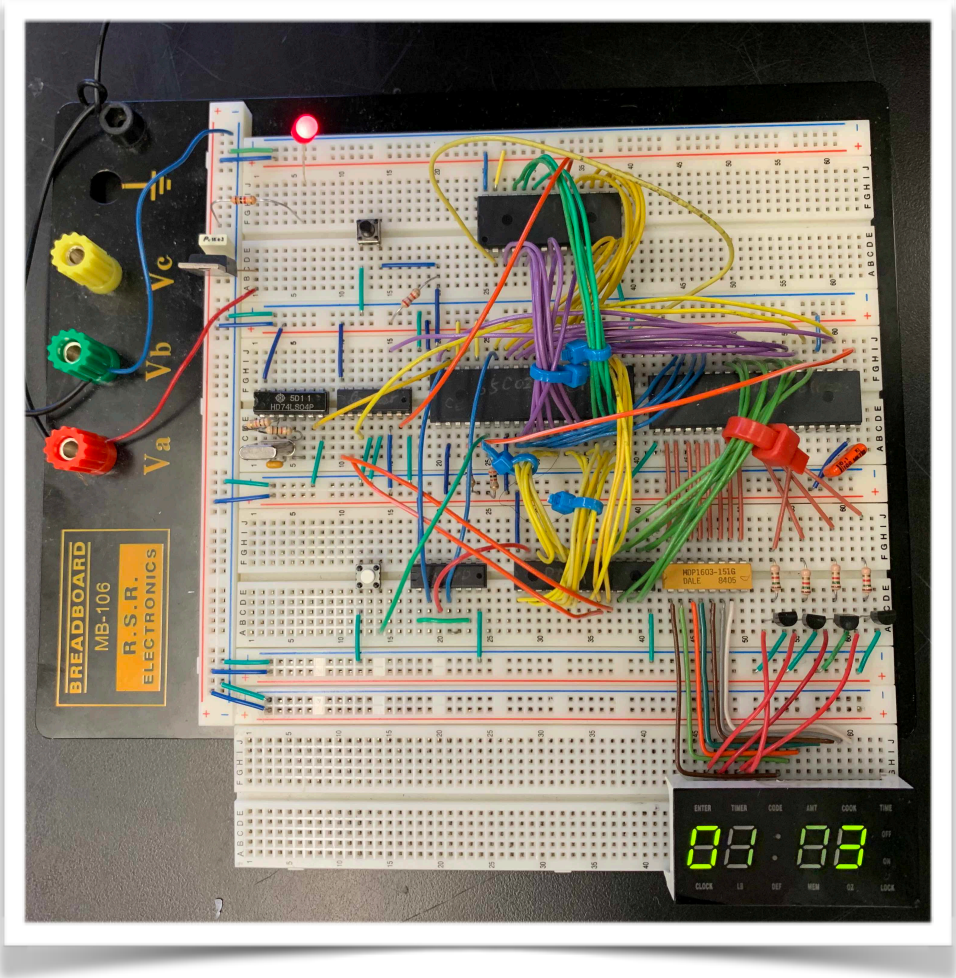
Summative Report: Stopwatch with Seven-Segment

Displays Interface Using 65C02 CPU

by Peter Ke

June 6th, 2019

Partner: Anish Patel



Introduction

This project is to build a stopwatch using 65C02 CPU, 28C64 EEPROM and 65C22 VIA to control 4 seven-segment displays. Methods such as multiplexing, setting up interrupts and indexed addressing are applied.

Technical Specifications

Aspect	Value
Clock Speed	2 MHz
RAM	32 Kb
ROM	8 Kb
Byte Order	little endian
Circuit Voltage	5 V
Wordlength	8-bit
Input / Output (Memory Mapped)	16 I/O lines
# Interrupt Lines	2 (IRQ and NMI)
Address Bus	16 bits
Addressable Memory	64 Kb ($0000 to $FFFF)
Accumulator	8 bits
X Index Register	8 bits
Y Index Register	8 bits
Program Counter	16 bits
Stack Pointer	8 bits
Processor Status	8 bits

Hardware Operations

- Power stabilizes to 5V before reset button pulls resets of W65C02S (CPU) and W65C22S (VIA) low.

