EXPERIMENT 1

FAMILIARIZING YOURSELF WITH THE KIM-1

The purpose of the first experiment is to acquaint you with the mechanics of entering and running a program in the KIM. The numbers we enter are not expected to mean anything to us at this point in the course. A secondary purpose of experiment 1 is to test the programmable timer used by the KIM to load tapes.

The KIM uses MOS devices which can be damaged by high static discharges. The board comes in a conductive plastic bag which protects it from such damage. The board is also relatively immune from damage when connected to a power supply. Handle the board by its edges when moving it from bag to power supply. Discharge yourself on any convenient grounded surface, if you suspect that you may be carrying charge.

Power is applied through the lower edge connector on the board. The connector is applied with the power connections (red and black wires) at the top. Apply the power connector to the board BEFORE the power supply is plugged into AC power.

1. Upon powering up the KIM, push the Reset button labeled RS. This should cause the display to light. This indicates to us that the KIM's monitor program is running. The KIM display, like the display of a calculator is generated one digit at a time. By multiplexing the digits at a high rate, a continuous display is observed. Between the display scans, the KIM monitor multiplexes the keyboard to determine if a key has been pressed. A key closure is serviced when it is found. Ocasionally, due to external electrical noise, but more frequently due to operator error, the computer will leave the monitor or user program and quit executing code altogether. The computer is said to have"bombed". No damage is done. The computer is set on the track again by pressing the RESET button. The RESET button is also used to terminate execution of a user program and return control to the monitor program.

2. Place the slide switch on the keyboard in the OFF position. This switch has nothing to do with power and is used to enable a "debug" feature which we will inves-

tigate in the future. Similarly, refrain from pressing the ST or STOP button, until its use has been explained in the course.

3. The numbered keys 0 thru F are used to enter either address or data information. Reset puts the monitor in the address mode. The address mode can also be obtained by pressing the AD or address mode key.

Having done this, press the numbered keys and observe that the numbers shift in right to left (calculator style) in the leftmost four digits of the display. The left four digits represent an address. The right two digits represent the data at that address. Leave the address with $0000 in the display. The two right digits are the data located at memory location $0000.

4. Press the DA or data mode key. Nothing appears to happen. However, now when the numbered keys are pressed, the address remains at $0000 and instead the data changes. This tells us that location $0000 is read/write memory, more commonly called RAM. Leave location $0000 with data $A9.

5. It would be very cumbersone to have to enter each address when entering a program. Fortunately that's not necessary. Press the button labelled "+". Note that PLUS advances the address one location. The PLUS function is insensitive to mode. We are still in Data mode and may now enter data at $0001 without having to press the DA button again. Enter data $FF at $0001.

6. Using the PLUS button to access consecutive addresses, enter the program found on the following page.

7. How do you know that you have entered the program correctly? To check, go back to location $0000. We get there by pressing AD for address mode and keying in $0000. We should observe the $A9 which we entered there previously. Just hit PLUS to see the next location etc. If we encounter an error, we just enter the correct data. But don't fcrget to first hit the DA key. It's easy to leave out data. Be sure that the address and data agree at the end of the program. Check through at least twice that you have entered the program correctly.

ADDRESS	DATA
$0000	$A9
$0001	$FF
$0002	$8D
$0003	$47
$0004	$17
$0005	$20
$0006	$19
$0007	$1F
$0008	$2C
$0009	$47
$000A	$17
$000B	$10
$000C	$F8
$000D	$20
$000E	$63
$000F	$1F
$0010	$D0
$0011	$EE

8. After you are satisfied that the data has been entered correctly, you are ready to run the program. As you might have guessed, the program starts at $0000. Press AD and enter $0000. The program is executed by pressing the GO button. In this program, the display will remain lit if the program has been entered correctly. If the display goes blank, the computer has "bombed". Hit RESET to get back into the monitor and carefully recheck your code.

