CTRONIX AutoCall LD1.0
8048 microcontroller-based
large wall alarm clock

Paul Dallas, SV1UI, 2015, with the (ever so) kind assistance of Ctronix.

INTRO

Sometime in the 90's, I salvaged an AutoCall LD1.0 display unit from a skip. I powered it up recently and it worked fine, so I wrote some software for it to turn it into a wall clock. There are three reasons I present this here:

1. To assist in understanding the software written for its MAB8039HL (8048) processor; the code may be useful for others making a microcontroller-based clock.
2. As a tribute to an absolutely magnificent design by Ctronix!
3. As an aid to anyone repairing an AutoCall unit of that era.

Please note that Ctronix have (ever so) kindly agreed to my publishing the schematics, and have even provided the PCB layouts shown in Appendix E; their co-operation exceeded my wildest dreams...

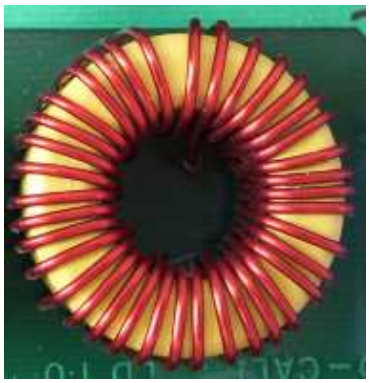
CONTENTS

FEATURES

- Shows hours, minutes, seconds, day, month, year.
- 12h or 24h display selectable, with AM / PM indication.
- Automatically caters for leap years.
- Displays "HAPPY BIRTHDAY" on a specified (hard-coded, I fear) date.
- Alarm with "silence" function.
- Battery backup to cater for mains failures (power cuts / brownouts / blackouts).
- Timing obtained from mains frequency, with automatic fall-over to internal crystal clock during mains failure.
- Display brightness automatically adapts to ambient light level.

SCHEMATICS

The schematics shown in Appendix F are generally complete, apart from the decoupling capacitors. Each IC (pretty much) is decoupled with a 10nF ceramic. Also, the values of some components could not be read without removing them from the board, something I wanted to avoid. So, no values are provided for these. I didn't go to great lengths to record everything, since this was meant to be published more as a reference project rather than one for direct duplication. I have not managed to read the part numbers of T12 and D21, as these were fitted on wrap-around heatsinks. I suppose any decent PNP TO220 transistor (e.g. a TIP42B) and fast TO220-2 diode (e.g. a BYW29-100 - do observe polarity!) would probably do fine; If you'd really like to know the original part numbers, let me know and I'll pull the heatsinks off and see what they are.



Inductor L1 is a different story. You can calculate its value from the 78S40 datasheet (I get about 120 microhenries), and then find a suitable toroid / gapped pot core / E-I ferrite core / whatever to wind it on. The original has "plenty" of turns of wire on a "large" (about 1,5" dia.) ferrite core. If you have a specific core in mind and know its Al and Bmax at 25kHz, I could work the rest out for you on request.

Finally, the piezoelectric sounder LS1 is marked "AT-23K Taiwan, Projects Unlimited". It is an intriguing animal having 3 pins. It is wrapped around transistor T2, as follows:

- Its "main" terminal connects to T2's Collector.
- Its "ground" terminal connects to T2's Emitter.
- Its "feedback" terminal connects to R6 and R7.

I did find such sounders for sale on the internet, presumably old stock. I suppose other externally-driven piezoelectric sounders (e.g. Digi-Key's AT-2830-TWT-R) will probably also do.

HOW IT WORKS

The MAB8039HL microcontroller IC1 runs in external program memory mode, with the program code stored in IC3. Only half of IC3 is actually used, so presumably a 2732 EPROM could also be used instead of the 2764.

As the microcontroller multiplexes its data and address lines (lower 8 bits) on its pins D0 to D7, IC4 is used to demultiplex the address lines and provide separate address lines to IC3.

The sixteen 7-segment displays are driven in a multiplexed fashion using the following 8-state cycle:

State 0: The microcontroller serially puts out data bits for segments A, B, C, D, E, F, G of the 7-segment displays. D0 carries information for the lowermost row, while D3 carries information for the uppermost row. The microcontroller software writes these using a MOVX instruction, which has the effect of toggling the microcontroller's WR pin; this in turn clocks the segment data into shift registers IC6 to IC9 (via IC5), which retain it for the next state in the cycle. Note that in state 0, the microcontroller writes the segment data for the four leftmost displays.

State 1: The microcontroller strobes P2.7 low to switch on the leftmost display column.

State 2: The microcontroller serially puts out the segment data for the 2nd. from left display column.

State 3: The microcontroller strobes P2.6 low to switch on the 2nd. from left display column.

State 4: The microcontroller serially puts out the segment data for the 2nd. from right display column.

State 5: The microcontroller strobes P2.5 low to switch on the 2nd. from right display column.

State 6: The microcontroller serially puts out the segment data for the rightmost display column.

State 7: The microcontroller strobes P2.4 low to switch on the rightmost display column.

We then return to state 0. With the original AutoCall LD1.0 software, the entire cycle took about 13msec, and each state lasted about 1,6msec.

Ctronix have pointed out that the unit I have is rather a rare version. A more common variant is the AutoCall with 20mm digit height configured as 6 rows of 4 digits. The two extra rows in that case are driven by data lines D4 and D5.

The microcontroller port P1 is used as follows:

- P1.0 to P1.4 are used as inputs, and are controlled by DIP switches 1 to 5.
- P1.5 drives the sounder. When pulled low, T3 and T1 are switched on, and the piezo sounder squeaks. T2 forms an oscillator, with feedback taken directly from the piezo sounder itself.
- P1.6 and P1.7 are used for driving the RS485 transceiver, IC2. These are not actually used in the clock application, since IC2 is used solely as a receiver.

Other processor lines:

- The !INT input is driven by the RS485 receiver of IC2. This would normally be used to receive information from the Ctronix control unit, but in this application it is used to receive a filtered sample of 50 or 60Hz AC mains, which is used as the clock's source of timing.
- The T0 general purpose input is brought to the centre pin of a three-pin socket; I have connected this to ground via pushbutton switch SA, which is used as the "Alarm on-off / Alarm silence / Clock set" pushbutton.
- The T1 general purpose input is driven by DIP switch 6.

The circuit operates off two separate power supply rails (three if you also count PSU_HV, but never mind): The +13V rail which powers the segment display driver circuitry, and the +5V rail which powers the microcontroller and parts around it (and also IC10). The logic level translation is performed by IC5. One advantage of this arrangement is that it allows the display segment voltage

to vary; I used this to provide a display dimming function, as described later.

The +13V supply is controlled by IC11, the (then) ultra-modern 78S40; this runs at about 25kHz. R56 and R57 provide feedback for the +13V rail, which is compared against IC11's internal 1,25V reference. If the voltage on the +13V rail is too low, IC11 increases the width of the pulses supplied to provide more energy to the L-C filter (L1, C10) and thus increase the output voltage.

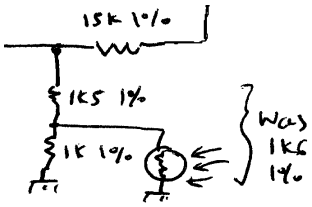
The watchdog is an interesting design: The first display column strobe (COL1) is supposed to pulse low every 13msec or so. This pulse train goes through C3 and IC10F and each pulse discharges C2 through D19. If pulses on COL1 stop, then the input of IC10F will coast to 0V, its output will become permanently high, so C2 will stop discharging via D19. C2 will thus slowly charge up through R5, eventually reaching the threshold on the input of IC10F, thus making IC10F's output low. This pulls pin 1 (the diode cathode) of IC11 low, pulling its anode low. This in turn pulls the input OP+ of IC11's op-amp low, bringing its output (pin 4) low and thus causing a microcontroller reset. The same op-amp also keeps the microcontroller reset if the input voltage is too low.

The entire circuit draws around 1,1A DC from a 24V power supply, with all 7-segment displays on.

MINOR HARDWARE ADDITIONS

1. I found that the display was very comfortable to read even in bright sunlight, but too bright (erm... annoying, really) at night. It was never meant to be used as a bedroom clock anyway; it was supposed to be installed in restaurants and pubs.

   So, I replaced R57 with a 1K5 1% resistor in series with a FR7-1020 CdS photocell. A 1K resistor was then placed directly across the photocell, and the photocell was poked through a hole in the enclosure. This way, the +13V rail is at 8,8V when the photocell is dark (thus dimming the displays), and at 12,9V under bright ambient light.



2. I never had the AutoCall LD1.0's control unit; I only ever had the display unit. So I guessed the display unit was meant to be powered from 24VDC unregulated. I built a simple power supply for this in a separate box, and also included 8 NiCd batteries (NiMh will do just as well, if not better) to keep the clock running during power cuts. The power supply also provides a filtered sample of AC mains to the clock, which is used as the clock's source of timing. Note that CB is a non-polar capacitor!

SOFTWARE

The software is written in 8048 assembly. The complete listing can be found in Appendix C. It is assembled with Asm48 version 0.4.1, an excellent piece of multi-platform freeware by the Adventure Vision Development Team of MEGA (http://www.adventurevision.net); this is available on SourceForge, among other places.

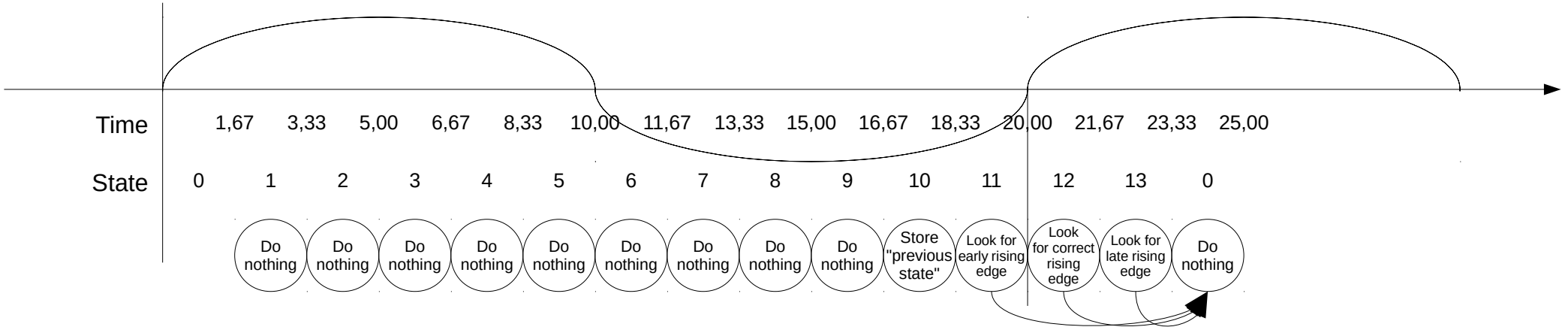
After initialising the variables and performing some preliminary tasks, the microcontroller sets its timer to time out after about 1,67msec (clocked by the crystal). The microcontroller then enters a message loop which essentially does nothing apart from waiting for the timer to time out. This message loop structure does nothing worth interrupting, so timer interrupts have not been used.

Every time the timer times out it is restarted, and the message loop branches out to the state machines.

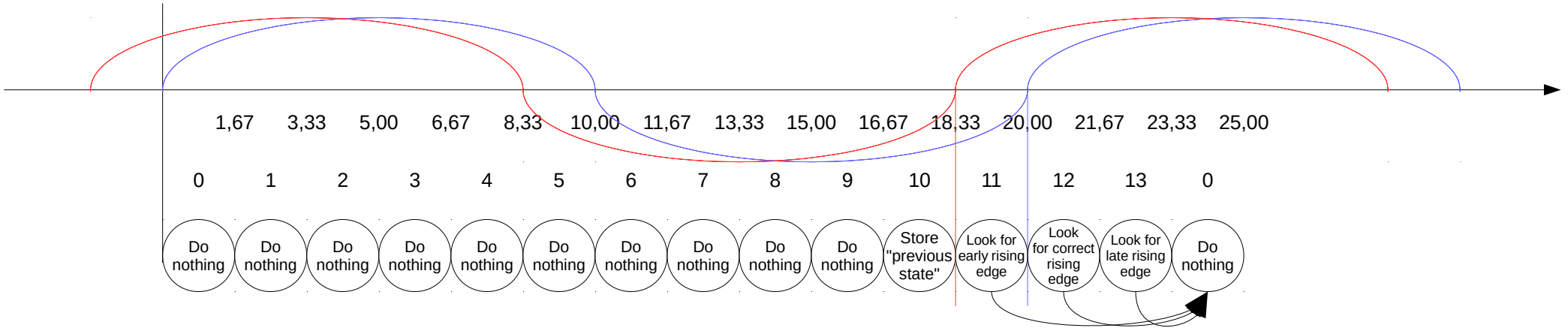
The clock has two state machines, the clock recovery state machine and the main state machine.

Clock recovery state machine

This is a form of PLL which provides the clock tick once a second. It does this by detecting rising edges on the !INT input, and setting a flag (R7's bit 3) every 50th or 60th rising edge detected (depending on whether DIP switch 6 on T1 is set for 50 or 60Hz operation). Unfortunately, the 8048 series microcontrollers only have a level-triggered interrupt, which is pretty inconvenient. For this reason, a polled arrangement is used:



Let's say the 50Hz sinewave on !INT had a rising edge at t=0,00msec. Its following rising edge will be at 1/50Hz=20,00msec.
The "previous state" of !INT is stored in state 10 at 16,67msec, which is certainly logic 0.
The "rising edge" on !INT should occur in state 12 at 20,00msec.
However, it might just not be caught at state 12, as it might just not have risen yet. In this case, it will be caught at state 13.



Blue: If in the previous cycle the rising edge had been caught in state 12, this would have reset the state machine exactly on the rising edge. In this case, the rising edge will again be caught in state 12, or if it just has not risen yet, then it will be caught in state 13.
Red: However, if in the previous cycle the rising edge had been caught in state 13, this would have reset the state machine 1,67msec into the wave. The rising edge will now be caught in state 11, or if it just has not risen yet, then it will be caught in state 12.

Let us examine the 50Hz case (the 60Hz case is similar): Once a rising edge on !INT is detected, nothing is done for 9x1,67msec=15msec. From 16,67msec onwards, we look for a new rising edge on !INT. If it is found, then the clock tick counter is decremented and the clock recovery state machine is reset. On the 50th. count R7's bit 3 is set, to flag that one second has elapsed.

If a rising edge on !INT is not found until 21,67msec, we give up; there is probably a power cut. In this case a flag (R7 bit 1) is left clear, informing the clock recovery state machine that !INT is not toggling as it should; in this case, the clock recovery state machine itself decrements the clock tick counter every 20msec (as timed by the microcontroller's 1,67msec timer), to fill in for the missing rising edge on !INT.

This way, if there is a signal on the !INT input at approximately the correct mains frequency, it is used as the clock's source of timing. If not, then the clock generates its own timing based on its crystal-controlled timer. In practice, this works remarkably well.

The relevant flowcharts can be found in Appendices A and B; I kept these out of main text, as they are rather complicated and could do more harm than good...

Main state machine

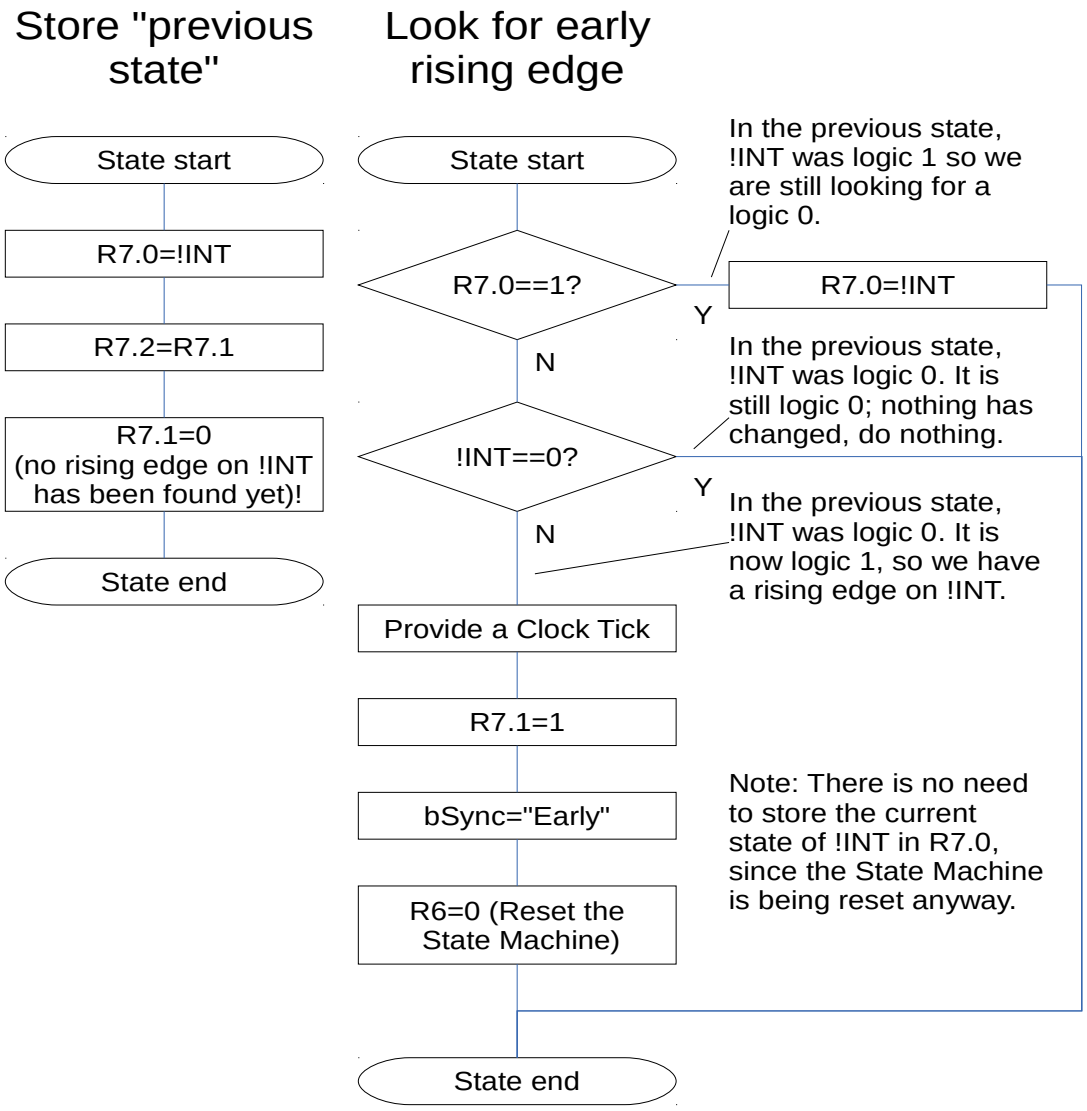
This runs the 8-state cycle for the displays, described above in the "how it works" section. States 1, 3, 5 and 7 don't really have much to do as regards driving the displays. For this reason, other tasks are also performed in state 1 (incrementing the time if one second has elapsed) and state 3 (alarm clock functions); nothing extra has been implemented in states 5 and 7 (yet).