- When resets are pulled low, CPU and VIA cycles their reset functions to end all current software operations. CPU starts running code at location pointed by reset vectors. This ensures that both chips are not garbled from unstable power. The reset functions take up 7 cycles.
- The oscillator circuit starts generating clock pulses at 2MHz.
- CPU loads the code from IO0 through IO7 of the EEPROM to D0 through D7 of the CPU. The reset vector points to location $E000, the starting point of the EEPROM in the memory map.
- The code sets all I/O lines on VIA to output by writing hex FF to the data direction registers of the VIA, mapped into the CPU's address range at $8003 (Port A) and $8002 (Port B).
- To display a number of one seven-segment display and not the others, the VIA feeds low to the corresponding transistor of that one display and high to the others to only send data to that display through Port A, since the displays are common cathode. The CPU repeats these commands.
- The IRQ line of CPU is normally pulled high. Timer 1 of the VIA is setup through software and the IRQ line is pulled low every 33 milliseconds approx. After an interrupt is triggered the IRQ line is pulled high again for the next interrupt. This process makes up a real time clock to update the displays.
- The NMI line of CPU is normally pulled high. A button connects this line to ground when it is pressed, pulling it low and triggering a non-maskable interrupt. When the button is released, the NMI line is pulled high again.

I/O Functions of VIA

Port	Input / Output	Function
PA0	Output	f-segments of the seven-segment displays

Port	Input / Output	Function
PA1	Output	g-segments of the seven-segment displays
PA2	Output	e-segments of the seven-segment displays
PA3	Output	d-segments of the seven-segment displays
PA4	Output	c-segments of the seven-segment displays
PA5	Output	b-segments of the seven-segment displays
PA6	Output	a-segments of the seven-segment displays
PB1	Output	controls transistor of SSD showing ones digit of minute
PB2	Output	controls transistor of SSD showing tens digit of second
PB3	Output	controls transistor of SSD showing ones digit of second
PB4	Output	controls transistor of SSD showing tens digit of minute

Integrated Circuits

Part Number	Component / Value	Function	Purpose
W65C02S	U1	Central Processing Unit (CPU)	Sending Instructions to Other Parts (Acting Like a Brain)
W65C22S	U2	Versatile Interface Adapter (VIA)	Controlling Seven-Segment Displays and Generating Timer Interrupt
AT28C64B	U3	Electrically Erasable Programmable Read-Only Memory (EEPROM)	Storing Machine Code to Instruct CPU
SN74LS04N	U4	Hex Inverter Oscillator Circuit	Oscillator Circuit
SN74LS138N	U5	3-8 Decoder	Mapping ROM and I/O Chip into Memory Map of CPU

Part Number	Component / Value	Function	Purpose
SN74LS00N	U6	Quad 2-Input NAND	Interfacing RAM Chip for Proper Read/ Write with CPU
CY7C199CN	U7	32 KB RAM	Fast-Access Memory for CPU

Software

The software of this project can be broken down into four sections, setting up registers, multiplexing the four seven-segment displays, responding to a VIA timer interrupt, and responding to an interrupt generated from the press of a button.

1. Setup

The code starts with giving different registers names so that addresses are referenced with readable English. The assembler will substitute occurrences of these names with the actual addresses.

- The *TimerSetup* routine loads the initial four digits of minute and second (all zeros), clears *TimerCount* and *TimerOn* registers. *TimerOn* can be either 0 or 1 to enable/disable the stopwatch. *TimerCount* will be discussed in the VIA timer interrupt section.
- The *SSD_PatternSetup* routine simply calls the *LoadPattern* subroutine to load 8-bit binary values whose seven digits out of eight correspond to on/off of the seven segments.
- Port A and Port B of the VIA are set to output by writing hex FF into their data direction registers.
- To set up VIA's timer to free-run mode to generate interrupts at every timeout, the CPU writes to the high and low counters of VIA timer 1 to initiate the timer. Writing hex C34E (little endian) will make timer 1 time out every 33 milliseconds. Writing binary 11000000 to the

interrupt enable register will enable timer 1 interrupt in VIA. 6502 powers up with the interrupt disable bit set meaning the processor won't respond to a low on the IRQ line, so the CPU clears the interrupt disable bit (CLI) at the end.