EXPERIMENT 2

INPUT/OUTPUT - KIM-1 and USER

The purpose of Experiment 2 is to introduce the subject of Input/Output. We will be connecting external I/O to the KIM using the two user programmable ports. We will also investigate the input built into the KIM board, namely the keyboard, and the output, namely the six digit seven segment display.

All of the remaining experiments will make use of the two user programmable ports on the KIM, Port A, and Port B. A very thorough description of these ports is to be found in the section called INPUT/OUTPUT. For simplicity, we will hereafter use Port A as an output port. A row of eight individual LEDs is connected to Port A to serve as an output indication. A row of four slide switches is connected to the low four bits of Port B which will be used only as an input Port.

In the I/O discussion we found out that both the programmable ports are initially configured as input ports by the RESET function. Therefore it will only be necessary to program Port A as and output port.

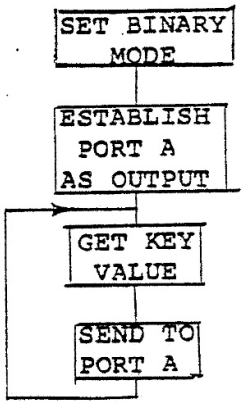
1. Key in the address $1701. This is the data direction register of Port A. Note that it is presently $00. Hit DAta and enter $FF. It is now configured as an output. Hit ADdress and enter $1700. Be careful not to hit RESET. Now hit DA and enter any data. You can now observe the data you entered on the external LEDs. See what the various hex characters look like. Hit reset. Note that $FF appears, since Reset programs port A as an input port. Hit DA and try to enter data. You can not. To observe data you will have to go to $1701 and again write $FF.

2. Enter Address $1702. This is Port B Data. Port B has already been configured as an input port by Reset. Use the four slide switches to affect the low four bits of Port B.

3.

3. Input for the stand-alone KIM is the on-board keyboard. The KIM checks for a key being pressed by a routine in the Monitor programs called GETKEY, located in ROM at location $1F6A. Since GETKEY is a subroutine, we are free to use it for our purposes. A complete listing of the KIM monitor programs can be found in Appendix I of the KIM-1 User's Manual. GETKEY in particular can be found on page 26. GETKEY uses the add instruction, and assumes that the computer is already in the binary mode. Thus the user should clear the decimal mode prior to calling GETKEY. If we look at the comments in the listing we see that GETKEY returns with the value of the key pressed in the "A" register. If no key is pressed, $15 is left in A.

To illustrate GETKEY, let us propose the following program. First configure Port A as an output port so that results can observed on the discrete LEDs. Now use GETKEY and display the result at Port A, cycling continuously. It is customary to show the flow in a program using a flow chart.



From the flow chart, we now write down the instructions which will accomplish the desired program. We use the manufacturer's symbology or mnemonics for the instruction names, and invent our own labels for hardware addresses and instruction locations which are reference by the program. This format is called assembly language. Finally, we go back and look up the codes for the instructions and fill in any address arguments. This last process is called assembling. A program which would do this to text which we enter through a terminal is called an assembler. We will be doing all of our assembly by hand.

```
                        ORIGIN = $0000   Program begins here
                        GETKEY = $1F6A
                        PADD   = $1701   Port A Data Direction
                        PAD    = $1700   Port A Data

$0000  D8               START   CLD      Set binary mode
$0001  A2 FF                    LDX# $FF
$0003  8E 01 17                 STX PADD   Make Port A Output
$0006  20 6A 1F         LOOP    JSR GETKEY      get key
$0009  8D 00 17                 STA PAD    Send to Port A
$000C  4C 06 00                 JMP LOOP
```

In this format, which is traditional assembly language format, there is one instruction per line, which may take one, two, or three bytes of code (and address locations). Since the addresses are not consecutive, it is very easy to make mistakes. It may prove easier initially to write the program with one byte per line as below. Of course, care must be taken to leave the proper number of lines for a particular instruction.

```
$0000  D8               START   CLD
$0001  A2                       LDX#
$0002  FF                       $FF
$0003  8E                       STX
$0004  01                       PADD
$0005  17                       ----
$0006  20                       JSR
$0007  6A                       GETKEY
$0008  1F                       ----
$0009  8D                       STA
$000A  00                       PAD
$000B  17                       ----
$000C  4C                       JMP
$000D  06                       LOOP
$000E  00                       ----
```

Note that in the instruction column, a dashed line is used to take up the space of a third byte in three byte instructions. Note that by using labels for instruction locations, we can write a program without having to know in advance at what memory location a particular instruction will fall.