TO-DO LIST

There are a couple more things I would still like to do one day:

1. Find the part numbers of the (few) missing parts.
2. Improve the photocell's response to light: I have found that even in a well-lit room, the 7-segment displays don't quite reach maximum brightness. A lower resistance photocell might be better here.
3. Make the birthdate where the clock displays "HAPPY BIRTHDAY" settable.
4. Make the clock automatically adjust for daylight savings. This is a bit of a pig, because different countries and regions have different arrangements for daylight savings. Worse still, these might change as governments change their policies. So, the "daylight savings start date", "daylight savings end date" and maybe also the time at which the clock goes back / forth an hour should be settable.
5. The clock currently displays the date in "European" format, i.e. DD-MM-YYYY. I suppose this could be made settable. An idea would be to make it so that if SW6 is set for 50Hz (Europe) then the European DD-MM-YYYY format is used, and if SW6 is set for 60Hz (U.S.) then the MM-DD-YYYY display format is used.

APPENDIX A: CLOCK RECOVERY STATE MACHINE FLOWCHARTS



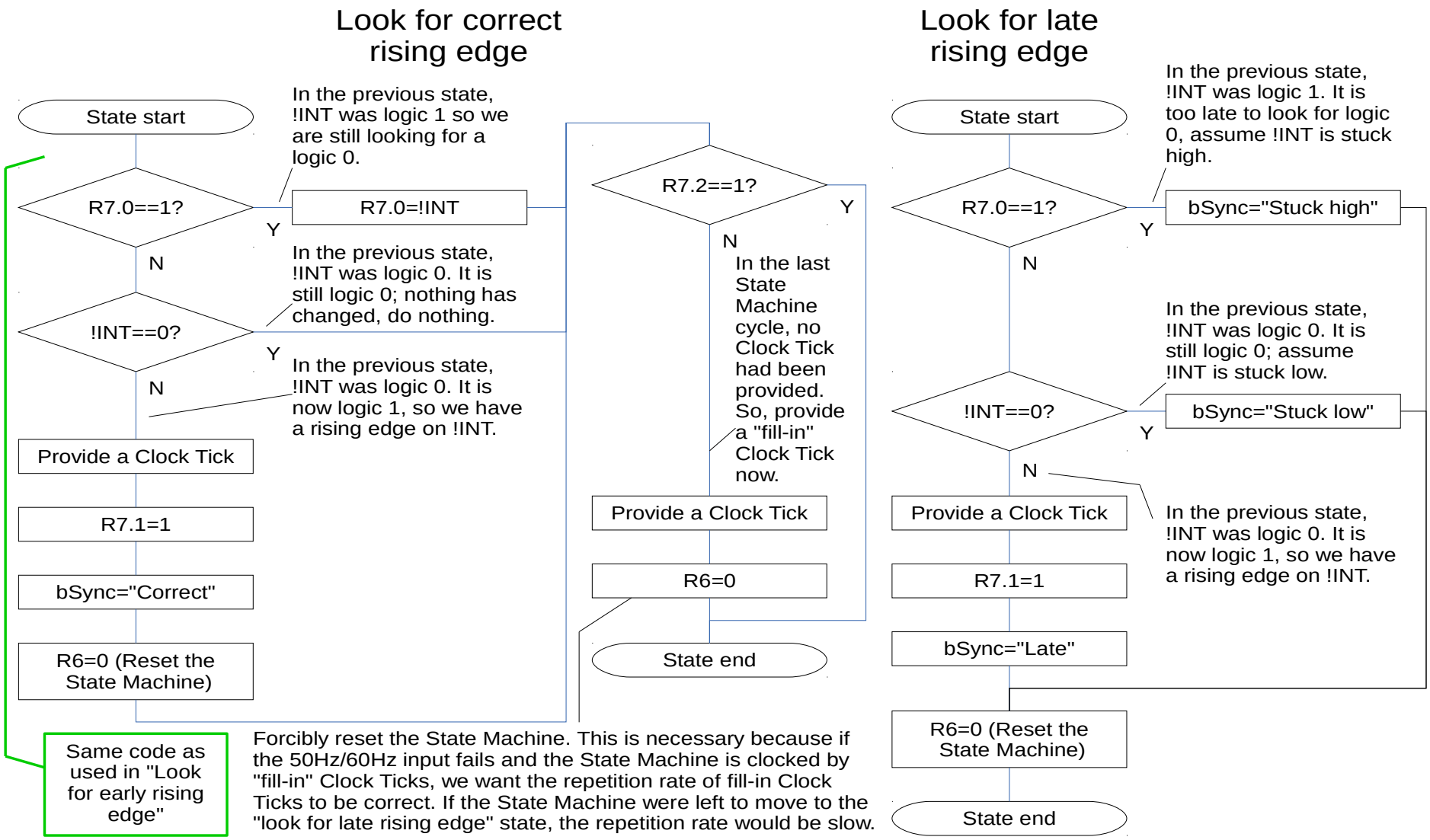
Flag R7.0 is used for storing the "previous state of !INT", i.e. the state of !INT during the previous state.

Flag R7.1 shows whether a rising edge was found on !INT:
- R7.1=0: A rising edge was not found on ! INT, so provide a "fill-in" Clock Tick in the next Clock Recovery State Machine cycle.
- R7.1=1: A rising edge was found on !INT, so do not provide a "fill-in" Clock Tick in the next Clock Recovery State Machine cycle.

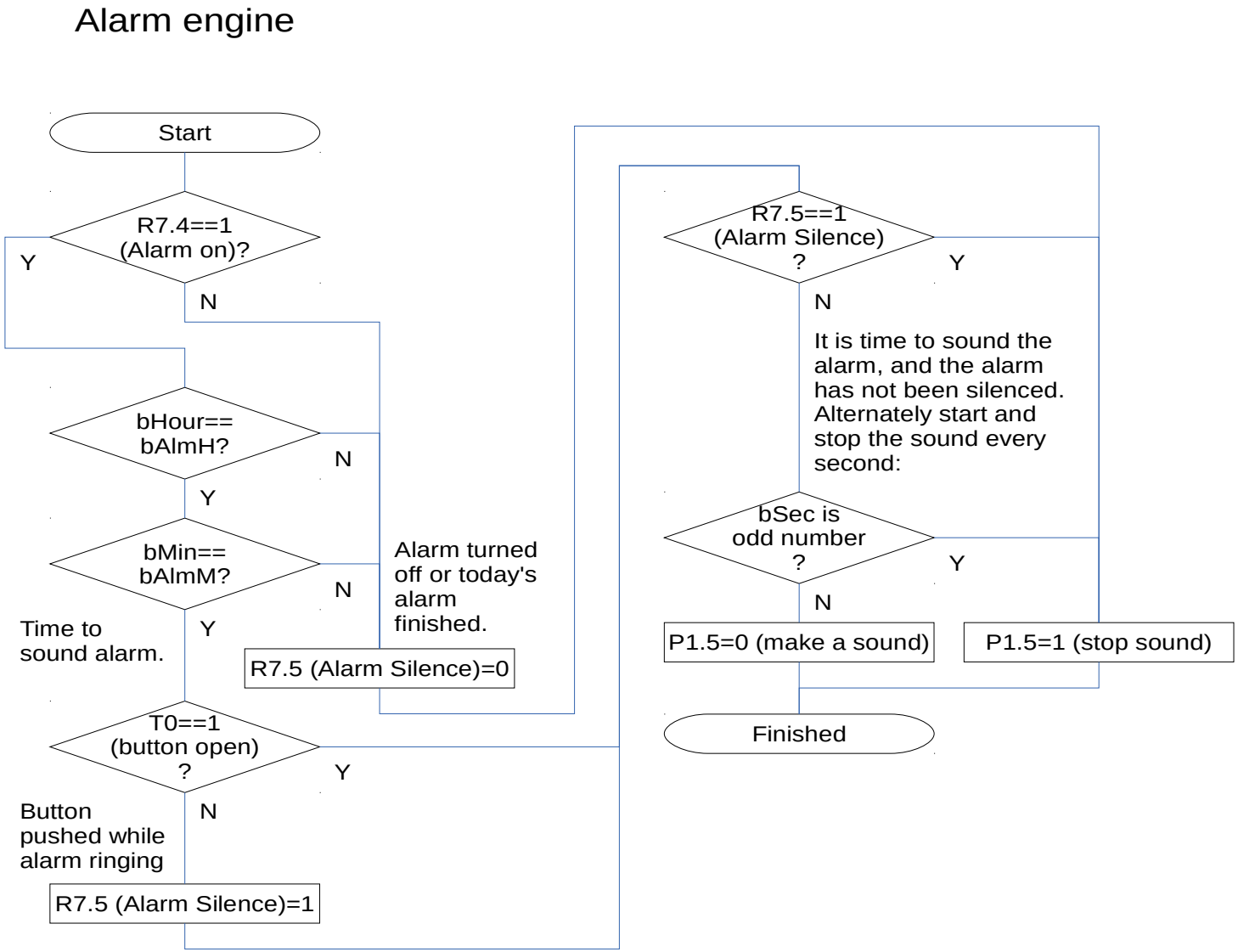
Flag R7.2 is a copy of R7.1, made early in the Clock Recovery State Machine cycle. It signals wheterh a rising edge was found on !INT in the previous Clock Recovery State Machine Cycle.

R6 is the state counter for the Clock Recovery State Machine.

bSync is a memory variable used for displaying the state of the Clock Recovery State Machine where the rising edge on ! INT was found. This is useful for debugging.


Look for correct rising edge
State start
In the previous state, !INT was logic 1 so we are still looking for a logic 0.
R7.0==1?
R7.0=!INT
Y
N
In the previous state, !INT was logic 0. It is still logic 0; nothing has changed, do nothing.
!INT==0?
Y
N
In the previous state, !INT was logic 0. It is now logic 1, so we have a rising edge on !INT.
Provide a Clock Tick
R7.1=1
bSync="Correct"
R6=0 (Reset the State Machine)
R7.2==1?
Y
N
In the last State Machine cycle, no Clock Tick had been provided. So, provide a "fill-in" Clock Tick now.
Provide a Clock Tick
R6=0
State end
Same code as used in "Look for early rising edge"
Forcibly reset the State Machine. This is necessary because if the 50Hz/60Hz input fails and the State Machine is clocked by "fill-in" Clock Ticks, we want the repetition rate of fill-in Clock Ticks to be correct. If the State Machine were left to move to the "look for late rising edge" state, the repetition rate would be slow.
Look for late rising edge
State start
In the previous state, !INT was logic 1. It is too late to look for logic 0, assume !INT is stuck high.
R7.0==1?
bSync="Stuck high"
Y
N
In the previous state, !INT was logic 0. It is still logic 0; assume !INT is stuck low.
!INT==0?
bSync="Stuck low"
Y
N
In the previous state, !INT was logic 0. It is now logic 1, so we have a rising edge on !INT.
Provide a Clock Tick
R7.1=1
bSync="Late"
R6=0 (Reset the State Machine)
State end

APPENDIX B: ALARM ENGINE FLOWCHART



Flag R7.4 will be used as the "Alarm On" flag. When it is logic 1, the alarm is on.

Flag R7.5 will be used as the "Alarm Silenced" flag. It will be 0, unless the alarm has been silenced by pushing the button on T0 while the alarm is ringing.

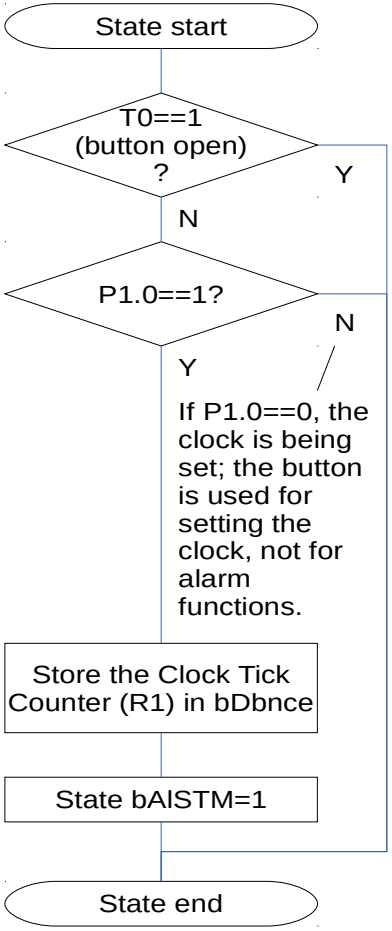
bDbnce is a variable which will store the value of the Clock Tick Counter (R1) the moment the button is pressed.
R1 cycles every second. So, if the button is still pressed when R1 again comes around to R1==bDbnce, we know the button has been pressed for one second.

bAlSTM is a variable which will be used as the Alarm State Machine Counter.

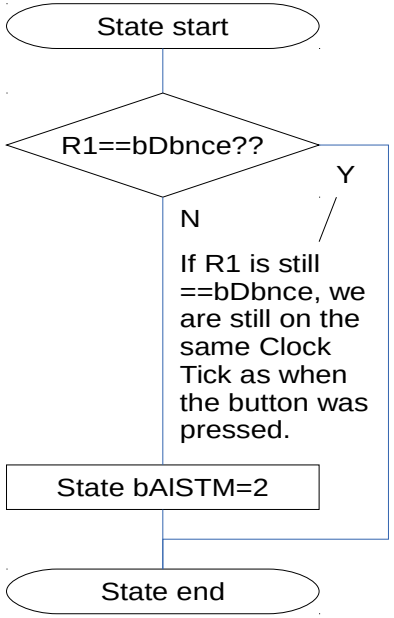
T0 button Alarm on-off State Machine

A three-state state machine is used to determine whether the button on T0 has been held pressed for 1 second, to toggle Alarm on-off.
In State 0, the State Machine checks whether the button has been pressed. If it has, it moves to state 1. The Clock Tick Counter (R1) is stored in bDbnce.
In State 1, the State Machine waits until the Clock Tick Counter advances beyond its current value (==bDbnce).
In State 2, the State Machine knows that the Clock Tick Counter has advanced beyond bDbnce. The Clock Tick Counter cycles every 1 second, so when the Clock Tick Counter is again ==bDbnce, we know that 1 second has elapsed. If the button is still pressed, then Alarm on-off is toggled.

