adjusted

DW 0FC62H ;C (play again; half note)

DW 0FCCAH ;D (quarter note, etc.)

DW 0FD28H ;E -major third

DW 0FD52H ;F

DW 0FDA1H ;G -perfect fifth

DW 0FDE6H ;A

DW 0FE26H ;B

DW 0FE41H ;C'

DW 0FE41H ;C' (play 4 times; whole note)

DW 0FE41H

DW 0FE41H

temp: LENGTH EQU (temp -TABLE) / 2 ;LENGTH = # of notes

END