4. The KIM's primary output is a six-digit seven segment LED display. The KIM uses the display mainly to display the data in a particular memory location. A program called SCAND at $1F19 reads the data at the location pointed at by the address pointer, stores it in a zero page location (INH $00F9) and then causes the address pointer and the memory contents to be displayed on the six digit display. The low part of the address pointer (called POINTL in KIM software) is location $00FA. The high part of the address pointer, called POINTH, is $00FB. The program starting at location SCANDS ($1F1F) causes the data in the three zero page locations referenced to be displayed as three pairs of hex digits.

The user may in fact store his own numerical information in these zero page locations and use SCANDS to display it. Note, however, that SCANDS multiplexes the display just once which lasts only a few milliseconds. To observe a continuous display, SCANDS must be called repeatedly. The program that follows causes three arbitrary constants to appear on the display. Execute this program to see the result.

```
$000F  A9          LDA#          Load first number
$0010  01          $01
$0011  85          STA           Store it in left display
$0012  FB          POINTH
$0013  A9          LDA#          Load second number
$0014  02          $02
$0015  85          STA           Store it in middle
$0016  FA          POINTL
$0017  A9          LDA#          Load third number
$0018  03          $03
$0019  85          STA           Store in on right
$001A  F9          INH
$001B  20  LOOP2   JSR           Call the display routine
$001C  1F          SCANDS
$001D  1F          ----
$001E  4C          JMP           Loop back
$001F  1B          LOOP2
$0020  00          ----
```

5. Modify the program so that constants of your choosing appear in the display.

6. Finally, you might try to combine the earlier program using GETKEY, with the last program such that the number of the key pressed appears across the display. For example, if you press "1", the display should read 010101.

It might help quite a bit to sketch out a flow chart of this new program which combines two programs. You may find that you would like to remove an instruction without having to move everything else up. Any unwanted byte can always be replaced with a one byte instruction that does nothing. Most computers have such an instruction called a No-operation or NOP. The 6502 has an NOP with code $EA.