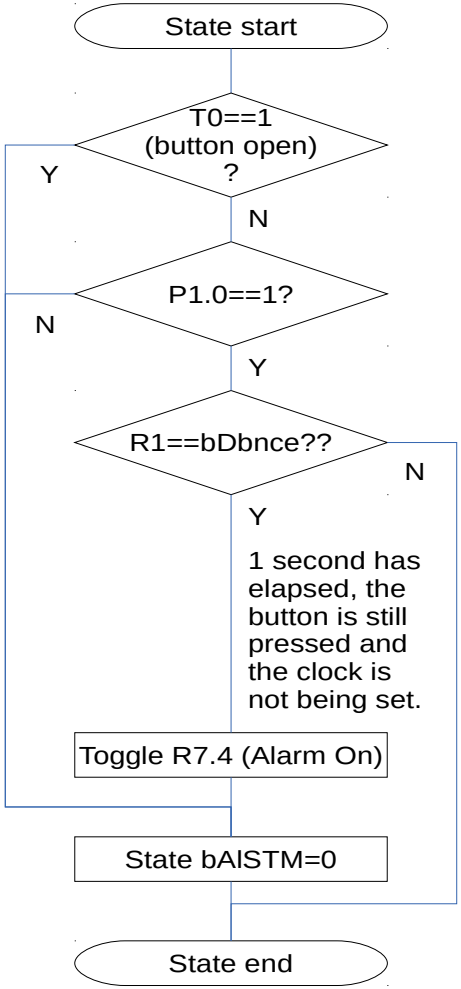
State 0: Check button on T0



State 1: Wait for Clock Tick



State 2: Wait for 1 second



APPENDIX C: SOFTWARE LISTING

In days of old when men were bold though trousers were invented, we used to load code into our terminals, Sinclairs, Commodore PETs, whatever by typing it in. Clearly, I don't suggest anyone does this. I include the software listing below none-the-less, in case file ALARMH.asm gets detached from this document and is no longer obtainable for some reason.

```
; ALARMH.ASM: Contains the complete clock and alarm software.

; Register use: Only Bank 0 registers will be used in main code. It is
; generally wise to leave Bank 1 registers unused, for use by interrupt service
; routines.
;   R0: General purpose register, may be altered by routines.
;   R1: Clock tick counter, decrements from 50 or 60 to 0.
;   R2: Character to be written to the top display.
;   R3: Character to be written to the display 2nd. from top.
;   R4: Character to be written to the display 2nd. from bottom.
;   R5: Character to be written to the bottom display.
;   R6: Clock Recovery State Machine current state.
;   R7: This is used as a bit field:
;       R7.0: "previous !INT state" flag. Stores the previous state of !INT, so
;         it can be compared to the current state of !INT.
;       R7.1: Set to 1 if a rising edge was found on !INT in the current Clock
;         Recovery State Machine cycle.
;       R7.2: a copy of R7.1, made early in the Clock Recovery State Machine
;         cycle. This is logic 1 if a rising edge was found on !INT in the
;         previous Clock Recovery State Machine cycle.
;       R7.3: Seconds Increment Flag. Set to 1 if the Clock Tick Counter (R1)
;         has decremented to zero; set to 0 otherwise.
;         - It is set to 1 by the Clock Recovery State Machine.
;         - It is set to 0 in Main State Machine State 1 (lSt1), which uses it.
;       R7.4: Alarm On flag. When it is logic 1, the alarm is on.
;       R7.5: Alarm Silenced flag. It is 0, unless the alarm has been silenced
;         by pushing the button on T0 while the alarm is ringing.

; Flags used:
;   F0: Not used.
;   F1: Not used, useful for passing information to interrupt service routines.


; DIP switches on port P1:
;   SW1 on P1.0: Used as "Clock Set". To set the clock, set to On (logic 0).
;   SW2 on P1.1: -
;   SW3 on P1.2:  | Used by subroutine SetClock to set the clock.
;   SW4 on P1.3: -  On is logic 0, off is logic 1:
;                   SW4  | SW3  | SW2  |
;                   P1.3 | P1.2 | P1.1 | Function
;                   -------------------------------------------------
;                   On   | On   | On   | Reset Seconds
;                   On   | On   | Off  | Set Minute
;                   On   | Off  | On   | Set Hour
;                   On   | Off  | Off  | Set Date
;                   Off  | On   | On   | Set Month
;                   Off  | On   | Off  | Set Year
;                   Off  | Off  | On   | Set Alarm Hour
;                   Off  | Off  | Off  | Set Alarm Minute
;    SW5 on P1.4: On (logic 0): 12h clock. Off (logic 1): 24h clock.
```

```
; DIP switch SW6 on T1:
;   SW6 switches between 50 and 60Hz: On (logic 0): 50Hz. Off (logic 1): 60Hz.

; External pushbutton on T0:
; - When in clock set mode (P1.0==0), push the pushbutton (making T0 logic 0),
;   to advance the counter being set.
; - When in normal mode (P1.0==1), holding the pushbutton pressed for 1 second
;   will toggle the alarm on and off. Also, pushing the pushbutton while the
;   alarm is sounding will silence the alarm.

; Alarm output:
;   P1.5 is the alarm output. When pulled logic low, the alarm sounds.


; ---------------------------------- DEFINITIONS -----------------------------


; Birthday:
.equ eBrthD    #0x16    ; 22nd. of
.equ eBrthM    #0x04    ; April
.equ eBrthY    #0xFA    ; 2's complement of last two digits of birth year (06).

; 7-segment display definitions:
;    --A--
;   |     |
;   F     B
;   |     |
;    --G--
;   |     |
;   E     C
;   |     |
;    --D--
;                         ABCDEFG0
.equ eBlank    #0x00    ; 00000000
.equ eNum0     #0xFC    ; 11111100
.equ eNum1     #0x60    ; 01100000
.equ eNum2     #0xDA    ; 11011010
.equ eNum3     #0xF2    ; 11110010
.equ eNum4     #0x66    ; 01100110
.equ eNum5     #0xB6    ; 10110110
.equ eNum6     #0xBE    ; 10111110
.equ eNum7     #0xE0    ; 11100000
.equ eNum8     #0xFE    ; 11111110
.equ eNum9     #0xF6    ; 11110110
.equ eLetA     #0xEE    ; 11101110
.equ eLetB     #0x3E    ; 00111110
.equ eLetD     #0x7A    ; 01111010
.equ eLetHC    #0x6E    ; 01101110
.equ eLetHL    #0x2E    ; 00101110
.equ eLetL     #0x1C    ; 00011100
.equ eLetI     #0x08    ; 00001000
.equ eLetP     #0xCE    ; 11001110
.equ eLetR     #0x0A    ; 00001010
.equ eLetT     #0x1E    ; 00011110
.equ eLetY     #0x76    ; 01110110

; 7-segment display definitions for Clock Recovery State Machine display:
;                         ABCDEFG0
.equ eCRStU    #0x7C    ; 01111100: U, displayed while bSync has not been set.
.equ eCRStL    #0x1C    ; 00011100: L, displayed when !INT input is stuck low.
```

```
.equ eCRStH    #0x6E    ; 01101110: H, displayed when !INT input is stuck high.
.equ eCREgE    #0x80    ; 10000000: Displayed for early rising edge on !INT.
.equ eCREgC    #0x02    ; 00000010: Displayed for on-time rising edge on !INT.
.equ eCREgL    #0x10    ; 00010000: Displayed for late rising edge on !INT.

; RAM variables
.equ bSec      #0x20    ; Seconds counter.
.equ bMin      #0x21    ; Minutes counter.
.equ bHour     #0x22    ; Hour counter.
.equ bDate     #0x23    ; Date counter.
.equ bMonth    #0x24    ; Month counter.
.equ bYear     #0x25    ; Year counter.
.equ bAlmHr    #0x26    ; Alarm hour.
.equ bAlmMn    #0x27    ; Alarm minute.

.equ bSync     #0x28    ; Character to show in order to display the Clock
                        ; Recovery State Machine state where !INT input
                        ; rising edge was found.
.equ bStMSt    #0x29    ; Main State Machine State Counter.
.equ bDbnce    #0x2A    ; Stores the value of the Clock Tick Counter (R1) the
                        ; moment the button on T0 is pressed.
.equ bAlStM    #0x2B    ; Alarm State Machine Counter.


; -------------------------- RESET / INTERRUPT VECTORS ----------------------


.org 0x0000                     ; RESET vector.
  JMP lMain

.org 0x0003                     ; External interrupt vector.
  RETR

.org 0x0007                     ; Timer interrupt vector.
  RETR


; --------------------------- MAIN PROGRAM ROUTINE ---------------------------


lMain:
; Main program routine.
  ; Initialization:
  ; Select Bank 0 registers:
    SEL RB0

  ; Initialize RAM variables:
    CLR A
    MOV R0, bSec
    MOV @R0, A                  ; Initialize bSec=0.
    MOV R0, bMin
    MOV @R0, A                  ; Initialize bMin=0.
    MOV R0, bAlmMn
    MOV @R0, A                  ; Initialize bAlmMn=0.
    MOV R0, bHour
    MOV @R0, A                  ; Initialize bHour=0.
    MOV R0, bAlmHr
    MOV @R0, A                  ; Initialize bAlmHr=0.
    INC A
    MOV R0, bDate
```

```
    MOV @R0, A                  ; Initialize bDate=1.
    MOV R0, bMonth
    MOV @R0, A                  ; Initialize bMonth=1.
    MOV R0, bYear
    MOV @R0, #0x0F              ; Initialize bYear=15.

    MOV R0, bSync
    MOV @R0, eCRStU             ; Initialize bSync to "Undefined".

    MOV R0, bStMSt              ; Reset Main State Machine
    MOV @R0, #0x07              ; (start at 7, to be later incremented to 0).

    CLR A
    MOV R0, bDbnce
    MOV @R0, A                  ; Initialize bDbnce=0 (arbitrary value).
    MOV R0, bAlStM
    MOV @R0, A                  ; Initialize Alarm on-off State Machine Counter
                                ; to 0.
  ; Initialize flags:
  ; The Clock Recovery State Machine flags R7.0 to R7.3 are set to 0.
  ; The Alarm flags R7.4 and R7.5 are set to 0.
    MOV R7, #0x00

  ; Clock Recovery State Machine initialization:
    ANL P1, #0xBF               ; Make P1.6 low to disable SN75176's output.
    MOV R6, #0x00               ; Reset the Clock Recovery State Machine.
  ; Initialise the Clock Tick Counter (R1):
    MOV R1, #0x3C               ; Start with R1=60 (for 60Hz).
    JT1 l60HzC                  ; If T1 is high, leave R1 at 60.
    MOV R1, #0x32               ; T1 was low, so start with R1=50 (for 50Hz).
  l60HzC:


  ; Start the State Machine Timer, which runs the State Machines.
  ; The crystal frequency is 4.608MHz, so the timer tick is
  ; 1/4.608MHz*3*5*32=104usec. If I set the timer to overflow every 16 ticks,
  ; then I will have an overflow every about 1,67msec. This is the period I
  ; found the original software used for display column multiplexing.
  ; Note that between each timer overflow, 32*16=512 instruction cycles can
  ; can be processed.
    MOV A, #0xF0                ; (256-16, for a timer count of 16).
    MOV T, A
    STRT T

  lELoop:
  ; Event loop:
  ; Has the State Machine Timer timed out?
    JTF lStChg                  ; Yes, so enter Main State Machine.
    JMP lELoop                  ; No, so do noting.

  lStChg:
  ; The State Machine Timer has timed out.
  ; Restart the timer:
    MOV A, #0xF0
    MOV T, A


; ----------------- CLOCK RECOVERY STATE MACHINE ROUTINES ---------------------
```

```
; The Clock Recovery State Machine has 11 states (60Hz) or 13 states (50Hz).
; This is clocked by the State Machine Timer, thus separating each state by
; 1,67msec.
; Within the 11 or 13 states of this state machine, the following are active:
; With t=1/60Hz (if T1==0) or t=1/50Hz (if T1==1),
; - lCREgE would catch an early rising edge on !INT (i.e. at less than t).
; - lCREgC would catch a rising edge on !INT occurring at time==t.
; - lCREgL would catch a late rising edge on !INT (i.e. at more than t).
; At state lCRPrv, the State Machine starts looking for a logic 0 on the
; !INT input. When this is found (if not in lCRPrv then in lCREgE or
; lCREgC), it sets flag R7.0 to 0.
; The State Machine then starts looking for a logic 1 on the !INT input.
; When this is found, a rising edge on the !INT input has been detected,
; so a Clock Tick is provided and the State Machine is reset to state 0.
; If no rising edge on !INT is detected until the final state (lCREgL),
; the State Machine assumes we have lost mains power. It sets flag R7.1
; and resets the State Machine to State 0. In the new Clock Recovery State
; Machine cycle, state lCREgC checks whether R7.1 had been set. In this
; case, it will provide a "fill-in" Clock Tick to compensate for the
; earlier missing rising edge on !INT.
; Variable bSync is set to the state in which the rising edge was found.
    MOV A, R6                   ; Increment the Clock Recovery State Counter
    INC A                       ; (R6) to the next state.
    ANL A, #0x0F                ; If R6 exceeds 15, reset it to 0. Note that
                                ; this should never happen, it is just put
                                ; here as a precautionary measure.
    MOV R6, A

    JT1 l60HzA                  ; If T1 is high, we are running at 60Hz.

  ; T1 was low, so use state machine for 50Hz.
    ADD A, #lSM50Hz             ; 50Hz Cl. Rec. St. Machine jump table origin.
    JMPP @A                     ; Select the appropriate routine.
  lSM50Hz:                      ; 50Hz Clock Recovery State Machine jump table:
    .db #lCREnd                    ;   State  0.  0,00msec. Unreachable.
    .db #lCREnd                    ;   State  1.  1,67msec. Do nothing.
    .db #lCREnd                    ;   State  2.  3,33msec. Do nothing.
    .db #lCREnd                    ;   State  3.  5,00msec. Do nothing.
    .db #lCREnd                    ;   State  4.  6,67msec. Do nothing.
    .db #lCREnd                    ;   State  5.  8,33msec. Do nothing.
    .db #lCREnd                    ;   State  6. 10,00msec. Do nothing.
    .db #lCREnd                    ;   State  7. 11,67msec. Do nothing.
    .db #lCREnd                    ;   State  8. 13,33msec. Do nothing.
    .db #lCREnd                    ;   State  9. 15,00msec. Do nothing.
    .db #lCRPrv                    ;   State 10. 16,67msec. Store previous
state.
    .db #lCREgE                    ;   State 11. 18,33msec. Look for early
edge.
    .db #lCREgC                    ;   State 12. 20,00msec. Look for correct
edge.
    .db #lCREgL                    ;   State 13. 21,67msec. Look for late edge.
    .db #lCREnd                    ;   State 14. 23,33msec. Unreachable.
    .db #lCREnd                    ;   State 15. 25,00msec. Unreachable.

  l60HzA:
  ; T1 was high, so use state machine for 60Hz.
    ADD A, #lSM60Hz             ; 60Hz Cl. Rec. St. Machine jump table origin.
    JMPP @A                     ; Select the appropriate routine.
  lSM60Hz:                      ; 60Hz Clock Recovery State Machine jump table:
    .db #lCREnd                    ;   State  0.  0,00msec. Unreachable.
```

```
    .db #lCREnd                     ;   State  1.  1,67msec. Do nothing.
    .db #lCREnd                     ;   State  2.  3,33msec. Do nothing.
    .db #lCREnd                     ;   State  3.  5,00msec. Do nothing.
    .db #lCREnd                     ;   State  4.  6,67msec. Do nothing.
    .db #lCREnd                     ;   State  5.  8,33msec. Do nothing.
    .db #lCREnd                     ;   State  6. 10,00msec. Do nothing.
    .db #lCREnd                     ;   State  7. 11,67msec. Do nothing.
    .db #lCRPrv                     ;   State  8. 13,33msec. Store previous
state.
    .db #lCREgE                     ;   State  9. 15,00msec. Look for early
edge.
    .db #lCREgC                     ;   State 10. 16,67msec. Look for correct
edge.
    .db #lCREgL                     ;   State 11. 18,33msec. Look for late edge.
    .db #lCREnd                     ;   State 12. 20,00msec. Unreachable.
    .db #lCREnd                     ;   State 13. 21,67msec. Unreachable.
    .db #lCREnd                     ;   State 14. 23,33msec. Unreachable.
    .db #lCREnd                     ;   State 15. 25,00msec. Unreachable.

  lCRPrv:
  ; Set the "previous !INT state" flag (R7.0) to the current state of !INT,
  ; for use in later comparisons.
    MOV A, R7                   ; Get R7.
    ANL A, #0xFE                ; Start by setting Acc. bit 0 to 0.
    JNI lINT0A                  ; If !INT is logic 0, leave Acc. bit 0 at 0.
    ORL A, #0x01                ; !INT was logic 1, so set Acc. bit 0 to 1.
  lINT0A:
  ; Note: The result (which is to go in R7.0) has been left in the Accumulator.

  ; Store a copy of R7.1 in R7.2, because R7.1 is set to 0 below, while its
  ; value is needed in lCREgC.
    ORL A, #0x04                ; Start by setting Acc. bit 2 to 1.
    JB1 l71E1A                  ; If R7.1==1, leave Acc. bit 2 at 1.
    ANL A, #0xFB                ; R7.1 was 0, so set Acc. bit 2 to 0.
  l71E1A:
  ; Note: The result (which is to go in R7.2) has been left in the Accumulator.

  ; Set R7.1 to 0 (since no rising edge on !INT has been found yet)!
    ANL A, #0xFD                ; Set Acc. bit 1 to 0.
    MOV R7, A                   ; Store R7.
    JMP lCREnd                  ; Finished.

  lCREgE:
  ; Look for early rising edge on !INT0.
  ; - If the previous !INT state (R7.0) was 1, this means !INT had not fallen
  ;   to logic 0. Continue searching for a logic 0 on !INT (really, just
  ;   store the value of !INT in R7.0).
  ; - If the previous value of !INT (i.e. R7.0) is 0, then
  ;   - if the current value of !INT is 0, do nothing (since nothing has
  ;     changed).
  ;   - if the current value of !INT is 1, we have a rising edge on !INT.
  ;     In this case,
  ;     - A Clock Tick is provided,
  ;     - R7.1 is set to 1 to inform the next Clock Recovery State Machine
  ;       cycle that a Clock Tick has been provided,
  ;     - bSync is set to "Early"
  ;     - the Clock Recovery State Machine is reset to state 0.
    MOV A, R7                   ; Get R7.
    JB0 l70E1A                  ; If R7.0==1 go and set R7.0 to !INT.
```

```
  JNI lCREnd                    ; Is !INT still logic 0? If so, do nothing.

                                ; The previous !INT state (R7.0) was logic 0
                                ; and the current !INT state is logic 1, so
                                ; we have a rising edge on !INT:
  CALL lClkTk                     ;   Provide a Clock Tick,

  MOV A, R7
  ORL A, #0x02                  ;   Set R7.1 to 1,
  MOV R7, A

  MOV R0, bSync                 ;   Set bSync to "Early",
  MOV @R0, eCREgE

  MOV R6, #0x00                 ;   Reset the Clock Recovery State Machine,
  JMP lCREnd                    ;   Finished.

l70E1A:                         ; Set R7.0 to !INT. Note: Acc. already is =R7:
  ANL A, #0xFE                  ;   Start by setting Acc. bit 0 to 0.
  JNI lINT0C                    ;   If !INT is logic 0, leave Acc. bit 0 at 0.
  ORL A, #0x01                  ;   !INT was logic 1, so set Acc. bit 0 to 1.
lINT0C:
  MOV R7, A                     ;   Store R7.
  JMP lCREnd                    ;   Finished.

lCREgC:
; Look for "correct" rising edge on !INT0 (i.e. arriving at the expected
; time). The routine is the same as for lCREgE.
  MOV A, R7                     ; Get R7.
  JB0 l70E1B                    ; If R7.0==1 go and set R7.0 to !INT.

  JNI lINT0D                    ; Is !INT still logic 0? If so, do nothing.

                                ; The previous !INT state (R7.0) was logic 0
                                ; and the current !INT state is logic 1, so
                                ; we have a rising edge on !INT:
  CALL lClkTk                     ;   Provide a Clock Tick,

  MOV A, R7
  ORL A, #0x02                  ;   Set R7.1 to 1,
  MOV R7, A

  MOV R0, bSync                 ;   Set bSync to "On-time",
  MOV @R0, eCREgC

  MOV R6, #0x00                 ;   Reset the Clock Recovery State Machine,
  JMP lINT0D                    ;   Finished.

l70E1B:                         ; Set R7.0 to !INT. Note: Acc. already is =R7:
  ANL A, #0xFE                  ;   Start by setting Acc. bit 0 to 0.
  JNI lINT0E                    ;   If !INT is logic 0, leave Acc. bit 0 at 0.
  ORL A, #0x01                  ;   !INT was logic 1, so set Acc. bit 0 to 1.
lINT0E:
  MOV R7, A                     ;   Store R7.
                                ;   Finished.
lINT0D:
; Additionally, if the previous Clock Recovery State Machine cycle had not
; provided a Clock Tick (this is signalled by R7.2==0), then provide a
; "fill-in" Clock tick to compensate, and forcibly reset the Clock Recovery
; State Machine:
```

```
  MOV A, R7                     ; Get R7.
  JB2 lCREnd                    ; If R7.2==1, the previous Clock Recovery State
                                ; Machine cycle had provided a Clock Tick, so
                                ; do nothing.
  CALL lClkTk                     ; Else, provide a "fill-in" Clock Tick,
  MOV R6, #0x00                 ; Reset the Clock Recovery State Machine,
  JMP lCREnd                    ; Finished.

lCREgL:
; Look for late rising edge on !INT0.
; - If the previous !INT state (R7.0) was 1, this means !INT had not fallen
;   to logic 0. It is too late to look for a logic 0 on !INT now; assume
;   that !INT is stuck high.
; - If the previous value of !INT (i.e. R7.0) is 0, then
;   - if the current value of !INT is 0, assume !INT is stuck low.
;   - if the current value of !INT is 1, we have a rising edge on !INT.
;     In this case,
;     - A Clock Tick is provided,
;     - R7.1 is set to 1 to inform the next Clock Recovery State Machine
;       cycle that a Clock Tick has been provided,
;     - bSync is set "Late",
;     - the Clock Recovery State Machine is reset to state 0.
  MOV A, R7                     ; Get R7.
  JB0 l70E1C                    ; If R7.0==1 go and set bSync to "Stuck high".

  JNI lINT0F                    ; Is !INT still logic 0? If so, go and set
                                ; bSync to "Stuck low".

                                ; The previous !INT state (R7.0) was logic 0
                                ; and the current !INT state is logic 1, so
                                ; we have a rising edge on !INT:
  CALL lClkTk                     ;   Provide a Clock Tick,

  MOV A, R7
  ORL A, #0x02                  ;   Set R7.1 to 1,
  MOV R7, A

  MOV R0, bSync                 ;   Set bSync to "Late".
  MOV @R0, eCREgL
  JMP lSkipA

l70E1C:                         ; !INT stuck high: Set bSync to "Stuck high".
  MOV R0, bSync
  MOV @R0, eCRStH
  JMP lSkipA

lINT0F:                         ; !INT stuck low: Set bSync to "Stuck low".
  MOV R0, bSync
  MOV @R0, eCRStL

lSkipA:
  MOV R6, #0x00                 ; Reset the Clock Recovery State Machine.
  JMP lCREnd                    ; Finished.

lCREnd:                         ; Clock Recovery State Machine end.

; Jump to Main State Machine:
  JMP lMStM                     ; A JMP instruction is used because of the
                                ; change in memory page.
```

```
; ---------------------- MAIN STATE MACHINE ROUTINES -------------------------