2. Multiplexing

Multiplexing of the displays are done in a loop called *main*. In each iteration, the first display is selected, CPU waits for a bit, writes the pattern of the number in *MinuteTens* to Port A and the first display lights up. The same repeats for the second (*MinuteOnes*), the third (*SecondTens*) and the fourth (*SecondOnes*). The CPU does this very fast so that human eyes only sees four numbers light up at the same time.

- Selection of a display is done through turning on and off transistors. The cathode of each display is connected to the collector of a PNP transistor and the base is connected to a pin of Port B. The displays are common cathode so writing a 0 to the transistor will turn on the display.
- Delays are after selecting a display and before writing a pattern to avoid flickering. Selecting a new display through Port B takes some time and this process might not be finished before the next pattern is sent to Port A. In this way, the previous display will flicker the new pattern undesirably.
- To load the correct pattern of a number, the software uses indexed addressing. Registers that store patterns of 0 to 9 are located at $200 to $209 in memory. By loading the number into the Y register, the CPU looks for the register at location $200 + Y, which will be the pattern of that number. This method is more efficient than comparing a number to 0 through 9.

- The patterns of 0 though 9 are inverted so a simple exclusive-or operation with hex FF will invert them back.

3. Timer Interrupt

The timer interrupt determines when to increase the digits of minute and second. This is triggered by VIA timer 1. It is important to save data in accumulator, X and Y index registers at the beginning of the interrupt routine and retrieve them at the end.

- If *TimerOn* is 0, retrieve accumulator, X and Y registers and return from interrupt routine.
- *TimerCount* stores the number of interrupts generated. When it reaches 30, the digits of minute and second get incremented. Remember that the VIA timer 1 pulls the IRQ line low every 33 milliseconds, so 30 interrupts will give roughly 1 second.
- When incrementing the digits, the CPU determines if there is a carry and increment the next digit. For example, when *SecondOnes* reaches 9, the program jumps to *SecondOnesCarry* subroutine to increment *SecondTens*.
- In *TimerCleanUp* routine, hex FF is loaded into the high and low latches of VIA timer 1 which will be transferred to the counters automatically. This gives the timer the next interval of interrupt.

4. Interrupt from Button

There are two interrupt lines on 65C02, the IRQ and the NMI, which stands for non-maskable interrupt. The NMI line is used for the button which starts the stopwatch going. An interrupt will be triggered each time the NMI line is pulled low. It is important to save data in accumulator, X and Y index registers at the beginning of the interrupt routine and retrieve them at the end.

- Pressing a button means that the user either wants to start the stopwatch or stop the stopwatch. Which one is the purpose is not important to the program itself since it just needs to replace 1 with a 0 or replace 0 with a 1. This can be easily done by an exclusive-or operation with 1 on *TimerOn*.
- Debouncing must be considered since the press of a button is not an instantaneous event. The user might keep the button down (pulling NMI line low) for a few hundred milliseconds which is a long time for the CPU running at 2MHz. Multiple interrupts could be triggered at one press and mess up the program. Delaying a bit after first detection of a low on NMI line will allow the user to release the button before another interrupt is triggered. This will make the software more consistent.

5. Assembly Code (see Appendix A)
6. Assembly Listing (see Appendix B)

Conclusion

The initial purpose of this project is to display UI with a KS0108-driven LCD. However, software complexity and hardware limitations prevented this project from being finished on time. For example, on the software side, plotting an image onto the LCD requires specific algorithms for calculating the column (horizontal) and page (vertical). The entire screen is split into halves and each controlled by one KS0108 chip, and every 8 pixels in a column correspond to one byte of data. This architecture makes the software much more complicated not to mention the lack of support by the Apatco manuals. After switching to seven-segment displays, the program successfully sets up interrupts fired from VIA timer despite poorly phrased data sheet. Tools such

as scope and the 6502 Simulator debugger helps a lot with troubleshooting of both hardware and software. Multiplexing is a smooth process nevertheless.