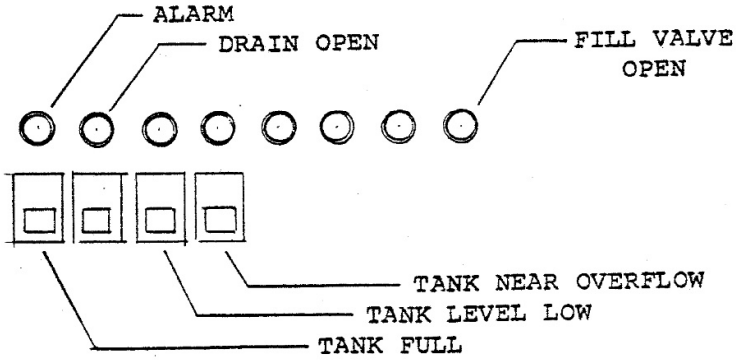
EXPERIMENT 3

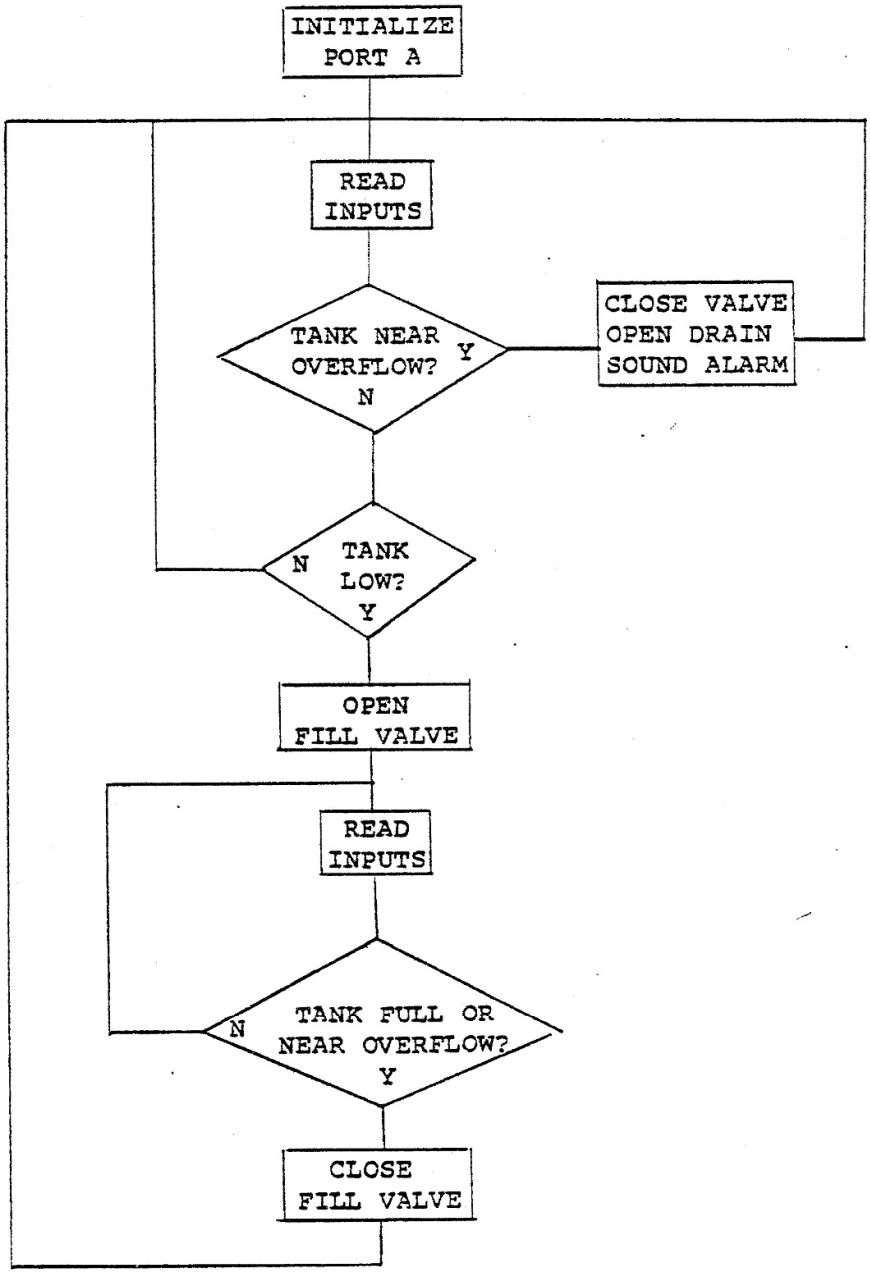
INPUT/OUTPUT - CONTROLLER APPLICATION

In this experiment, we will use the KIM as a controller in a real-world application. Even though the example program is a gross simplification of a real problem, it will give us a good idea how computers are used in controller applications. We will use the LEDs connected to PORT A to indicate our Outputs. We will use use the switches connected to PORT B to indicate the real-world inputs.

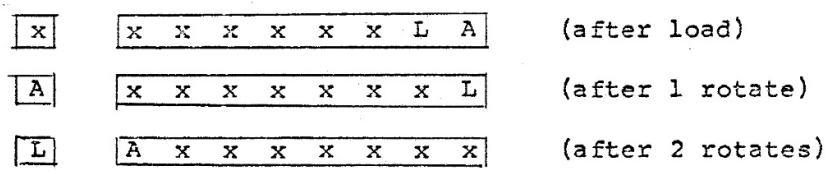
1. The controller's function is to periodically refill a tank (on demand). A fill valve will open whenever a "tank low" switch is activated. The fill valve will close when the "tank full" or "tank near overflow" switches are activated. If the latter occurs, in addition to closing the fill valve, a drain valve will open and an alarm will sound.