.org 0x0100                     ; Helps avoid page break within routine.


lMStM:
; The Main State Machine has eight states, which it cycles through using
; memory variable bStMSt as the State Counter.
; The states are:
; - 0: Writes leftmost display column segments.
; - 1: Strobes P2.7 low to switch on leftmost display column,
;      then deals with seconds increment.
; - 2: Writes 2nd. from left display column segments.
; - 3: Strobes P2.6 low to switch on 2nd. from left display column,
;      then runs the T0 button Alarm on-off State Machine and the alarm
;      engine.
; - 4: Writes 2nd. from right display column segments.
; - 5: Strobes P2.5 low to switch on 2nd. from right display column.
; - 6: Writes rightmost display column.
; - 7: Strobes P2.4 low to switch on rightmost display column.
; The Main State Machine then returns to 0.
    MOV R0, bStMSt              ; Increment the Main State Machine State
    MOV A, @R0                  ; Counter to the next State.
    INC A
    ANL A, #0x07                ; If State==8, then make State=0.
    MOV @R0, A
  ; Jump to the routine for the State:
    ADD A, #lStJmp - 0x0100     ; Main State Machine jump table origin.
    JMPP @A                     ; Select the appropriate routine.
  lStJmp:                       ; Main State Machine jump table:
    .db #lStLc0 - 0x0100
    .db #lStLc1 - 0x0100
    .db #lStLc2 - 0x0100
    .db #lStLc3 - 0x0100
    .db #lStLc4 - 0x0100
    .db #lStLc5 - 0x0100
    .db #lStLc6 - 0x0100
    .db #lStLc7 - 0x0100
  ; "Real" jumps are necessary due to the possibility of a change
  ; in memory page:
  lStLc0:
    JMP lSt0
  lStLc1:
    JMP lSt1
  lStLc2:
    JMP lSt2
  lStLc3:
    JMP lSt3
  lStLc4:
    JMP lSt4
  lStLc5:
    JMP lSt5
  lStLc6:
    JMP lSt6
  lStLc7:
    JMP lSt7
```

```
lSt0:
; State 0: Write segments for leftmost display column.
  ; P2.7->P2.4 all logic high:
    ORL P2, #0xF0               ; Upper 4 bits.

  ; Write leftmost display column:
    CALL lDWhat                   ; Find out what is to be displayed.
    ADD A, #lS0Jmp - 0x0100     ; State 0 jump table origin.
    JMPP @A                     ; Select the appropriate action.
  lS0Jmp:
    .db #lS0Tme         - 0x0100
    .db #lS0Alm - 0x0100
    .db #lS0HBr - 0x0100

  lS0Tme:
  ; Display the time (leftmost column).
    MOV R0, bHour               ; Get hour.
    IN A, P1                    ; Get P1 state.
    JB4 l24hA                   ; Is P1.4 logic 1 (SW5 Off, 24h clock)?
    MOV A, @R0                  ; - No (12h clock). Get hour,
    CALL l24H12                   ;   Convert hours from 24h to 12h,
    JMP l24hB                   ;   Done.
  l24hA:
    MOV A, @R0                  ; - Yes (24h clock). Get hour.
  l24hB:
    CALL l7SegT                   ; Convert to seven segment (tens).
    MOV R2, A                   ; Leftmost top character.

    MOV R0, bSec                ; Get second.
    MOV A, @R0
    CALL l7SegT                   ; Convert to seven segment (tens).
    MOV R3, A                   ; Leftmost 2nd. from top character.

    MOV R0, bDate               ; Get date.
    MOV A, @R0
    CALL l7SegT                   ; Convert to seven segment (tens).
    MOV R4, A                   ; Leftmost 2nd. from bottom character.

    MOV R5, eNum2               ; Leftmost bottom character ("2").
    JMP lS0Fin                  ; Finished.

  lS0Alm:
  ; Display the alarm time (leftmost column).
    MOV R0, bAlmHr              ; Get alarm hour.
    IN A, P1                    ; Get P1 state.
    JB4 l24hI                   ; Is P1.4 logic 1 (SW5 Off, 24h clock)?
    MOV A, @R0                  ; - No (12h clock). Get alarm hour,
    CALL l24H12                   ;   Convert hours from 24h to 12h,
    JMP l24hF                   ;   Done.
  l24hI:
    MOV A, @R0                  ; - Yes (24h clock). Get alarm hour.
  l24hF:
    CALL l7SegT                   ; Convert to seven segment (tens).
    MOV R2, A                   ; Leftmost top character.

    MOV R3, eBlank              ; Leftmost 2nd. from top character.
    MOV R4, eBlank              ; Leftmost 2nd. from bottom character.
    MOV R5, eBlank              ; Leftmost bottom character.
    JMP lS0Fin                  ; Finished.
```

```
  lS0HBr:
  ; Display Happy Birthday (leftmost column).
    MOV R2, eLetHC              ; Leftmost top character.
    MOV R3, eLetY               ; Leftmost 2nd. from top character.
    MOV R4, eLetB               ; Leftmost 2nd. from bottom character.
    MOV R5, eLetHL              ; Leftmost bottom character.
    JMP lS0Fin                  ; Finished.

  lS0Fin:
  ; Write leftmost column.
    CALL lsWCol                    ; Write column.
    JMP lELoop                  ; Return.


lSt1:
; State 1:
  ; Strobe P2.7 logic low to switch on leftmost display column:
    ANL P2, #0x7F

  ; Deal with the seconds increment flag:
    MOV A, R7                   ; Get R7.
    JB3 lSecA                   ; Is R7.3==1 (i.e. Has one second elapsed)?
    JMP lELoop                  ; - No, so do nothing (Return).

  lSecA:
  ; One second has elapsed.
    IN A, P1                    ; Get P1 state.
    JB0 lNCStA                  ; Is P1.0 (Not Clock Set)==1?
                                ; - No, so check T0 (the pushbutton).
    JT0 lEndA                   ;     If it is logic 1 (unpressed), do nothing.
    CALL lStClk                    ;     If it is logic 0 (pressed), set clock.
    JMP lEndA

  lNCStA:
    CALL lIncTm                    ; - Yes, so increment time.

  lEndA:
    MOV A, R7
    ANL A, #0xF7                ; Clear the Seconds Increment Flag (R7.3).
    MOV R7, A
    JMP lELoop                  ; Return.


lSt2:
; State 2: Write segments for 2nd. from left display column.
  ; P2.7->P2.4 all logic high:
    ORL P2, #0xF0               ; Upper 4 bits.

  ; Write 2nd. from left display column:
    CALL lDWhat                    ; Find out what is to be displayed.
    ADD A, #lS2Jmp - 0x0100     ; State 2 jump table origin.
    JMPP @A                     ; Select the appropriate action.
  lS2Jmp:
    .db #lS2Tme         - 0x0100
    .db #lS2Alm - 0x0100
    .db #lS2HBr - 0x0100

  lS2Tme:
  ; Display the time (2nd. from left column).
    MOV R0, bHour               ; Get hour.
```

```
    IN A, P1                    ; Get P1 state.
    JB4 l24hC                   ; Is P1.4 logic 1 (SW5 Off, 24h clock)?
    MOV A, @R0                  ; - No (12h clock). Get hour,
    CALL l24H12                    ;   Convert hours from 24h to 12h,
    JMP l24hD                   ;   Done.
  l24hC:
    MOV A, @R0                  ; - Yes (24h clock). Get hour.
  l24hD:
    CALL l7SegU                    ; Convert to seven segment (units).
    MOV R2, A                   ; 2nd. from left, top character.

    MOV R0, bSec                ; Get second.
    MOV A, @R0
    CALL l7SegU                    ; Convert to seven segment (units).
    MOV R3, A                   ; 2nd. from left, 2nd. from top character.

    MOV R0, bDate               ; Get date.
    MOV A, @R0
    CALL l7SegU                    ; Convert to seven segment (units).
    MOV R4, A                   ; 2nd. from left, 2nd. from bottom character.

    MOV R5, eNum0               ; 2nd. from left, bottom character ("0").
    JMP lS2Fin                  ; Finished.

  lS2Alm:
  ; Display alarm time (2nd. from left column).
    MOV R0, bAlmHr              ; Get alarm hour.
    IN A, P1                    ; Get P1 state.
    JB4 l24hJ                   ; Is P1.4 logic 1 (SW5 Off, 24h clock)?
    MOV A, @R0                  ; - No (12h clock). Get alarm hour,
    CALL l24H12                    ;   Convert hours from 24h to 12h,
    JMP l24hG                   ;   Done.
  l24hJ:
    MOV A, @R0                  ; - Yes (24h clock). Get alarm hour.
  l24hG:
    CALL l7SegU                    ; Convert to seven segment (units).
    MOV R2, A                   ; 2nd. from left, top character.

    MOV R3, eBlank              ; 2nd. from left, 2nd. from top character.
    MOV R4, eBlank              ; 2nd. from left, 2nd. from bottom character.
    MOV R5, eBlank              ; 2nd. from left, bottom character.
    JMP lS2Fin                  ; Finished.

  lS2HBr:
  ; Display Happy Birthday (2nd. from left column).
    MOV R2, eLetA               ; 2nd. from left, top character.
    MOV R3, eBlank              ; 2nd. from left, 2nd. from top character.
    MOV R4, eLetI               ; 2nd. from left, 2nd. from bottom character.
    MOV R5, eLetD               ; 2nd. from left, bottom character.
    JMP lS2Fin                  ; Finished.

  lS2Fin:
  ; Write 2nd. from left column.
    CALL lsWCol                    ; Write column.
    JMP lELoop                  ; Return.


.org 0x0200                     ; Helps avoid page break within routine.
```

```
lSt3:
; State 3:
  ; Strobe P2.6 logic low to switch on 2nd. from left display column:
    ANL P2, #0xBF

  ; Alarm on-off State Machine:
  ; A three-state State Machine is used to determine whether the button on T0
  ; has been held pressed for 1 second, to toggle Alarm on-off.
  ; - In State 0, the State Machine checks whether the button has been pressed.
  ;   If it has, the Clock Tick Counter (R1) is stored in bDbnce and we move to
  ;   State 1.
  ; - In State 1, the State Machine waits until the Clock Tick Counter advances
  ;   beyond its current value (==bDbnce).
  ; - In State 2, the State Machine knows that the Clock Tick Counter has
  ;   advanced beyond bDbnce. The Clock Tick Counter cycles every 1 second, so
  ;   when the Clock Tick Counter is again ==bDbnce, we know that 1 second has
  ;   elapsed. If the button is still pressed, then Alarm on-off is toggled.
    MOV R0, bAlStM              ; Get the Alarm on-off State Machine counter.
    MOV A, @R0
    ADD A, #lAlStJ - 0x0200     ; Alarm on-off State Machine jump table origin.
    JMPP @A                     ; Select the appropriate routine.
  lAlStJ:                       ; Alarm on-off State Machine jump table:
    .db #lAlSM0 - 0x0200
    .db #lAlSM1 - 0x0200
    .db #lAlSM2 - 0x0200

  lAlSM0:
    ; State 0: Check button on T0:
    JT0 lASEnd                  ; Is T0==1 (button open)? If so, do nothing.
    IN A, P1                    ; Get P1 state.
    JB0 lNCStB                  ; Is P1.0 (Not Clock Set)==1?
    JMP lASEnd                  ; - No, so the clock is being set; the button
                                ;   is used for setting the clock, not for
                                ;   alarm functions. So, do nothing.
  lNCStB:                       ; - Yes, so the clock is not being set.
    MOV A, R1                   ;   Store the Clock Tick Counter (R1)
    MOV R0, bDbnce              ;   in bDbnce,
    MOV @R0, A
    MOV R0, bAlStM              ;   Proceed to Alarm on-off State Machine
    MOV @R0, #0x01              ;   State 1,
    JMP lASEnd                  ;   Finished.

  lAlSM1:
    ; State 1: Wait for Clock Tick:
    MOV R0, bDbnce              ; Is R1==bDbnce? If so, we are still on the
    MOV A, R1                   ; Clock Tick as when the button was first
    XRL A, @R0                  ; pressed:
    JZ lASEnd                   ; - Yes, so do nothing.
    MOV R0, bAlStM              ; - No, so proceed to Alarm on-off State
    MOV @R0, #0x02              ;   Machine State 2.
    JMP lASEnd                  ;   Finished.

  lAlSM2:
    ; State 2: Wait for 1 second:
    JT0 lASMRs                  ; Is T0==1 (button open)? If so, go and reset
                                ; the Alarm on-off State Machine.
    IN A, P1                    ; Get P1 state.
    JB0 lNCStD                  ; Is P1.0 (Not Clock Set)==1?
    JMP lASMRs                  ; - No, so the clock is being set; go and reset
                                ;   the Alarm on-off State Machine.
```

```
lNCStD:                         ; - Yes, so the clock is not being set.
  MOV R0, bDbnce                ; Is R1==bDbnce?
  MOV A, R1
  XRL A, @R0
  JNZ lASEnd                    ; - No, so do nothing.
                                ; - Yes, so 1 second has elapsed, the
                                ;   button is still pressed and the clock is
                                ;   not being set.
  MOV A, R7                     ;   Toggle R7.4 (the Alarm On flag):
  JB4 lAlOnB                    ;   Is R7.4==1?
  ORL A, #0x10                  ;   - No, so set R7.4=1,
  MOV R7,A                      ;     and store R7.
  JMP lASMRs
lAlOnB:                         ;   - Yes,
  ANL A, #0xEF                  ;     so set R7.4=0,
  MOV R7,A                      ;     and store R7.

lASMRs:                         ; Reset Alarm on-off State Machine to State 0:
  MOV R0, bAlStM
  MOV @R0, #0x00

lASEnd:                         ; Alarm on-off State Machine finished.

; Alarm engine:
  MOV A, R7                     ; Get R7.
  JB4 lAlOnA                    ; Is R7.4==1 (Alarm On)?
  JMP lNoAlm                    ; - No, so do not sound alarm.

lAlOnA:                         ; - Yes, the alarm is on.
  MOV R0, bHour                 ;   Is bHour==bAlmHr?
  MOV A, @R0
  MOV R0, bAlmHr
  XRL A, @R0
  JNZ lNoAlm                    ;   - No, so do not sound alarm.
                                ;   - Yes.
  MOV R0, bMin                  ;     Is bMin==bAlmMn?
  MOV A, @R0
  MOV R0, bAlmMn
  XRL A, @R0
  JNZ lNoAlm                    ;     - No, so do not sound alarm.
                                ;     - Yes.
  JT0 lButOA                    ;       Is T0==1 (button open)?
                                ;       - No, so the button is pushed while the
                                ;         alarm is ringing. Silence the alarm:
  MOV A, R7                     ;         Get R7.
  ORL A, #0x20                  ;         Set R7.5 (Alarm Silence)=1.
  MOV R7, A                     ;         Store R7.
lButOA:                         ;  - Yes, T0==1, so the button is open.

  MOV A, R7                     ;       Get R7.
  JB5 lNoSnd                    ;  Is R7.5==1 (Alarm Silence)?
                                ;       - No, so it is time to sound the alarm
                                ;         and the alarm is not silenced.
                                ;         Alternately start and stop the sound
                                ;         every second:
   MOV R0, bSec                 ;         Get the seconds counter bSec.
   MOV A, @R0
   JB0 lNoSnd                   ;         Is bSec an odd number?
                                ;         - If yes, stop the sound.
   ANL P1, #0xDF                ;         - No. Set P1.5=0 to make a sound,
```

```
     JMP lELoop                    ;           Return.

  lNoAlm:                          ; Either the alarm is turned off, or today's
                                   ; alarm is finished. Set R7.5 (Alarm Silence)=0
                                   ; in preparation for tomorrow's alarm:
    MOV A, R7                      ; Get R7,
    ANL A, #0xDF                   ; Set R7.5 (Alarm Silence)=0,
    MOV R7, A                      ; Store R7.
  lNoSnd:
    ORL P1, #0x20                  ; Set P1.5=1 to stop the sound.
    JMP lELoop                     ; Return.


lSt4:
; State 4: Write segments for 2nd. from right display column.
  ; P2.7->P2.4 all logic high:
    ORL P2, #0xF0                  ; Upper 4 bits.

  ; Write 2nd. from right display column:
    CALL lDWhat                     ; Find out what is to be displayed.
    ADD A, #lS4Jmp - 0x0200        ; State 4 jump table origin.
    JMPP @A                        ; Select the appropriate action.
  lS4Jmp:
    .db #lS4Tme         - 0x0200
    .db #lS4Alm - 0x0200
    .db #lS4HBr - 0x0200

  lS4Tme:
  ; Display the time (2nd. from right column).
    MOV R0, bMin                   ; Get minute.
    MOV A, @R0
    CALL l7SegT                     ; Convert to seven segment (tens).
    MOV R2, A                      ; 2nd. from right, top character.

  ; 2nd. from right, 2nd. from top character (R3):
  ; If P1.4 is logic 1 (SW5 Off, 24h clock): R3=eBlank.
  ; If P1.4 is logic 0 (SW5 On,  12h clock): R3=eLetA (AM) or eLetP (PM).
    MOV R3, eBlank                 ; Make R3=eBlank to begin with.
    IN A, P1                       ; Get P1 state.
    JB4 l24hE                      ; Is P1.4 logic 1?
                                   ; - No (12h clock), so write AM or PM:
    MOV R0, bHour                  ;   Get hour.
    MOV A, @R0
    ADD A, #0xF4                   ;   Set carry flag if bHour>=12.
    MOV R3, eLetA                  ;   Let's start with R3=eLetA.
    JNC lAMB                       ;   Is bHour>=12? (use carry flag set above).
    MOV R3, eLetP                  ;   - Yes it is, so make R3=eLetP.
  lAMB:                             ;   - No it isn't, so leave R3=eLetA.
  l24hE:                           ; - Yes (24h clock), so leave R3=eBlank.

    MOV R0, bMonth                 ; Get month.
    MOV A, @R0
    CALL l7SegT                     ; Convert to seven segment (tens).
    MOV R4, A                      ; 2nd. from right, 2nd. from bottom character.

    MOV R0, bYear                  ; Get year.
    MOV A, @R0
    CALL l7SegT                     ; Convert to seven segment (tens).
    MOV R5, A                      ; 2nd. from right, bottom character.
    JMP lS4Fin                     ; Finished.
```

```
  lS4Alm:
  ; Display alarm time (2nd. from right column).
    MOV R0, bAlmMn               ; Get alarm minute.
    MOV A, @R0
    CALL l7SegT                     ; Convert to seven segment (tens).
    MOV R2, A                    ; 2nd. from right, top character.

  ; 2nd. from right, 2nd. from top character (R3):
  ; If P1.4 is logic 1 (SW5 Off, 24h clock): R3=eBlank.
  ; If P1.4 is logic 0 (SW5 On,  12h clock): R3=eLetA (AM) or eLetP (PM).
    MOV R3, eBlank               ; Make R3=eBlank to begin with.
    IN A, P1                     ; Get P1 state.
    JB4 l24hH                    ; Is P1.4 logic 1?
                                 ; - No (12h clock), so write AM or PM:
    MOV R0, bAlmHr               ;   Get alarm hour.
    MOV A, @R0
    ADD A, #0xF4                 ;   Set carry flag if bAlmHr>=12.
    MOV R3, eLetA                ;   Let's start with R3=eLetA.
    JNC lAMC                     ;   Is bHour>=12? (use carry flag set above).
    MOV R3, eLetP                ;   - Yes it is, so make R3=eLetP.
  lAMC:                             ;   - No it isn't, so leave R3=eLetA.
  l24hH:                         ; - Yes (24h clock), so leave R3=eBlank.

    MOV R4, eBlank               ; 2nd. from right, 2nd. from bottom character.
    MOV R5, eBlank               ; 2nd. from right, bottom character.
    JMP lS4Fin                   ; Finished.

  lS4HBr:
  ; Display Happy Birthday (2nd. from right column).
    MOV R2, eLetP                ; 2nd. from right, top character.

    MOV R0, bYear                ; Get current year.
    MOV A, @R0
    ADD A, eBrthY                ; Subtract birth year.
    CALL l7SegT                     ; Convert to seven segment (tens).
    MOV R3, A                    ; 2nd. from right, 2nd. from top character.

    MOV R4, eLetR                ; 2nd. from right, 2nd. from bottom character.
    MOV R5, eLetA                ; 2nd. from right, bottom character.
    JMP lS4Fin                   ; Finished.

  lS4Fin:
  ; Write 2nd. from right column.
    CALL lsWCol                     ; Write column.
    JMP lELoop                   ; Return.


.org 0x0300                      ; Helps avoid page break within subroutine.


lSt5:
; State 5:
  ; Strobe P2.5 logic low to switch on 2nd. from right display column:
    ANL P2, #0xDF
    JMP lELoop                   ; Return.


lSt6:
; State 6: Write segments for rightmost display column.
```

```
; P2.7->P2.4 all logic high:
  ORL P2, #0xF0                ; Upper 4 bits.

; Write rightmost display column:
  CALL lDWhat                    ; Find out what is to be displayed.
  ADD A, #lS6Jmp - 0x0300      ; State 6 jump table origin.
  JMPP @A                      ; Select the appropriate action.
lS6Jmp:
  .db #lS6Tme         - 0x0300
  .db #lS6Alm - 0x0300
  .db #lS6HBr - 0x0300

lS6Tme:
; Display the time (rightmost column).
  MOV R0, bMin                 ; Get minute.
  MOV A, @R0
  CALL l7SegU                    ; Convert to seven segment (units).
  MOV R2, A                    ; Rightmost top character.

                               ; Rightmost 2nd. from top character:
  MOV R3, eLetL                ; Begin with the letter "L" ("Alarm On").
  MOV A, R7                    ; Get R7.
  JB4 lAlOnC                   ; Is R7.4==1?
  MOV R3, eBlank               ; - No, so R3=blank ("Alarm Off").
lAlOnC:                        ; - Yes, so leave R3=letter "L".

  MOV R0, bMonth               ; Get month.
  MOV A, @R0
  CALL l7SegU                    ; Convert to seven segment (units).
  MOV R4, A                    ; Rightmost 2nd. from bottom character.

  MOV R0, bYear                ; Get year.
  MOV A, @R0
  CALL l7SegU                    ; Convert to seven segment (units).
  MOV R5, A                    ; Rightmost bottom character.
  JMP lS6Fin                   ; Finished.

lS6Alm:
; Display alarm time (rightmost column).
  MOV R0, bAlmMn               ; Get alarm minute.
  MOV A, @R0
  CALL l7SegU                    ; Convert to seven segment (units).
  MOV R2, A                    ; Rightmost top character.

                               ; Rightmost 2nd. from top character:
  MOV R3, eLetL                ; Begin with the letter "L" ("Alarm On").
  MOV A, R7                    ; Get R7.
  JB4 lAlOnD                   ; Is R7.4==1?
  MOV R3, eBlank               ; - No, so R3=blank ("Alarm Off").
lAlOnD:                        ; - Yes, so leave R3=letter "L".

  MOV R4, eBlank               ; Rightmost 2nd. from bottom character.

  MOV R0, bSync                ; Get Clock Recovery State Machine status
  MOV A, @R0                   ; display character.
  MOV R5, A                    ; Rightmost bottom character.
  JMP lS6Fin                   ; Finished.

lS6HBr:
; Display Happy Birthday (rightmost column).
```

```
    MOV R2, eLetP               ; Rightmost top character.

    MOV R0, bYear               ; Get current year.
    MOV A, @R0
    ADD A, eBrthY               ; Subtract birth year.
    CALL l7SegU                   ; Convert to seven segment (units).
    MOV R3, A                   ; Rightmost 2nd. from top character.

    MOV R4, eLetT               ; Rightmost 2nd. from bottom character.
    MOV R5, eLetY               ; Rightmost bottom character.
    JMP lS6Fin                  ; Finished.

  lS6Fin:
  ; Write rightmost column.
    CALL lsWCol                   ; Write column.
    JMP lELoop                  ; Return.


lSt7:
; State 7:
  ; Strobe P2.4 logic low to switch on rightmost display column.
    ANL P2, #0xEF
    JMP lELoop                  ; Return.



; ---------------------------- GENERAL SUBROUTINES ----------------------------


.org 0x0400                     ; Helps avoid page break within subroutine.


lClkTk:
; Subroutine ClockTick:
; Provides a "Clock Tick". It decrements the Clock Tick Counter R1.
; - If R1 has not reached zero, it just returns.
; - If R1 has reached zero, the Seconds Increment Flag (R7.3) is set to inform
;   the Main State Machine that one second has elapsed.
;   Then, R1 is reset to 60 (for 60Hz) or 50 (for 50Hz).
; Affects R1.
; Affects bit 3 of R7.
    DJNZ R1, lCTEnd             ; Decrement the Clock Tick Counter. If it
                                ; has not reached zero, we are finished.

    MOV A, R7                   ; The Clock Tick Counter has reached zero, so:
    ORL A, #0x08                ; Set the Seconds Increment Flag (R7.3),
    MOV R7, A
                                ; Re-Initialise the Clock Tick Counter:
    MOV R1, #0x3C               ; Start with R1=60 (for 60Hz).
    JT1 lCTEnd                  ; If T1 is high, leave R1 at 60.
    MOV R1, #0x32               ; T1 was low, so set R1=50 (for 50Hz).
  lCTEnd:
    RET


lDWhat:
; Subroutine DisplayWhat:
; Determines what is to be displayed and returns the result in the accumulator:
;   Accumulator=0 if the time is to be displayed.
;   Accumulator=1 if the alarm time is to be displayed.
```

```
;   Accumulator=2 if "Happy Birthday" is to be displayed.
; Affects R0.
    IN A, P1                    ; Get P1 state.
    ANL A, #0x0D                ; 00001101 (keep only P1.0, P1.2, P1.3).
    XRL A, #0x0C                ; 00001100: Are P1.0==0, P1.2==1, P1.3==1?
                                ; (i.e. is the alarm time being set)?
    JZ lDsAlm                   ; If so, go and display the alarm time.

    IN A, P1                    ; Get P1 state.
    JB0 lNCStC                  ; Is P1.0==0 (i.e. is the clock being set)?
    JMP lDsTme                  ; - Yes, so go and display the time.
  lNCStC:                       ; - No, the clock is not being set.

    JNT0 lDsAlm                    ; Is the button on T0 pushed (T0==0)? If so,
go
                                ; and display the alarm time.

    MOV R0, bDate               ; Get the date.
    MOV A, @R0
    XRL A, eBrthD               ; Is the date equal to birth date?
    JNZ lDsTme                  ; If not, go and display the time.
    MOV R0, bMonth              ; Get the month.
    MOV A, @R0
    XRL A, eBrthM               ; Is the month equal to birth month?
    JNZ lDsTme                  ; If not, go and display the time.
  ; The clock or alarm are not being set, the button on T0 is not pushed and
  ; the date and month are == birthday. Display Happy Birthday:
    MOV A, #0x02
    RET

  lDsAlm:                       ; Display the alarm time.
    MOV A, #0x01
    RET

  lDsTme:                       ; Display the time.
    CLR A
    RET


lsWCol:
; Subroutine WriteColumn:
; Uses subroutine lsWSeg to write the segment information to
; the displays in the column which is to be addressed with P2.7->P2.4 after
; this function returns.
; The data to be written to the display column should be provided as follows:
;   R2: Character to be written to the top display.
;   R3: Character to be written to the display 2nd. from top.
;   R4: Character to be written to the display 2nd. from bottom.
;   R5: Character to be written to the bottom display.
; Affects R0, R2, R3, R4, R5.
; The Carry flag is affected.
    CALL lsWSeg                    ; Write dummy segment into 4015 shift
register.
    CALL lsWSeg                    ; Write G segments.
    CALL lsWSeg                    ; Write F segments.
    CALL lsWSeg                    ; Write E segments.
    CALL lsWSeg                    ; Write D segments.
    CALL lsWSeg                    ; Write C segments.
    CALL lsWSeg                    ; Write B segments.
    CALL lsWSeg                    ; Write A segments.
```

```
    RET


lsWSeg:
; Subroutine WriteSegments:
; Writes the segment information in the LSB of R2, R3, R4, R5 to data bits
; D3, D2, D1 and D0 of a dummy external address.
; R0 is used as a temporary store.
; R2, R3, R4 and R5 are returned rotated right through carry by 1 bit.
; The Carry flag is affected.
    MOV R0, #0x00               ; Clear R0.

  ; Write segment for the top display to R0:
    MOV A, R2
    RRC A                       ; Move segment data to carry flag.
    MOV R2, A
    JNC lR2OK                   ; If segment is zero, jump.
    INC R0                      ; Segment was one so make lowest bit of R0 one.
  lR2OK:

  ; Prepare R0 to receive next bit:
    MOV A, R0
    RL A
    MOV R0, A
  ; Write segment for the 2nd. from top display to R0:
    MOV A, R3
    RRC A                       ; Move segment data to carry flag.
    MOV R3, A
    JNC lR3OK                   ; If segment is zero, jump.
    INC R0                      ; Segment was one so make lowest bit of R0 one.
  lR3OK:

  ; Prepare R0 to receive next bit:
    MOV A, R0
    RL A
    MOV R0, A
  ; Write segment for the 2nd. from bottom display to R0:
    MOV A, R4
    RRC A                       ; Move segment data to carry flag.
    MOV R4, A
    JNC lR4OK                   ; If segment is zero, jump.
    INC R0                      ; Segment was one so make lowest bit of R0 one.
  lR4OK:

  ; Prepare R0 to receive next bit:
    MOV A, R0
    RL A
    MOV R0, A
  ; Write segment for the bottom display to R0:
    MOV A, R5
    RRC A                       ; Move segment data to carry flag.
    MOV R5, A
    JNC lR5OK                   ; If segment is zero, jump.
    INC R0                      ; Segment was one so make lowest bit of R0 one.
  lR5OK:

  ; Send segment data, which is now in R0, to the displays:
    MOV A, R0
    MOVX @R0, A                    ; Dummy, irrelevant address in R0.
    RET
```

```
l24H12:
; Converts the 24 hour value in the accumulator (0 to 23) to the 12 hour value
; (0 to 12).
; The Carry flag is affected.
    JNZ #ln12AA                     ; Special case: Is it 12AM (hour===0)?
    MOV A, #0x0C                ; - Yes, so make A=12.
    RET
  ln12AA:                       ; - No, it is not 12AM.
    ADD A, #0xF3                ;   Set carry bit if Hour > 13
    JNC lAMA                    ;   Is it 1PM (hour==13) or beyond?
  ; It is 1PM (hour==13) or beyond, so subtract 12 from hour value.
  ; To subtract 12, we would have to add 12's two's complement (0xF4) to A.
  ; Since we have already added 0xF3 above, all we need to do is INC A.
    INC A
    RET
  lAMA:
  ; It is from 1 to 12AM, so A has to be returned unaltered. However, we have
  ; already added 0xF3 to A, so we need to add 0xF3's two's complement to
  ; bring A back to where it was.
    ADD A, #0x0D
    RET


.org 0x0500                     ; Helps avoid page break within subroutine.


lStClk:
; Subroutine SetClock: Increments the appropriate counter (minute, hour, date
; etc) relevant to the setting of P1.1, P1.2 and P1.3, or resets the seconds
; counter.
; Uses R0.
; The Carry flag is affected.
  ; Check the setting of P1.1, P1.2 and P1.3:
    IN A, P1                    ; Get P1 state.
    ANL A, #0x0E                ; 00001110 (keep only P1.1, P1.2 and P1.3).
    RR A                        ; Move to bits 0, 1 and 2.
    ADD A, #lSCJmp - 0x0500     ; SetClock jump table origin.
    JMPP @A                     ; Select the appropriate routine.
  lSCJmp:                       ; SetClock jump table:
    .db #lSCSe - 0x0500
    .db #lSCMi - 0x0500
    .db #lSCHr - 0x0500
    .db #lSCDt - 0x0500
    .db #lSCMo - 0x0500
    .db #lSCYr - 0x0500
    .db #lSCAH - 0x0500
    .db #lSCAM - 0x0500

  lSCSe:                        ; Reset the seconds counter:
    CLR A
    MOV R0, bSec
    MOV @R0, A
    RET

  lSCMi:
    CALL lIncMi                     ; Increment minute.
    RET
```

```
  lSCHr:
    CALL lIncHr                     ; Increment hour.
    RET

  lSCDt:
    CALL lIncDt                     ; Increment date.
    RET

  lSCMo:
    CALL lIncMo                     ; Increment month.
    RET

  lSCYr:
    CALL lIncYr                     ; Increment year.
    RET

  lSCAH:                          ; Increment Alarm Hour:
    MOV R0, bAlmHr                ; Get current Alarm Hour value.
    MOV A, @R0
    ADD A, #0xE9                  ; Set carry flag if bAlmHr >= 23.
    MOV A, @R0                    ; Get value of bAlmHr again.
    INC A                         ; Increment alarm hour.
    JNC lnH23B                    ; Was bAlmHr >= 23 (carry flag from above)?
    CLR A                         ; - Yes, so set bAlmHr=0.
  lnH23B:
    MOV @R0, A                    ; Store bAlmHr.
    RET

  lSCAM:                          ; Increment Alarm Minute:
    MOV R0, bAlmMn                ; Get current Alarm Minutes value.
    MOV A, @R0
    ADD A, #0xC5                  ; Set carry flag if bAlmMn >= 59.
    MOV A, @R0                    ; Get value of bAlmMn again.
    INC A                         ; Increment alarm minutes.
    JNC lnM59B                    ; Was bAlmMn >= 59 (carry flag from above)?
    CLR A                         ; - Yes, so set bAlmMn=0.
  lnM59B:
    MOV @R0, A                    ; Store bAlmMn.
    RET


lIncTm:
; Subroutine IncrementTime: Increments the seconds counter, and rolls over
; to all other time units.
; Uses R0.
; The Carry flag is affected.
  ; Increment the seconds counter bSec. If the seconds counter reaches 59,
  ; it is returned to 0 and the Carry flag is set.
    MOV R0, bSec
    MOV A, @R0
    ADD A, #0xC5                  ; Set carry flag if bSec >= 59.
    MOV A, @R0                    ; Get value of bSec again.
    INC A                         ; Increment seconds.
    JNC lnS59A                    ; Was bSec >= 59 (carry flag from above)?
    CLR A                         ; - Yes, so set bSec=0.
  lnS59A:
    MOV @R0, A                    ; Store bSec.

    JNC lEndB                     ; Did seconds roll over? If not, we are done.
```

```
  ; Seconds did roll over, so increment minutes:
    CALL lIncMi
    JNC lEndB                   ; Did minutes roll over? If not, we are done.

  ; Minutes did roll over, so increment hours:
    CALL lIncHr
    JNC lEndB                   ; Did hours roll over? If not, we are done.

  ; Hours did roll over, so increment date:
    CALL lIncDt
    JNC lEndB                   ; Did date roll over? If not, we are done.

  ; Date did roll over, so increment month:
    CALL lIncMo
    JNC lEndB                   ; Did month roll over? If not, we are done.

  ; Month did roll over, so increment year:
    CALL lIncYr

  lEndB:                        ; Finished.
    RET


lIncMi:
; Subroutine lIncrementMinutes. Increments the minute counter bMin.
; If the minutes counter reaches 59, it is returned to 0 and the Carry flag
; is set.
; Uses R0.
; Affects the Carry flag.
    MOV R0, bMin
    MOV A, @R0
    ADD A, #0xC5                ; Set carry flag if bMin >= 59.
    MOV A, @R0                  ; Get value of bMin again.
    INC A                       ; Increment minutes.
    JNC lnM59A                  ; Was bMin >= 59 (carry flag from above)?
    CLR A                       ; - Yes, so set bMin=0.
  lnM59A:
    MOV @R0, A                  ; Store bMin.
    RET


lIncHr:
; Subroutine lIncrementHours. Increments the hours counter bHour.
; If the hours counter reaches 23, it is returned to 0 and the Carry flag
; is set.
; Uses R0.
; Affects the Carry flag.
    MOV R0, bHour
    MOV A, @R0
    ADD A, #0xE9                ; Set carry flag if bHour >= 23.
    MOV A, @R0                  ; Get value of bHour again.
    INC A                       ; Increment hours.
    JNC lnH23A                  ; Was bHour >= 23 (carry flag from above)?
    CLR A                       ; - Yes, so set bHour=0.
  lnH23A:
    MOV @R0, A                  ; Store bHour.
    RET


lIncDt:
```

```
; Subroutine lIncrementDate. Increments the date counter bDate.
; If the date counter reaches the number of days in the month, it is returned
; to 1 and the Carry flag is set.
; Note that this subroutine does cater for leap years.
; Uses R0.
; Affects the Carry flag.
; Also uses bMonth, bYear.
  ; Set A to 2's complement of number of days in month:
    MOV R0, bMonth
    MOV A, @R0                  ; Get current month.
    ADD A, #0xFE                ; Is it February?
    JNZ lnFebA                  ; - No, so use lookup table.

  ; It is February, so use a special routine:
    MOV R0, bYear
    MOV A, @R0                  ; Get current year.
    ANL A, 0x03                    ; Keep only the lowest two bits.
    JZ lLeapY                   ; Are the lowest two bits 0?
    MOV A, #0xE4                ; - No (not leap year). A=2's complement of 28.
    JMP lContA
  lLeapY:
    MOV A, #0xE3                ; - Yes (leap year). A=2's complement of 29.
    JMP lContA

  lnFebA:
  ; It is not February, so use lookup table to find how many days in month.
    MOV R0, bMonth
    MOV A, @R0                  ; Get current month.
    ADD A, #lMDLkp - 0x0500     ; Lookup table base address.
    MOVP A, @A
    JMP lContA
  lMDLkp:                       ; Lookup table for days in each month:
    .db #0x00                   ; Dummy (month 0).
    .db #0xE1                   ; January      (2's complement of 31).
    .db #0xE4                   ; February     (2's complement of 28).
    .db #0xE1                   ; March        (2's complement of 31).
    .db #0xE2                   ; April        (2's complement of 30).
    .db #0xE1                   ; May          (2's complement of 31).
    .db #0xE2                   ; June         (2's complement of 30).
    .db #0xE1                   ; July         (2's complement of 31).
    .db #0xE1                   ; August (2's complement of 31).
    .db #0xE2                   ; September    (2's complement of 30).
    .db #0xE1                   ; October      (2's complement of 31).
    .db #0xE2                   ; November     (2's complement of 30).
    .db #0xE1                   ; December     (2's complement of 31).
  lContA:                       ; We arrive here with A==2's complement of
                                ; number of days in month.
    MOV R0, bDate
    ADD A, @R0                  ; Set carry flag if bDate >= days in month.
    MOV A, @R0                  ; Get value of bDate again.
    INC A                       ; Increment date.
    JNC lnDMxA                  ; Was bDate >= days in month (carry above)?
    MOV A, #0x01                ; - Yes, so set bDate=1.
  lnDMxA:
    MOV @R0, A                  ; Store bDate.
    RET


lIncMo:
; Subroutine lIncrementMonths. Increments the months counter bMonth.
```

```
; If the months counter reaches 12, it is returned to 1 and the Carry flag
; is set.
; Uses R0.
; Affects the Carry flag.
    MOV R0, bMonth
    MOV A, @R0
    ADD A, #0xF4                ; Set carry flag if bMonth >= 12.
    MOV A, @R0                  ; Get value of bMonth again.
    INC A                       ; Increment months.
    JNC lnM12A                  ; Was bMonth >= 12 (carry flag from above)?
    MOV A, #0x01                ; - Yes, so set bMonth=1.
  lnM12A:
    MOV @R0, A                  ; Store bMonth.
    RET


lIncYr:
; Subroutine lIncrementYears. Increments the years counter bYear.
; If the years counter reaches 99, it is returned to 15 and the Carry flag
; is set.
; Uses R0.
; Affects the carry flag.
    MOV R0, bYear
    MOV A, @R0
    ADD A, #0x9D                ; Set carry flag if bYear >= 99.
    MOV A, @R0                  ; Get value of bYear again.
    INC A                       ; Increment years.
    JNC lnY99A                  ; Was bYear >= 99 (carry flag from above)?
    MOV A, #0x0F                ; - Yes, so set bYear=15.
  lnY99A:
    MOV @R0, A                  ; Store bYear.
    RET


; -------------------- 7-SEGMENT DISPLAY LOOKUP SUBROUTINES -------------------


.org 0x0600                     ; Helps avoid page break within subroutine.


l7SegT:
; Translates the hexadecimal number in the accumulator to its 7-segment
; representation for the tens digit. The result is returned in the accumulator.
; Only works for numbers up to 99 decimal (0x63).
; Note that this is routine is frightfully wasteful of space; however, it
; returns in very few clock cycles (whereas a "proper" binary-to-BCD conversion
; would take about 80 clock cycles).
    ADD A, #l7SLkT - 0x0600     ; Lookup table base address.
    MOVP A, @A
    RET
  l7SLkT:                       ; Lookup table:
    .db eNum0                   ; Decimal 0.
    .db eNum0                   ; Decimal 1.
    .db eNum0                   ; Decimal 2.
    .db eNum0                   ; Decimal 3.
    .db eNum0                   ; Decimal 4.
    .db eNum0                   ; Decimal 5.
    .db eNum0                   ; Decimal 6.
    .db eNum0                   ; Decimal 7.
    .db eNum0                   ; Decimal 8.
```

```
.db eNum0                   ; Decimal 9.
.db eNum1                   ; Decimal 10.
.db eNum1                   ; Decimal 11.
.db eNum1                   ; Decimal 12.
.db eNum1                   ; Decimal 13.
.db eNum1                   ; Decimal 14.
.db eNum1                   ; Decimal 15.
.db eNum1                   ; Decimal 16.
.db eNum1                   ; Decimal 17.
.db eNum1                   ; Decimal 18.
.db eNum1                   ; Decimal 19.
.db eNum2                   ; Decimal 20.
.db eNum2                   ; Decimal 21.
.db eNum2                   ; Decimal 22.
.db eNum2                   ; Decimal 23.
.db eNum2                   ; Decimal 24.
.db eNum2                   ; Decimal 25.
.db eNum2                   ; Decimal 26.
.db eNum2                   ; Decimal 27.
.db eNum2                   ; Decimal 28.
.db eNum2                   ; Decimal 29.
.db eNum3                   ; Decimal 30.
.db eNum3                   ; Decimal 31.
.db eNum3                   ; Decimal 32.
.db eNum3                   ; Decimal 33.
.db eNum3                   ; Decimal 34.
.db eNum3                   ; Decimal 35.
.db eNum3                   ; Decimal 36.
.db eNum3                   ; Decimal 37.
.db eNum3                   ; Decimal 38.
.db eNum3                   ; Decimal 39.
.db eNum4                   ; Decimal 40.
.db eNum4                   ; Decimal 41.
.db eNum4                   ; Decimal 42.
.db eNum4                   ; Decimal 43.
.db eNum4                   ; Decimal 44.
.db eNum4                   ; Decimal 45.
.db eNum4                   ; Decimal 46.
.db eNum4                   ; Decimal 47.
.db eNum4                   ; Decimal 48.
.db eNum4                   ; Decimal 49.
.db eNum5                   ; Decimal 50.
.db eNum5                   ; Decimal 51.
.db eNum5                   ; Decimal 52.
.db eNum5                   ; Decimal 53.
.db eNum5                   ; Decimal 54.
.db eNum5                   ; Decimal 55.
.db eNum5                   ; Decimal 56.
.db eNum5                   ; Decimal 57.
.db eNum5                   ; Decimal 58.
.db eNum5                   ; Decimal 59.
.db eNum6                   ; Decimal 60.
.db eNum6                   ; Decimal 61.
.db eNum6                   ; Decimal 62.
.db eNum6                   ; Decimal 63.
.db eNum6                   ; Decimal 64.
.db eNum6                   ; Decimal 65.
.db eNum6                   ; Decimal 66.
.db eNum6                   ; Decimal 67.
.db eNum6                   ; Decimal 68.
```