Improvements can be made to the stopwatch by adding a pair of seven-segment displays to show hours. There can also be memory functionalities to record times, write them to ROM and load them back when the user wishes.

Appendix A

```
PortB:  .SET $8000
PortA:  .SET $8001
DDR_B:  .SET $8002          ; data direction register for Port B
DDR_A:  .SET $8003          ; data direction register for Port A

IER:  .SET $800E            ; interrupt enable register
ACR:  .SET $800B            ; auxiliary control register

VIA_TCL:  .SET $8004            ; lower counter of timer 1
VIA_TCH:  .SET $8005            ; higher counter of timer 1
VIA_TLL:  .SET $8006            ; lower latch of timer 1
VIA_TLH:  .SET $8007            ; higher latch of timer 1

Pattern0:  .SET $200        ; patterns to show 0 to 9 on seven-segment displays
Pattern1:  .SET $201
Pattern2:  .SET $202
Pattern3:  .SET $203
Pattern4:  .SET $204
Pattern5:  .SET $205
Pattern6:  .SET $206
Pattern7:  .SET $207
Pattern8:  .SET $208
Pattern9:  .SET $209

StoreA:  .SET $210          ; register to store accumulator before an interrupt
StoreX:  .SET $211          ; register to store X index register before an interrupt
StoreY:  .SET $212          ; register to store Y index register before an interrupt

TimerCount:  .SET $213      ; number of times a timer interrupt fired
TimerOn:  .SET $214         ; 1 to enable time increase, 0 to disable

MinuteTens:  .SET $215      ; tens digit of minute
MinuteOnes:  .SET $216      ; ones digit of minute
SecondTens:  .SET $217      ; tens digit of second
SecondOnes:  .SET $218      ; ones digit of second

  .ORG $E000
TimerSetup:
  LDA #$00                  ; initialize time to 00:00
  STA MinuteTens
  STA MinuteOnes
  STA SecondTens
  STA SecondOnes
```

```
  LDA #$00              ; clear timer count
  STA TimerCount
  LDA #$00              ; disable time increase
  STA TimerOn

SSD_PatternSetup:
  JSR LoadPattern       ; load patterns of seven-segment displays

VIA_Setup:
  LDA #$FF              ; set both Port A and Port B to output
  STA DDR_A
  STA DDR_B

VIA_TimerSetup:
  LDA #$4E              ; write $C34E to the counters of timer 1 to get timer going
  STA VIA_TCL                   ; and make it time out every 33 milliseconds
  LDA #$C3
  STA VIA_TCH
  LDA ACR               ; set timer 1 to free-run mode
  AND #$7F
  ORA #$40
  STA ACR
  LDA #$C0              ; enable timer 1 interrupt in VIA
  STA IER
  CLI                   ; clear interrupt disable bit to enable interrupts

main:
  JSR Select1           ; select first SSD
  JSR mpDelay                   ; delay
  LDY MinuteTens        ; load tens digit of minute into Y
  LDA $200,Y            ; load the pattern from address $200 + the number
  EOR #$FF              ; invert the pattern
  STA PortA             ; send to Port A

  JSR Select2           ; select second SSD
  JSR mpDelay                   ; delay
  LDY MinuteOnes        ; load one's digit of minute into Y
  LDA $200,Y            ; load the pattern from address $200 + the number
  EOR #$FF              ; invert the pattern
  STA PortA             ; send to Port A

  JSR Select3           ; select the third SSD
  JSR mpDelay                   ; delay
  LDY SecondTens        ; load tens digit of second into Y
  LDA $200,Y            ; load the pattern from address $200 + the number
  EOR #$FF              ; invert the pattern
```

```
  STA PortA                ; send to Port A

  JSR Select4              ; select the fourth SSD