Using the above statement of the problem, we can now draw a flow chart. Then we can use the flow chart to write a program in assembly language. Finally, we assemble the program. The figure below defines the function of the individual input and output bits, relative to the LEDs and switches.




INITIALIZE PORT A
READ INPUTS
TANK NEAR OVERFLOW?
Y
N
CLOSE VALVE
OPEN DRAIN
SOUND ALARM
N
TANK LOW?
Y
OPEN FILL VALVE
READ INPUTS
N
TANK FULL OR NEAR OVERFLOW?
Y
CLOSE FILL VALVE

2. Assemble the program which follows. You will have to calculate the relative offsets for two of the branch instructions. Note that in the main loop of the program the input information is loaded and rotated right twice. The figure below shows the results of these operations.




```
$0000  A2 00      START   LDX# $00
$0002  8E 00 17           STX PAD
$0005  CA                 DEX
$0006  8E 01 17           STX PADD
$0009  AD 02 17   MAINLP  LDA PBD
$000C  6A                 ROR A
$000D  6A                 ROR A
$000E  30 09              BMI ALARM
$0010  A9 00              LDA# $00
$0012  8D 00 17           STA PAD
$0015  B0 0A              BCS FILL
$0017  90 F0              BCC MAINLP
$0019  A9 C0      ALARM   LDA# $C0
$001B  8D 00 17   STORE   STA PAD
$001E  4C 09 00           JMP MAINLP
$0021  AD 00 17   FILL    LDA PAD
$0024  09 01              ORA# $01
$0026  8D 00 17           STA PAD
$0029  AD 02 17   TEST    LDA PBD
$002C  29 09              AND# $09
$002E  F0 F9              BEQ TEST
$0030  AD 00 17           LDA PAD
$0033  29 FE              AND# $FE
$0035  4C 1B 00           JMP STORE
```

3. Although this program may be run at full speed, it will be easier to follow if you single step it. What must be initialized before Single Step is operative?

EXPERIMENT 4

SOFTWARE TIMING

In this experiment, we will investigate the possibility of using our knowledge of the time it takes to execute any instruction to generate very precise timing.

1. The 6502 microprocessor chip on the KIM-1 computer board, like most microprocessors, runs off a crystal oscillator. The frequency of the crystal is 1.000 MHz and is quite stable, giving a clock cycle of on microsecond. The exact number of cycles that any instruction takes is documented for the user, and is in fact available on the programming card. Thus, by counting up the number of clock cycles in each instruction, we may determine exactly how long a particular program takes to execute right to the microsecond. Look at the following sample program.

```
                          cycles
            LDX# $64        2     (6x16 + 4 = 100 decimal)
LOOP        DEX             2
            BNE LOOP        3
```

The first instruction, which initializes "X", is executed only once for a fixed 2 microseconds. Each pass through the loop takes (2 + 3) five microseconds. The loop is performed 100 times for a total of 2 + 100(5) = 502 microseconds. Correct? Not quite. On the last pass through the loop, the branch is <u>not</u> taken, and a branch takes only 2 microseconds, if the branch is not taken. Thus the program above takes exactly 501 micro seconds. Now consider the discrete electronic hardware that would be required to produce a precise, temperature stable delay of 500 microseconds. It certainly would be more expensive than the couple of instructions it takes to do it very precisely with software.