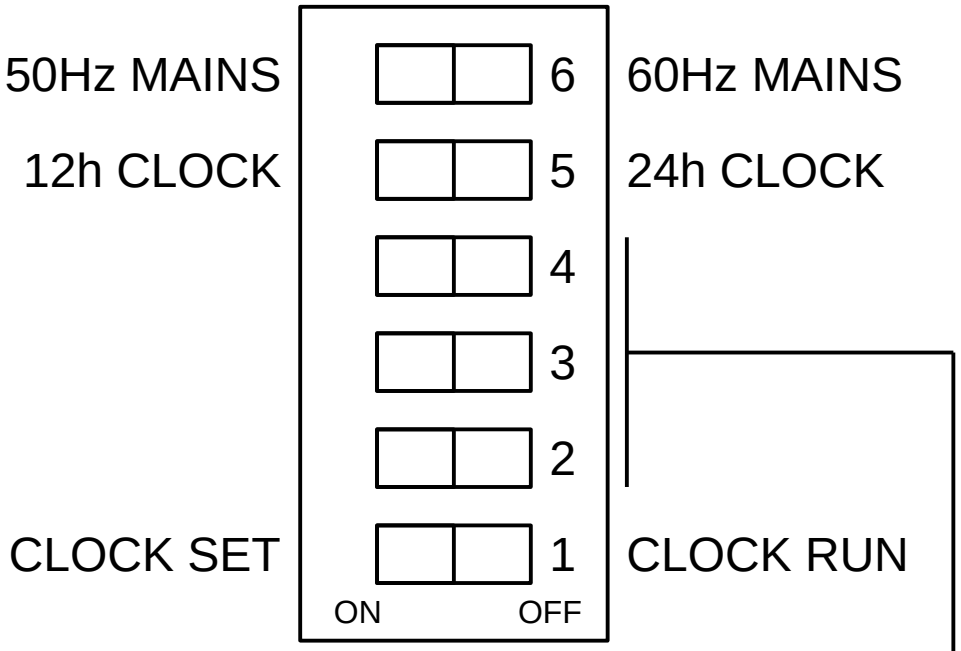
```
    .db eNum6                   ; Decimal 69.
    .db eNum7                   ; Decimal 70.
    .db eNum7                   ; Decimal 71.
    .db eNum7                   ; Decimal 72.
    .db eNum7                   ; Decimal 73.
    .db eNum7                   ; Decimal 74.
    .db eNum7                   ; Decimal 75.
    .db eNum7                   ; Decimal 76.
    .db eNum7                   ; Decimal 77.
    .db eNum7                   ; Decimal 78.
    .db eNum7                   ; Decimal 79.
    .db eNum8                   ; Decimal 80.
    .db eNum8                   ; Decimal 81.
    .db eNum8                   ; Decimal 82.
    .db eNum8                   ; Decimal 83.
    .db eNum8                   ; Decimal 84.
    .db eNum8                   ; Decimal 85.
    .db eNum8                   ; Decimal 86.
    .db eNum8                   ; Decimal 87.
    .db eNum8                   ; Decimal 88.
    .db eNum8                   ; Decimal 89.
    .db eNum9                   ; Decimal 90.
    .db eNum9                   ; Decimal 91.
    .db eNum9                   ; Decimal 92.
    .db eNum9                   ; Decimal 93.
    .db eNum9                   ; Decimal 94.
    .db eNum9                   ; Decimal 95.
    .db eNum9                   ; Decimal 96.
    .db eNum9                   ; Decimal 97.
    .db eNum9                   ; Decimal 98.
    .db eNum9                   ; Decimal 99.


l7SegU:
; Translates the hexadecimal number in the accumulator to its 7-segment
; representation for the units digit. The result is returned in the
; accumulator.
; Only works for numbers up to 99 decimal (0x63).
; Note that this is routine is frightfully wasteful of space; however, it
; returns in very few clock cycles (whereas a "proper" binary-to-BCD conversion
; would take about 80 clock cycles).
    ADD A, #l7SLkU - 0x0600     ; Lookup table base address.
    MOVP A, @A
    RET
  l7SLkU:                       ; Lookup table:
    .db eNum0                   ; Decimal 0.
    .db eNum1                   ; Decimal 1.
    .db eNum2                   ; Decimal 2.
    .db eNum3                   ; Decimal 3.
    .db eNum4                   ; Decimal 4.
    .db eNum5                   ; Decimal 5.
    .db eNum6                   ; Decimal 6.
    .db eNum7                   ; Decimal 7.
    .db eNum8                   ; Decimal 8.
    .db eNum9                   ; Decimal 9.
    .db eNum0                   ; Decimal 10.
    .db eNum1                   ; Decimal 11.
    .db eNum2                   ; Decimal 12.
    .db eNum3                   ; Decimal 13.
    .db eNum4                   ; Decimal 14.
```

```
.db eNum5                   ; Decimal 15.
.db eNum6                   ; Decimal 16.
.db eNum7                   ; Decimal 17.
.db eNum8                   ; Decimal 18.
.db eNum9                   ; Decimal 19.
.db eNum0                   ; Decimal 20.
.db eNum1                   ; Decimal 21.
.db eNum2                   ; Decimal 22.
.db eNum3                   ; Decimal 23.
.db eNum4                   ; Decimal 24.
.db eNum5                   ; Decimal 25.
.db eNum6                   ; Decimal 26.
.db eNum7                   ; Decimal 27.
.db eNum8                   ; Decimal 28.
.db eNum9                   ; Decimal 29.
.db eNum0                   ; Decimal 30.
.db eNum1                   ; Decimal 31.
.db eNum2                   ; Decimal 32.
.db eNum3                   ; Decimal 33.
.db eNum4                   ; Decimal 34.
.db eNum5                   ; Decimal 35.
.db eNum6                   ; Decimal 36.
.db eNum7                   ; Decimal 37.
.db eNum8                   ; Decimal 38.
.db eNum9                   ; Decimal 39.
.db eNum0                   ; Decimal 40.
.db eNum1                   ; Decimal 41.
.db eNum2                   ; Decimal 42.
.db eNum3                   ; Decimal 43.
.db eNum4                   ; Decimal 44.
.db eNum5                   ; Decimal 45.
.db eNum6                   ; Decimal 46.
.db eNum7                   ; Decimal 47.
.db eNum8                   ; Decimal 48.
.db eNum9                   ; Decimal 49.
.db eNum0                   ; Decimal 50.
.db eNum1                   ; Decimal 51.
.db eNum2                   ; Decimal 52.
.db eNum3                   ; Decimal 53.
.db eNum4                   ; Decimal 54.
.db eNum5                   ; Decimal 55.
.db eNum6                   ; Decimal 56.
.db eNum7                   ; Decimal 57.
.db eNum8                   ; Decimal 58.
.db eNum9                   ; Decimal 59.
.db eNum0                   ; Decimal 60.
.db eNum1                   ; Decimal 61.
.db eNum2                   ; Decimal 62.
.db eNum3                   ; Decimal 63.
.db eNum4                   ; Decimal 64.
.db eNum5                   ; Decimal 65.
.db eNum6                   ; Decimal 66.
.db eNum7                   ; Decimal 67.
.db eNum8                   ; Decimal 68.
.db eNum9                   ; Decimal 69.
.db eNum0                   ; Decimal 70.
.db eNum1                   ; Decimal 71.
.db eNum2                   ; Decimal 72.
.db eNum3                   ; Decimal 73.
.db eNum4                   ; Decimal 74.
```

```
.db eNum5                       ; Decimal 75.
.db eNum6                       ; Decimal 76.
.db eNum7                       ; Decimal 77.
.db eNum8                       ; Decimal 78.
.db eNum9                       ; Decimal 79.
.db eNum0                       ; Decimal 80.
.db eNum1                       ; Decimal 81.
.db eNum2                       ; Decimal 82.
.db eNum3                       ; Decimal 83.
.db eNum4                       ; Decimal 84.
.db eNum5                       ; Decimal 85.
.db eNum6                       ; Decimal 86.
.db eNum7                       ; Decimal 87.
.db eNum8                       ; Decimal 88.
.db eNum9                       ; Decimal 89.
.db eNum0                       ; Decimal 90.
.db eNum1                       ; Decimal 91.
.db eNum2                       ; Decimal 92.
.db eNum3                       ; Decimal 93.
.db eNum4                       ; Decimal 94.
.db eNum5                       ; Decimal 95.
.db eNum6                       ; Decimal 96.
.db eNum7                       ; Decimal 97.
.db eNum8                       ; Decimal 98.
.db eNum9                       ; Decimal 99.
```