  JSR mpDelay                      ; delay
  LDY SecondOnes           ; load ones digit of second into Y
  LDA $200,Y               ; load the pattern from address $200 + the number
  EOR #$FF                 ; invert the pattern
  STA PortA                ; send to Port A

  JMP main                 ; jump back to main

mpDelay:                   ; delay routine for multiplexing
  LDA #$00                 ; takes 1536 cycles
  LDX #$06
mpDelayB:
  LDY #$FF
mpDelayA:
  DEY
  BNE mpDelayA
  DEX
  BNE mpDelayB
  RTS


debounceDelay:             ; delay routine for debouncing
  LDA #$00                 ; takes 50176 cycles
  LDX #$C4
loopB:
  LDY #$FF
loopA:
  DEY
  BNE loopA
  DEX
  BNE loopB
  RTS

Select1:                   ; select first SSD
  LDA #$EF    ; 11101111
  STA PortB
  RTS

Select2:                   ; select second SSD
  LDA #$FD    ; 11111101
  STA PortB
  RTS
```

```
Select3:                ; select third SSD
  LDA #$FB    ; 11111011
  STA PortB
  RTS

Select4:                ; select fourth SSD
  LDA #$F7    ; 11110111
  STA PortB
  RTS

LoadPattern:            ; load patterns for numbers on displays
  LDA #$7D
  STA Pattern0
  LDA #$30
  STA Pattern1
  LDA #$6E
  STA Pattern2
  LDA #$7A
  STA Pattern3
  LDA #$33
  STA Pattern4
  LDA #$5B
  STA Pattern5
  LDA #$5F
  STA Pattern6
  LDA #$79
  STA Pattern7
  LDA #$7F
  STA Pattern8
  LDA #$7B
  STA Pattern9
  RTS


  .ORG $F000
TimerISR:               ; triggered by VIA timer 1
  STA StoreA            ; save accumulator
  STX StoreX            ; save x register
  STY StoreY            ; save y register

  LDA TimerOn                   ; if time increase is disabled, go to clean up
  CMP #$01              ; else count up to 30
  BNE TimerCleanUp

CountUpTo30:
  INC TimerCount        ; increment count
```

```
  LDA TimerCount        ; if 30 increase timer
  CMP #$1E              ; else go to clean up
  BNE TimerCleanUp
  LDA #$00
  STA TimerCount

increaseTimer:
  LDA SecondOnes        ; increment ones of second
  CMP #$09              ; if 9, carry over
  BEQ SecondOnesCarry
  INC SecondOnes
  JMP TimerCleanUp

SecondOnesCarry:
  LDA #$00              ; increment tens of second
  STA SecondOnes        ; if 5, carry over
  LDA SecondTens
  CMP #$05
  BEQ SecondTensCarry
  INC SecondTens
  JMP TimerCleanUp

SecondTensCarry:
  LDA #$00              ; increment ones of minute
  STA SecondTens        ; if 9, carry over
  LDA MinuteOnes
  CMP #$09
  BEQ MinuteOnesCarry
  INC MinuteOnes
  JMP TimerCleanUp

MinuteOnesCarry:
  LDA #$00              ; increment tens of minute
  STA MinuteOnes        ; if 5, stop stopwatch
  LDA MinuteTens
  CMP #$05
  BEQ MinuteTensCarry
  INC MinuteTens
  JMP TimerCleanUp

MinuteTensCarry:
  LDA #$00
  STA TimerOn

TimerCleanUp:
  LDA #$FF              ; load hex FF into latches for next interrupt
```

```
  STA VIA_TLL
  STA VIA_TLH

  LDA StoreX              ; restore accumulator, x, y registers
  TAX
  LDA StoreY
  TAY
  LDA StoreA
  RTI


  .ORG $F500
IO_ISR:                   ; interrupt routine for button
  STA StoreA              ; save accumulator, x, y registers
  STX StoreX
  STY StoreY

  LDA TimerOn             ; enable time increase if disabled
  EOR #$01                ; disable time increase if enabled
  STA TimerOn

IO_ISR_CleanUp:

  JSR debounceDelay       ; delay for debouncing

  LDA StoreX              ; restore accumulator, x, y registers
  TAX
  LDA StoreY
  TAY
  LDA StoreA
  RTI


  .END
```