2. Now consider the following program. It terminates by jumping back into the KIM monitor. The entry point used is $1C4F.

```
START    LDX# $FF
         STX PADD
         INX
         STX PAD
         LDX# $09
LOOP     INC PAD
         DEX
         BNE LOOP
         JMP MON   ($1C4F)
```

The first four instructions of the program establish Port A as an output, and clear the data in Port A. The "X" register is initialized to the value nine before dropping into a loop in which "X" is used as a counter and decremented to zero. The only action in the loop is iacrementing Port A. Thus after Port A has been incremented nine times (from zero) the program jumps back to the monitor. What should we expect to see if we execute this program? If we add up the number of clock cycles, we will discover that the program executes in less than 120 microseconds, or about one tenth of a millisecond. If we are looking at the six digit display, it will be extinguished only for the tenth millisecond to execute the program before it is again being scanned in the monitor. In fact, with such a short absence it won't appear to even flicker. Similarly, if we look at the discrete LEDs, the number $09 will appear. We will not, of course, see the LEDs count. We can, however, see the program work if we Single Step. There is another way, though. Why don't we just slow down the program with software? In other words, why not insert some code into the loop that will consume enough time each pass through the loop so that we can see the results?

3. Consider the following program:

```
START    LDA# $FF      A9 FF          0000 01
         STA PADD      8D 01 17       02 03 04
         LDA# $00      A9 00          05 06
DISPLAY  STA PAD       8D 00 17       07
         CLC           18
         ADC# $01      69 01
         JSR DELAY     20 0002
         JMP DISPLAY   4C 07 00
```

This program, like the previous example, causes the external LEDs to count. However, note that the count does not terminate, but goes on forever. Also note, that within the loop is an instruction JSR DELAY. This causes us to leave the loop for some fixed amount of time. Assemble the above program at a RAM location of your choosing. You will need to leave spaces for the address of the subroutine called DELAY.

4. A little earlier, we saw that it was an easy matter to generate a half millisecond delay with a three instruction program. Clearly we can put this little program in a bigger loop and generate some number of half millisecond time periods. The program which follows uses this technique to generate 200 half millisecond = 100 milliseconds = one tenth second delay. Assemble this program at a RAM location of your choosing. (You now know what address to fill in, in the program above where you JSR DELAY.)

0200	A0	C8	DELAY	LDY# $C8 (200 decimal)
02	A2	63	LOOPY	LDX# $63
04	CA		LOOPX	DEX
05 06	D0	FD		BNE LOOPX
07	88			DEY
08 09	D0	F8		BNE LOOPY
10	60			RTS

5. Note that the delay produced is a function of the initial values of X and Y. Use different values of X and Y to get longer and shorter delays. What values of X and Y will produce the longest delay.

6. The action in the main loop is produced with the CLC, ADC# instructions. Try substituting these three bytes with other instructions, for example, CLC, SBC#, or RORa, or ROLa. If you use less than three bytes, be sure to use NOPs to fill the gaps. What happens when you use $6A, $66, $66 for those three bytes?

7. Can you verify that the total time delay for the subroutine above is: $T = Y(5X + 6) + 13$?

EXPERIMENT 5

HARDWARE TIMERS - INTERRUPTS

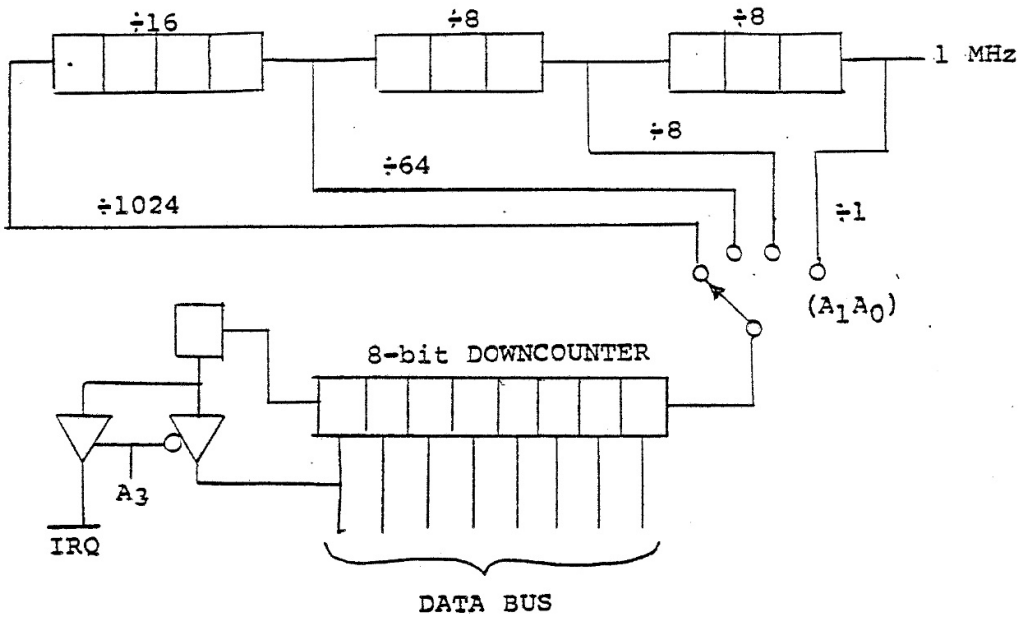
In this experiment we will investigate hardware timers, and use one to run a program on an interrupt basis.