APPENDIX D: INSTRUCTION SHEET



ITEM TO BE SET:

SW4	SW3	SW2	Function
On	On	On	Reset Seconds
On	On	Off	Set Minute
On	Off	On	Set Hour
On	Off	Off	Set Date
Off	On	On	Set Month
Off	On	Off	Set Year
Off	Off	On	Set Alarm Hour
Off	Off	Off	Set Alarm Minute

To set the clock:

- Set SW1 to "On".
- Set SW2, SW3 and SW4 to the desired positions.
- Push the pushbutton to advance the selected item.
- When finished with all settings, return SW1 to "Off".

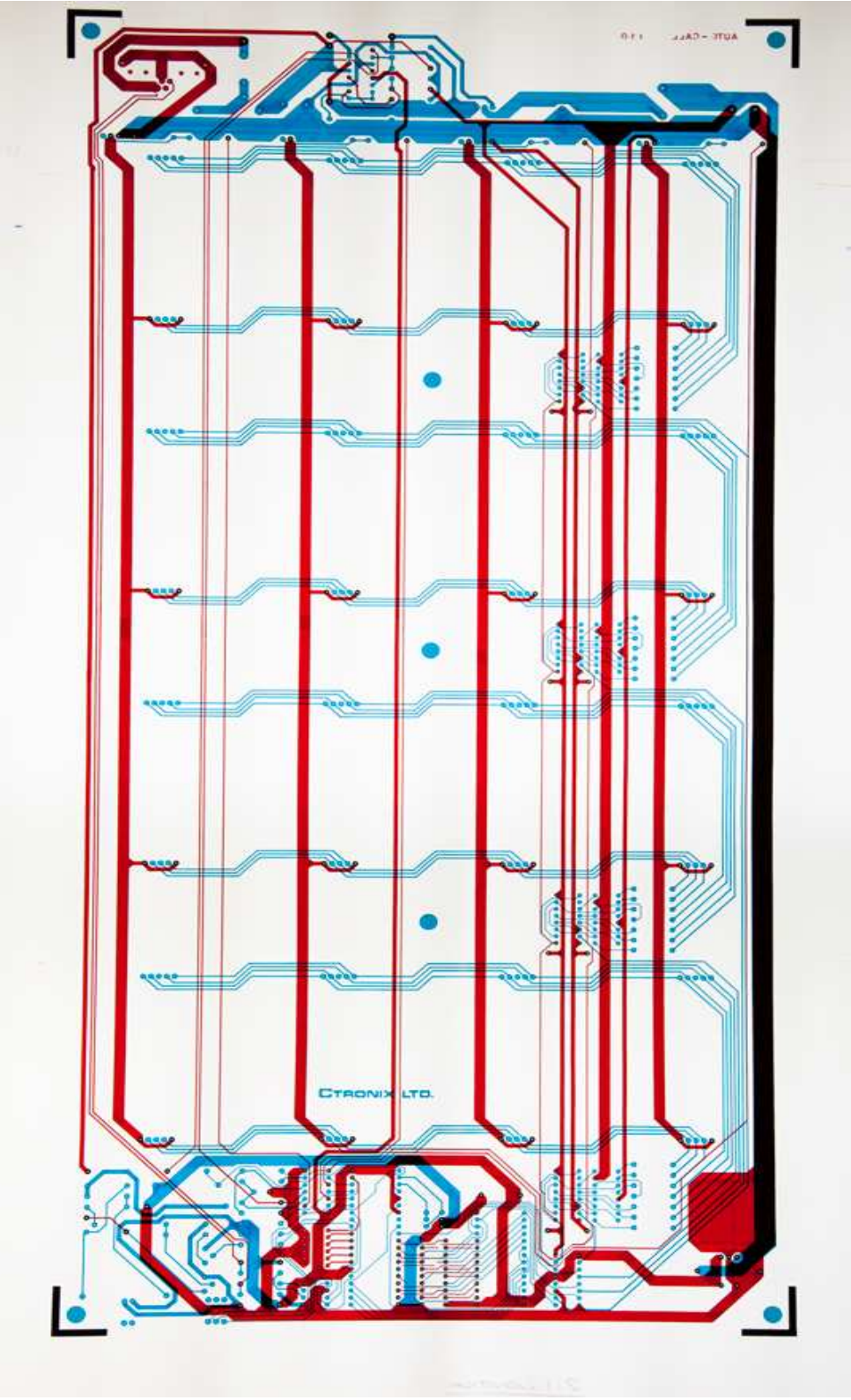
Pushbutton funtions:

- While SW1 is "On" (setting the clock): Pushing the button advances the selected item.
- While SW1 is "Off": Pushing the button displays the alarm time. Holding the button pressed for 1 second toggles the alarm on ("L" displayed on screen) and off.
- Pushing the button while alarm is sounding: Silences the alarm. The alarm will sound again tomorrow (if it is kept on).

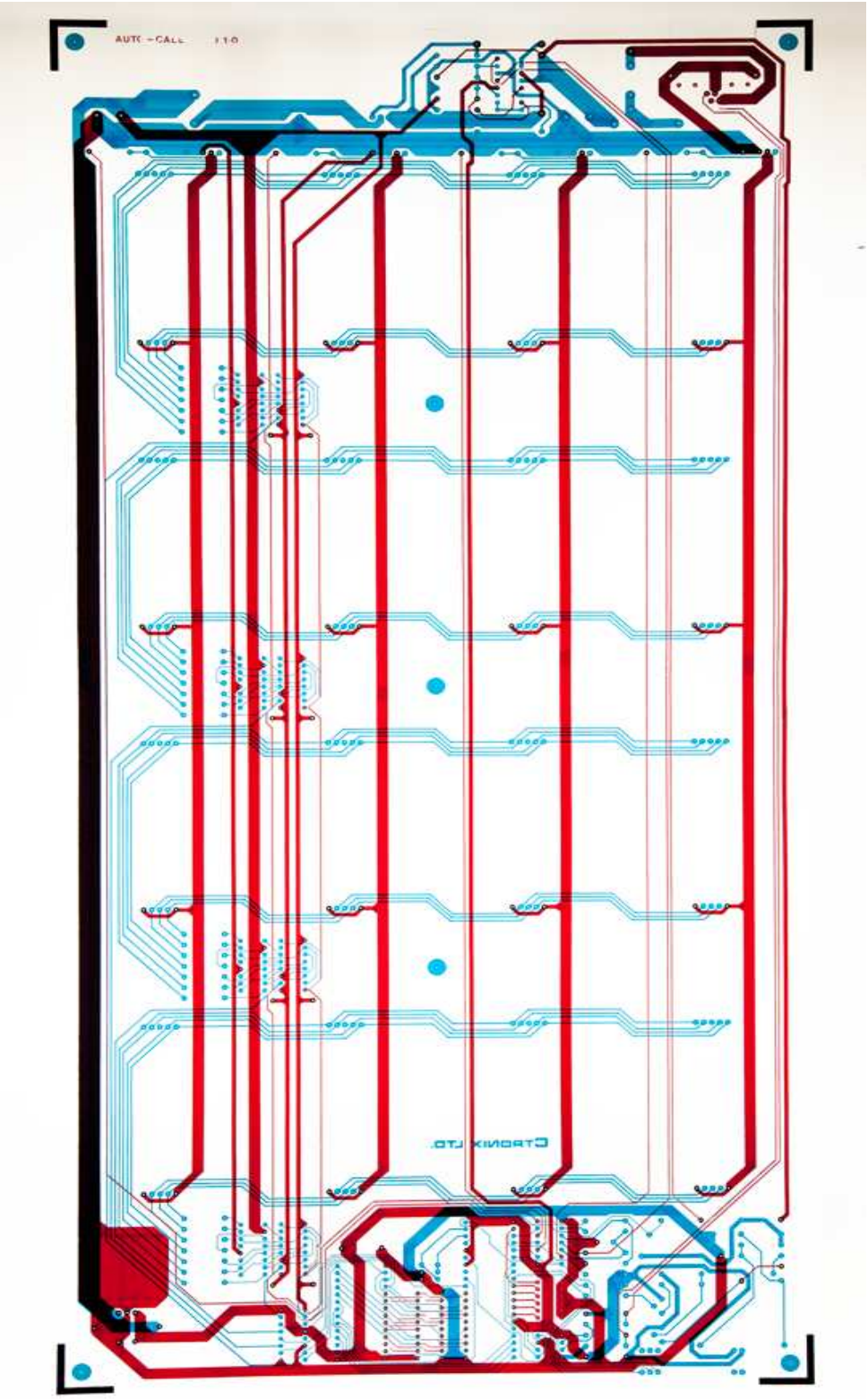
APPENDIX E: PCB LAYOUTS

Kindly provided by Ctronix:

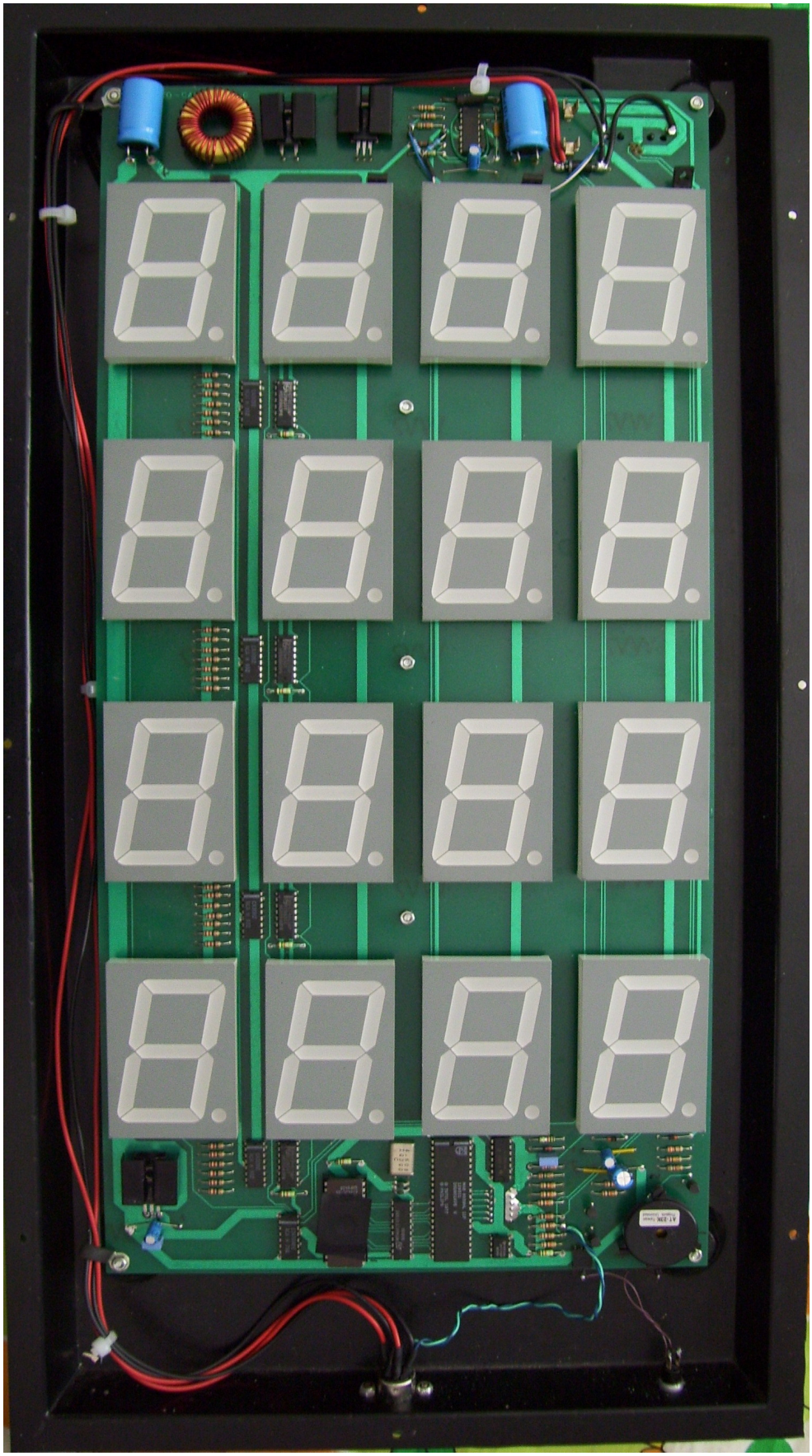
Bottom layer

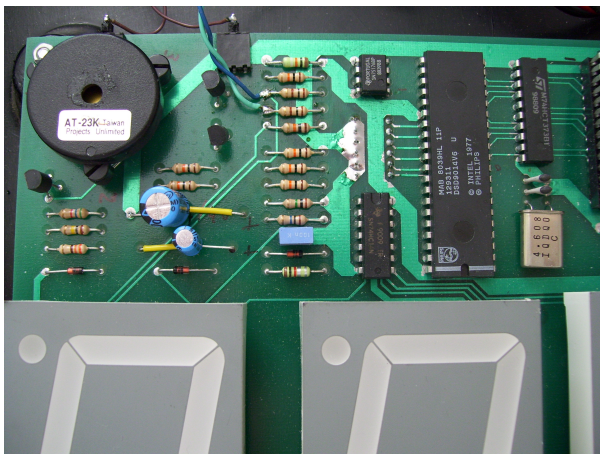
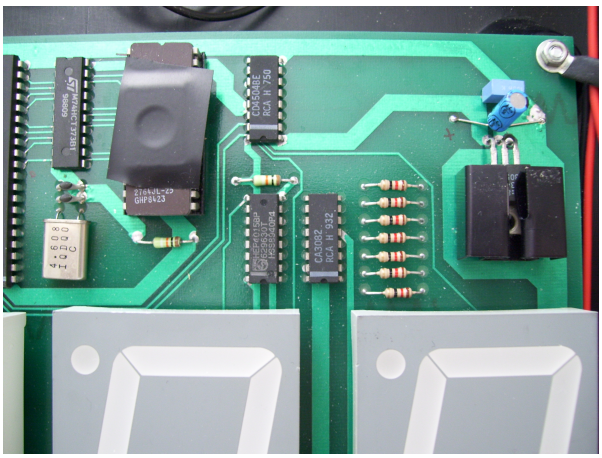
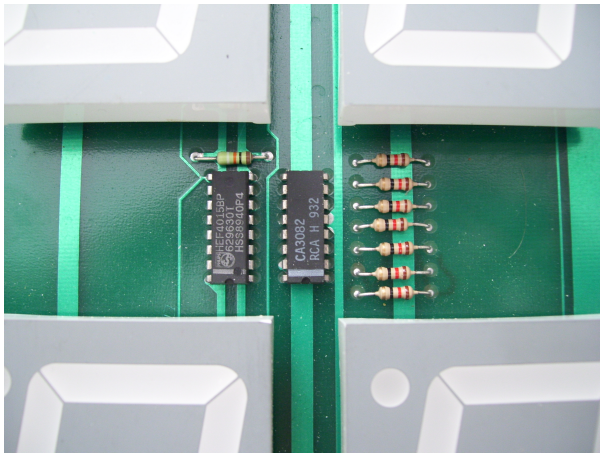
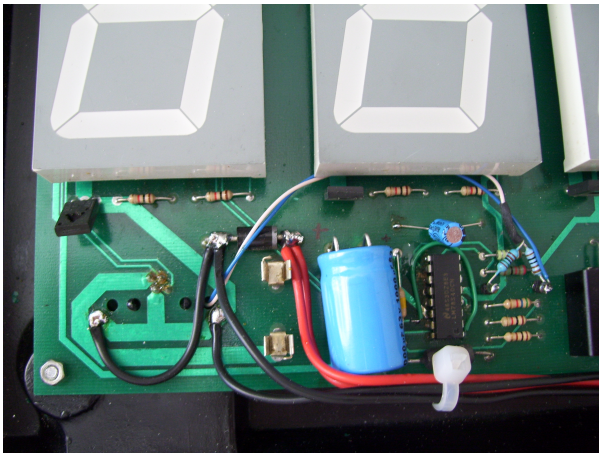
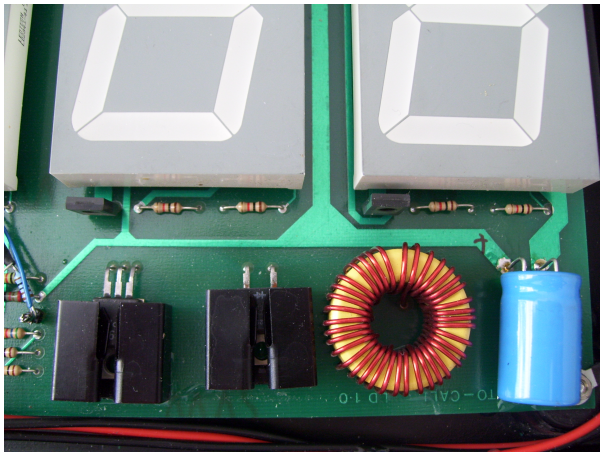
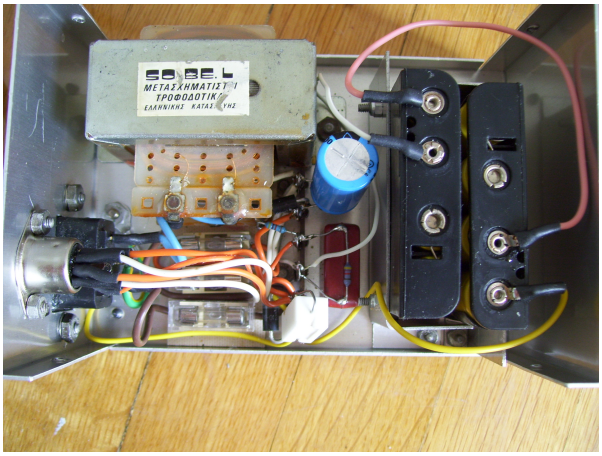


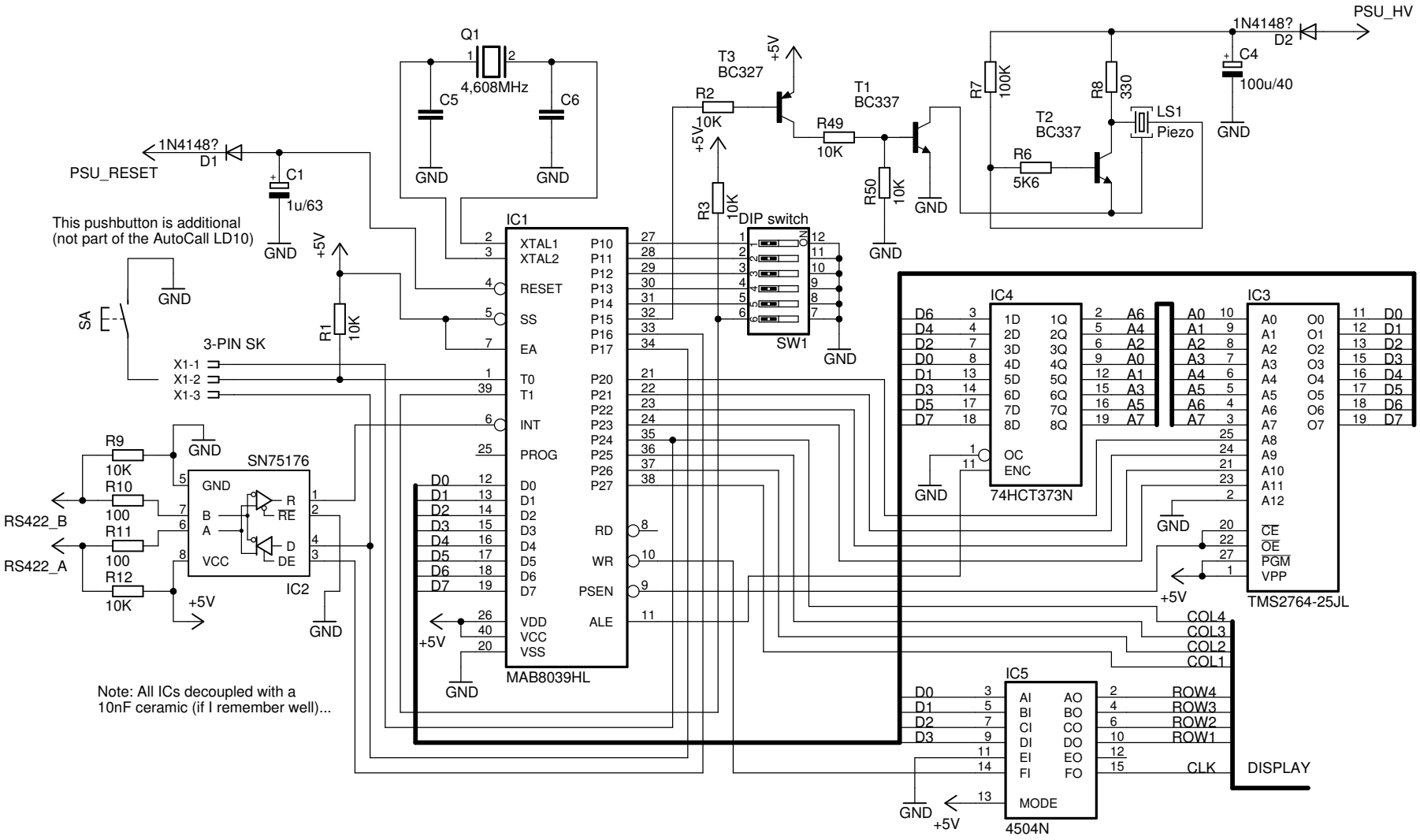
Top layer

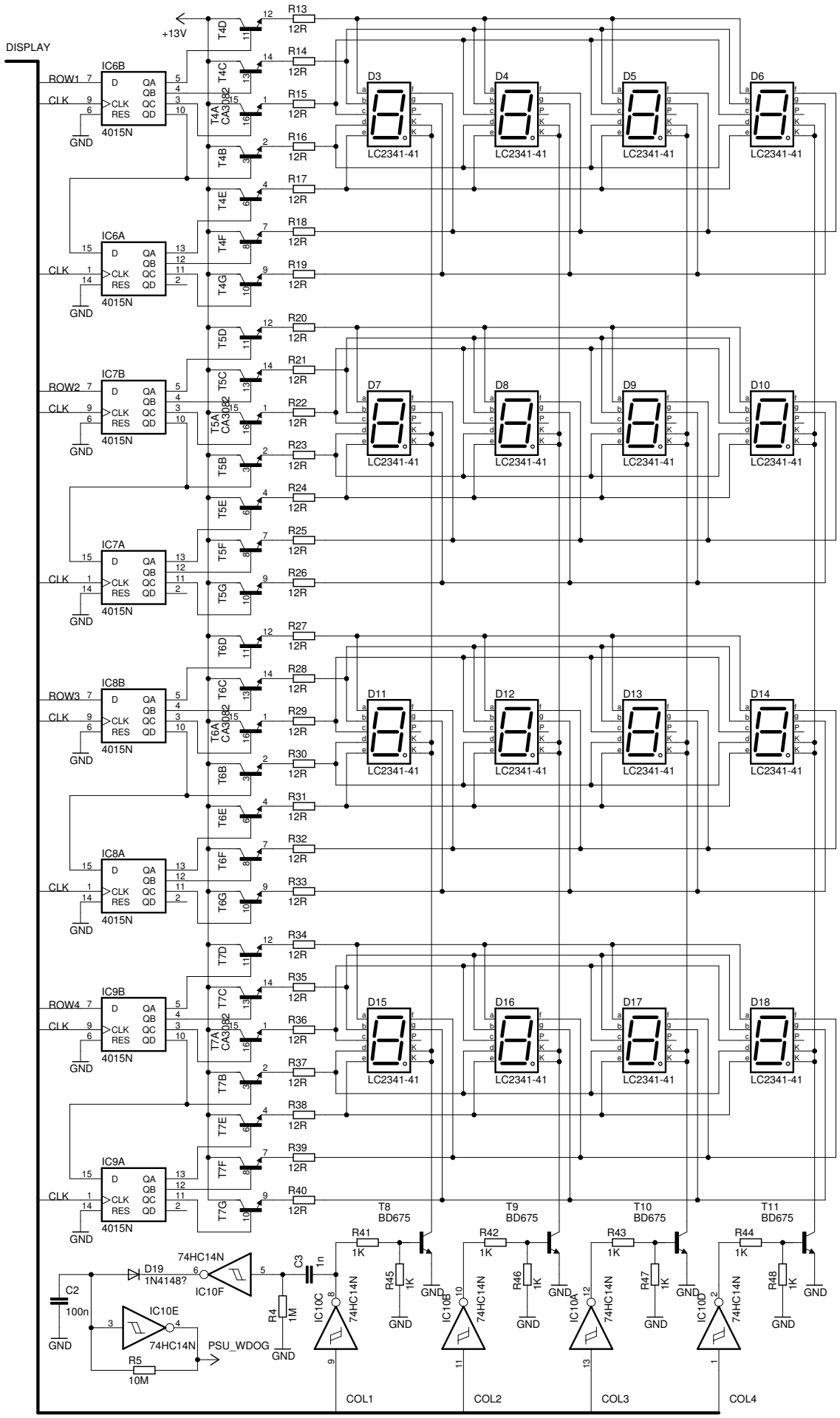


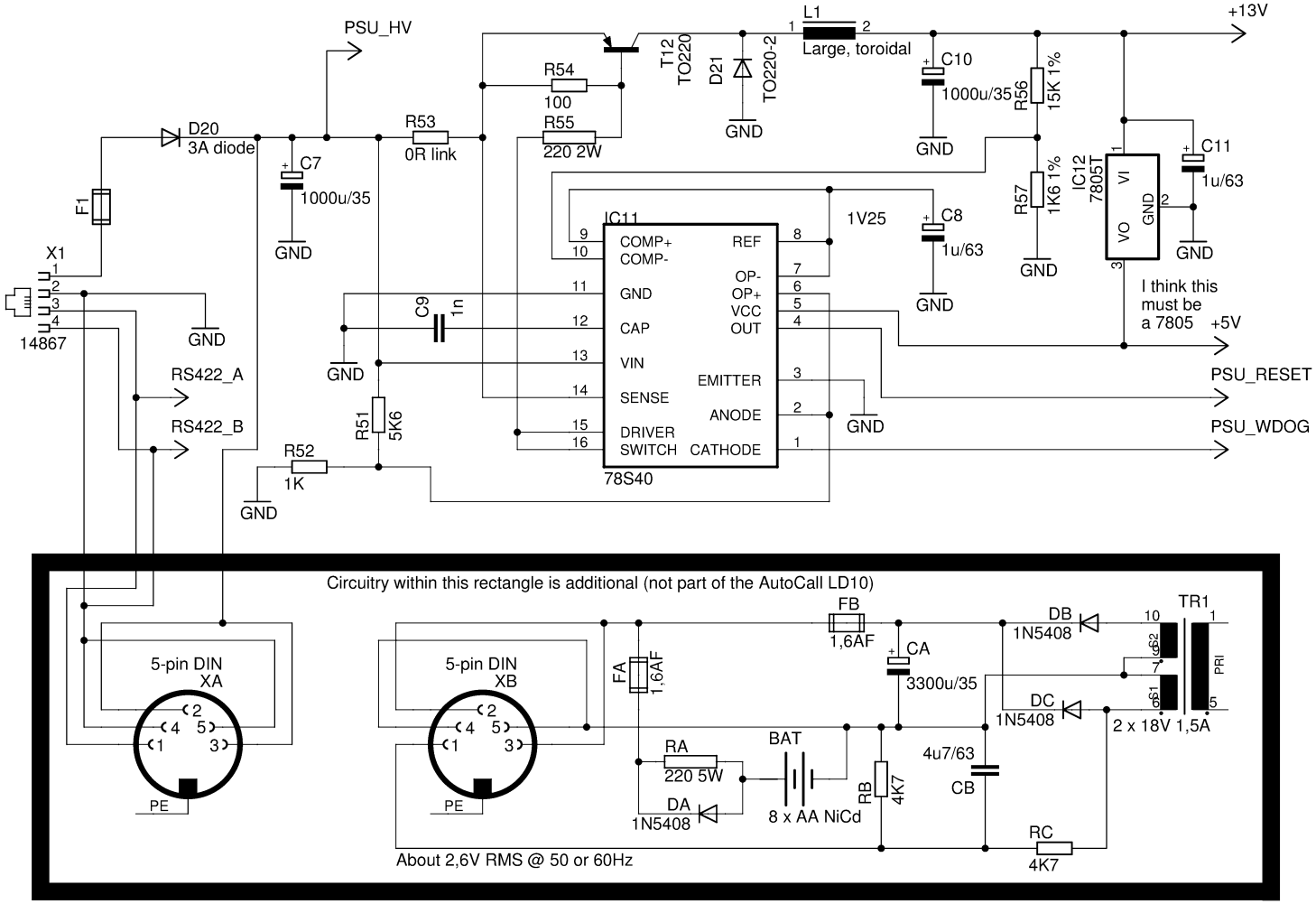
APPENDIX F: INNER WORKINGS AND SCHEMATICS




AT-23K-Taiwan
Projects Unlimited


CA3082
RCA H 932





Q1
4,608MHz
C5
C6
GND
T3
BC327
+5V
R2
10K
T1
BC337
R49
10K
R50
10K
R7
100K
R8
330
T2
BC337
R6
5K6
LS1
Piezo
1N4148?
D2
PSU_HV
C4
100u/40
1N4148?
D1
PSU_RESET
C1
1u/63
This pushbutton is additional
(not part of the AutoCall LD10)
SA
R1
10K
R3
10K
DIP switch
SW1
IC1
XTAL1
XTAL2
RESET
SS
EA
T0
T1
INT
PROG
VDD
VCC
VSS
P10
P11
P12
P13
P14
P15
P16
P17
P20
P21
P22
P23
P24
P25
P26
P27
RD
WR
PSEN
ALE
MAB8039HL
3-PIN SK
X1-1
X1-2
X1-3
R9
10K
R10
100
R11
100
R12
10K
RS422_B
RS422_A
SN75176
IC2
IC4
74HCT373N
IC3
TMS2764-25JL
OC
ENC
CE
OE
PGM
VPP
IC5
4504N
MODE
COL4
COL3
COL2
COL1
ROW4
ROW3
ROW2
ROW1
CLK
DISPLAY
Note: All ICs decoupled with a
10nF ceramic (if I remember well)...


DISPLAY
+13V
ROW1
ROW2
ROW3
ROW4
CLK
GND
IC6B
IC6A
IC7B
IC7A
IC8B
IC8A
IC9B
IC9A
4015N
CA3082
T4A
T4B
T4C
T4D
T4E
T4F
T4G
T5A
T5B
T5C
T5D
T5E
T5F
T5G
T6A
T6B
T6C
T6D
T6E
T6F
T6G
T7A
T7B
T7C
T7D
T7E
T7F
T7G
R13
R14
R15
R16
R17
R18
R19
R20
R21
R22
R23
R24
R25
R26
R27
R28
R29
R30
R31
R32
R33
R34
R35
R36
R37
R38
R39
R40
12R
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
D16
D17
D18
LC2341-41
T8
T9
T10
T11
BD675
R41
R42
R43
R44
R45
R46
R47
R48
1K
74HC14N
IC10A
IC10B
IC10C
IC10D
IC10E
IC10F
D19
1N4148?
C2
100n
C3
1n
R4
1M
R5
10M
PSU_WDOG
COL1
COL2
COL3
COL4


PSU_HV
+13V
L1
Large, toroidal
T12
TO220
D21
TO220-2
R54
100
R55
220 2W
R53
0R link
D20
3A diode
C7
1000u/35
C10
1000u/35
R56
15K 1%
R57
1K6 1%
IC12
7805T
C11
1u/63
C8
1u/63
1V25
F1
X1
14867
IC11
COMP+
COMP-
GND
CAP
VIN
SENSE
DRIVER
SWITCH
REF
OP-
OP+
VCC
OUT
EMITTER
ANODE
CATHODE
78S40
C9
1n
R51
5K6
R52
1K
I think this
must be
a 7805
+5V
PSU_RESET
PSU_WDOG
RS422_A
RS422_B
Circuitry within this rectangle is additional (not part of the AutoCall LD10)
5-pin DIN
XA
5-pin DIN
XB
PE
FA
1,6AF
FB
1,6AF
CA
3300u/35
DB
1N5408
DC
1N5408
TR1
2 x 18V 1,5A
RA
220 5W
DA
1N5408
BAT
8 x AA NiCd
RB
4K7
4u7/63
CB
RC
4K7
About 2,6V RMS @ 50 or 60Hz