Appendix B

Code Adress	Label (Routines)	6502 Instructions	Hex Machine Code
E000	TimerSetup	LDA #$00	A9 00
E002		STA MinuteTens	8D 62 F0
E005		STA MinuteOnes	8D 62 F0
E008		STA SecondTens	8D 3E F0
E00B		STA SecondOnes	8D 3E F0
E00E		LDA #$00	A9 00
E010		STA TimerCount	8D 67 F0
E013		STA TimerOn	8D 14 02
E016	SSD_PatternSetup	JSR LoadPattern	20 B4 E0
E019	VIA_Setup	LDA #$FF	A9 FF
E01B		STA DDR_A	8D 03 80
E01E		STA DDR_B	8D 02 80
E021	VIA_TimerSetup	LDA #$4E	A9 4E
E023		STA VIA_TCL	8D 05 80
E026		LDA #$C3	A9 C3
E028		STA VIA_TCH	8D 05 80
E02B		LDA ACR	AD 0B 80
E02E		AND #$7F	29 7F
E030		ORA #$40	09 40
E032		STA ACR	8D 0B 80
E035		LDA #$C0	A9 C0
E037		STA IER	8D 0E 80
E03A		CLI	58
E03B	Main	JSR Select1	20 AE E0
E03E		JSR mpDelay	20 88 E0
E041		LDY MinuteTens	AC 62 F0
E044		LDA $200,Y	B9 00 02
E047		EOR #$FF	49 FF
E049		STA PortA	8D 01 80

Code Adress	Label (Routines)	6502 Instructions	Hex Machine Code
E04C		JSR Select2	20 AE E0
E04F		JSR mpDelay	20 88 E0
E052		LDY MinuteOnes	AC 62 F0
E055		LDA $200,Y	B9 00 02
E058		EOR #$FF	49 FF
E05A		STA PortA	8D 01 80
E05D		JSR Select3	20 AE E0
E060		JSR mpDelay	20 88 E0
E063		LDY SecondTens	AC 3E F0
E066		LDA $200,Y	B9 00 02
E069		EOR #$FF	49 FF
E06B		STA PortA	8D 01 80
E06E		JSR Select4	20 AE E0
E071		JSR mpDelay	20 88 E0
E074		LDY SecondOnes	AC 3E F0
E077		LDA $200,Y	B9 00 02
E07A		EOR #$FF	49 FF
E07C		STA PortA	8D 01 80
E07F		JMP main	4C 3B E0
E082	mpDelay	LDA #$00	A9 00
E084		LDX #$06	A2 06
E086	mpDelayB	LDY #$FF	A0 FF
E088	mpDelayA	DEY	88
E089		BNE mpDelayA	D0 FD
E08B		DEX	CA
E08C		BNE mpDelayB	D0 FA
E08E		RTS	60
E08F	debounceDelay	LDA #$00	A9 00
E091		LDX #$C4	A2 C4
E093	loopB	LDY #$FF	A0 FF
E095	loopA	DEY	88

Code Adress	Label (Routines)	6502 Instructions	Hex Machine Code
E096		BNE loopA	D0 FD
E098		DEX	CA
E099		BNE loopB	D0 F8
E09B		RTS	60
E09C	Select1	LDA #$EF	A9 EF
E09E		STA PORTB	8D 00 80
E0A1		RTS	60
E0A2	Select2	LDA #$FD	A9 FD
E0A4		STA PORTB	8D 00 80
E0A7		RTS	60
E0A8	Select3	LDA #$FB	A9 FB
E0AA		STA PORTB	8D 00 80
E0AD		RTS	60
E0AE	Select4	LDA #$F7	A9 F7
E0B0		STA PORTB	8D 00 80
E0B3		RTS	60
E0B4	LoadPattern	LDA #$7D	A9 7D
E0B6		STA Pattern0	8D 09 02
E0B9		LDA #$30	A9 30
E0BB		STA Pattern1	8D 09 02
E0BE		LDA #$6E	A9 6E
E0C0		STA Pattern2	8D 09 02
E0C3		LDA #$7A	A9 7A
E0C5		STA Pattern3	8D 09 02
E0C8		LDA #$33	A9 33
E0CA		STA Pattern4	8D 09 02
E0CD		LDA #$5B	A9 5B
E0CF		STA Pattern5	8D 09 02
E0D2		LDA #$5F	A9 5F
E0D4		STA Pattern6	8D 09 02
E0D7		LDA #$79	A9 79