1. Each of the 6530 chips on the KIM-1 has a programmable timer. One is not used by the KIM monitor programs, and like the two ports, is available for user applications. This timer has base address $170X, where X depends on the timer function required. When we are not using the tape dump and load routines from the monitor, there is no reason why we can't use the second timer as well. Thus, everything about the timer in the discussion that follows also applies to the second timer. The only thing that changes is the base address which is 17\underline{4}$X.

2. A string of flip-flops can be so connected to form a binary counter. On the right we see the sequence generated by such a counter of four bits. Note that the least significant bit alternates 0-1-0-1-0-1 etc. The next least significant bit alternates 0-0-1-1-0-0-1-1-0-0 etc. Each flip-flop alternates between one and zero uniformly, however, each succeeding flip-flop changes at half the rate of its predecessor. Thus the input clock is divided in two in frequency by the first flip-flop which is in turn divided by two by the second flip-flop. Thus in a string of N flip-flops, the Nth_N flip-flop is dividing the input frequency by 2^N. For example, three stages can be used to divide by eight, four stages to divide by sixteen etc. This is of course the principle behind an electronic watch. This same principle is used to make rather complex timer chips for use with microprocessors. Such a chip has one or more timers that can be configured in a variety of ways with software.

```
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
```

The timers in the 6530 chips have the structure shown in the figure which follows. Since it is easiest to count events by loading a count and counting down to zero, the heart of the timer is an 8-bit downcounter.



The clock which drives the downcounter is one of four selected rates. The fastest rate is the microprocessor's 1 MHz crystal clock. This clock is also applied to a tapped binary dividing chain, giving clock periods of 1, 8, 64, and 1024 microseconds. The last, and slowest clock is about one milliseconds, giving a maximum timable delay of about one quarter second.

To use the timer, the number of intervals to be counted is written to the downcounter via the data bus and the downcounter begins counting down. When the counter reaches a count of zero, a ninth overflow bit is set. This bit can be used in one of two ways. It can be polled by reading the timer address on the MSB of the data bus, or the bit can be directly used to generate an interrupt. This choice is a function of address bit A_3. Address bits A_1 and A_0 are used to select the clock rate. Thus a total of eleven bits of information are programmed, three bits using the address bus. The table which follows shows what addresses should be used to select the counting rate and interrupt capability. (A more complete description of interval timer operation can be found in Appendix H of the KIM Users Manual.)

Divide Ratio	No Interrupt	Interrupt
1	$1704	$170C
8	$1705	$170D
64	$1706	$170E
1024	$1707	$170F

3. In the programs that follow, two asynchronous counting operations are going on simultaneously. The six digit seven segment display is incrementing at one rate in decimal. The external discrete LEDs are incrementing in binary at an unrelated rate. The rate of the six digit display is determined by software timing, the external display is serviced on an interrupt from the programmable timer. Load in the main program as follows.

```
$0000  A2  40        INIT     LDX# $40
$0002  8E  FE  17              STX $17FE
$0005  A2  00                  LDX# $00
$0007  8E  FF  17              STX $17FF
$000A  86  F9                  STXZ $F9
$000C  86  FA                  STXZ $FA
$000E  86  FB                  STXZ $FB
$0010  CA                      DEX
$0011  8E  01  17              STX PADD
$0014  8E  0F  17              STX TIMER
$0017  58                      CLI
$0018  F8                      SED
$0019  38            MAINLP    SEC
$001A  A2  FD                  LDX# $FD
$001C  B5  FC        ADD       LDAZX $FC
$001E  69  00                  ADC# $00
$0020  95  FC                  STAZX $FC
$0022  90  03                  BCC NEXT
$0024  E8                      INX
$0025  D0  F5                  BNE ADD
$0027  A9  20        NEXT      LDA# $20
$0029  85  80                  STAZ $80
$002B  20  1F  1F    SCANLP    JSR SCANDS
$002E  C6  80                  DECZ $80
$0030  D0  F9                  BNE SCANLP
$0032  F0  E5                  BEQ MAINLP
```

What location affects the timing of the main loop? Change this value to spped up or slow down the main display. Does it affect the external display?

The interrupt from the timer causes the following program to be executed. Note carefully, that the two programs are not contiguous.

```
$0040  48           IRQ     PHA         SAVE A REG
$0041  AD 02 17             LDA PBD     READ SWITCHES
$0044  0A                   ASL A       MULT X 4
$0045  0A                   ASL A
$0046  D0 02                BNE DONE
$0048  A9 FF                LDA# $FF    MAX TIME
$004A  8D 0F 17     DONE    STA TIMER
$004D  EE 00 17             INC PAD     COUNT
$0050  68                   PLA         RESTORE A REG
$0051  40                   RTI
```

Try to determine exactly how the switches on the Port B affect the external counting rate.

Did you remember to install the interrupt vector? That is, did you put the location of your IRQ program in RAM locations $17FE and $17FF?

BINARY, HEXIDECIMAL AND BCD NUMBERS

BINARY	DECIMAL	HEX
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	-	A
1011	-	B
1100	-	C
1101	-	D
1110	-	E
1111	-	F

Addition:

0	0	0	1
0	0	1	1
0	1	1	1
0	1	10	11

SOURCES OF MICROCOMPUTER INFORMATION

General Purpose Magazines

BYTE
70 Main Street
Peterborough, N.H. 03458
$15 a year, monthly, hardware and software

KILOBAUD
Peterborough, N.H. 03458
$15 a year, monthly, hardware and software

6502 Periodicals

User Notes: 6502
P.O. Box 33093
N. Royalton, OH 44133
$13/6 issues, hardware and software for 6502 computers

MICRO: 6502 Journal
P.O. Box 3
S. Chelmsford, MA 01824
$6/6 issues, bimonthly, hardware and software for 6502

Software Sources

6502 Program Exchange
2920 Moana
Reno, Nevada 89509
$.50 for program list

Pyramid Data Systems
6 Terrace Avenue
New Egypt, NJ 08533
Software, tapes, 5/$3.30 (10 min.) 5/$3.60 (20 min)

Micro-Ware Ltd.
27 Firstbrooke Road
Toronto, Ontario
CANADA M4E 2L2
Microchess for KIM, $15, excellent assembler $25

Hardware Sources

Johnson Computer
P.O. Box 523
Medina, OH 44256 line of accesories for KIM, 8K BASIC

Riverside Electronic Design, Inc.
1760 Niagara Street
Buffalo, N.Y. 14207

See also MICRO under 6502 periodicals

See also ads in KILOBAUD and MICRO

Micro Technology Unlimited
P.O. Box 4596
Manchester, N.H. 03108
Dot matrix video display, music board

Pickles & Trout
P.O. Box 11206
Goleta, Ca.93017
Kit $20 to make video monitor out of Hitachi B&W T.V.