Code Adress	Label (Routines)	6502 Instructions	Hex Machine Code
E0D9		STA Pattern7	8D 09 02
E0DC		LDA #$7F	A9 7F
E0DE		STA Pattern8	8D 09 02
E0E1		LDA #$7B	A9 7B
E0E3		STA Pattern9	8D 09 02
E0E6		RTS	60
F000	TimerISR	STA StoreA	8D 10 02
F003		STX StoreX	8E 11 02
F006		STY StoreY	8C 12 02
F009		LDA TimerOn	AD 14 02
F00C		CMP #$01	C9 01
F00E		BNE TimerCleanUp	D0 57
F010	CountUpTo30	INC TimerCount	EE 67 F0
F013		LDA TimerCount	AD 67 F0
F016		CMP #$1E	C9 1E
F018		BNE TimerCleanUp	D0 4D
F01A		LDA #$00	A9 00
F01C		STA TimerCount	8D 67 F0
F01F	increaseTimer	LDA SecondOnes	AD 3E F0
F022		CMP #$09	C9 09
F024		BEQ SecondOnesCarry	F0 18
F026		INC SecondOnes	EE 3E F0
F029		JMP TimerCleanUp	4C 67 F0
F02C	SecondOnesCarry	LDA #$00	A9 00
F02E		STA SecondOnes	8D 3E F0
F031		LDA SecondTens	AD 3E F0
F034		CMP #$05	C9 05
F036		BEQ SecondTensCarry	F0 06
F038		INC SecondTens	EE 3E F0
F03B		JMP TimerCleanUp	4C 67 F0
F03E	SecondTensCarry	LDA #$00	A9 00

Code Adress	Label (Routines)	6502 Instructions	Hex Machine Code
F040		STA SecondTens	8D 3E F0
F043		LDA MinuteOnes	AD 62 F0
F046		CMP #$09	C9 09
F048		BEQ MinuteOnesCarry	F0 18
F04A		INC MinuteOnes	EE 62 F0
F04D		JMP TimerCleanUp	4C 67 F0
F050	MinuteOnesCarry	LDA #$00	A9 00
F052		STA MinuteOnes	8D 62 F0
F055		LDA MinuteTens	AD 62 F0
F058		CMP #$05	C9 05
F05A		BEQ MinuteTensCarry	F0 06
F05C		INC MinuteTens	EE 62 F0
F05F		JMP TimerCleanUp	4C 67 F0
F062	MinuteTensCarry	LDA #$00	A9 00
F064		STA TimerOn	8D 14 02
F067	TimerCleanUp	LDA #$FF	A9 FF
F069		STA VIA_TLL	8D 07 80
F06C		STA VIA_TLH	8D 07 80
F06F		LDA StoreX	AD 11 02
F072		TAX	AA
F073		LDA StoreY	AD 12 02
F076		TAY	A8
F077		LDA StoreA	AD 10 02
F07A		RTI	40
F500	IO_ISR	STA StoreA	8D 10 02
F503		STX StoreX	8E 11 02
F506		STY StoreY	8C 12 02
F509		LDA TimerOn	AD 14 02
F50C		EOR #$01	49 01
F50E		STA TimerOn	8D 14 02
F511	IO_ISR_CleanUp	JSR debounceDelay	20 8F E0

Code Adress	Label (Routines)	6502 Instructions	Hex Machine Code
F514		LDA StoreX	AD 11 02
F517		TAX	AA
F518		LDA StoreY	AD 12 02
F51B		TAY	A8
F51C		LDA StoreA	AD 10 02
F51F		RTI	40
F520	.END		