6-1987

Making a rainbow workstation for a chemistry lab

John D. Bak
Union College - Schenectady, NY

Making a Rainbow Workstation for a Chemistry Lab

— By —

▷ John D. Bak ◁

◇ ◇ ◇ ◇ ◇ ◇ ◇ ◇ ◇ ◇ ◇ ◇

Submitted in partial fulfullment
of the requirements for
Honors in the Department of Chemistry

Union College
June 1987

ABSTRACT

BAK, JOHN D. *Making a Rainbow Workstation for a Chemistry Lab.* Chemistry Department, June 1987.

Chemical kinetics is the study of how quickly and by what means chemical reactions proceed. Some reactions are so fast and complicated that data must be taken very quickly and then lengthy calculations must be done to get results. Computers speed these studies greatly. The system to be described uses a *DEC-Rainbow* microcomputer as a terminal for a workstation. The system may collect 8k of buffered data at a rate of 1MHz and then upload the data to a VAX for analysis using the same Rainbow as a graphics terminal. The program for data analysis, called KinSim, will then accept the data and also other information about the chemical reactions in the same symbolic format that chemists use so that the analysis may be done. The analysis consists of simulating the chemical reactions from the information provided and compare it with the experimental data that was collected earlier on the Rainbow.

Table of Contents

Making a Rainbow Workstation for a Chemistry Lab

Table of Figures

Making a Rainbow Workstation for a Chemistry Lab

Chapter 1: Overveiw of the Project.

Introduction

This system was specifically designed to collect and analyze data from an instrument called a *flash photolysis spectrophotometer*, but it may also be used with other instruments. The way the "flash rig" works is that the chemicals to be reacted are put in a glass container, and then flashed by high intensity light from a xenon flash lamp or a laser. The length of this flash is typically less than 10 microseconds. Some of the energy of the light is absorbed by the chemicals and causes them to react. The reactions studied by this method may go to completion in as short a time as a hundred microseconds. The reaction is monitored by passing a continuous probe light through the sample cell and measuring the variation in transmitted intensity at a particular wavelength as a function of time. The transmitted light intensity can usually be correlated with the concentration of particular reactants, transients or products in the sample cell. This transmitted light is first converted to an electric current by a photomultiplier tube and then converted to a voltage by passing the current through a known resistance. This voltage is directly proportional to transmitted light intensity at the selected wavelength through the sample cell. This is the input to the interface circuit between the instrument and the Rainbow.

There are two basic needs that we want this system to fulfill. First we want to be able to collect data by computer. Second, we want to be able to analyze this data using the college's VAX cluster. The basic system is an instrument connected to an interface circuit with a buffer which is connected to a Rainbow. The Rainbow acts as the data collection station and also as a VAX terminal for the school's VAX cluster, where data analysis software resides. These two basic components, the data collection system and the data analysis software will be described herein.

Data Collection

System Requirements

There are several characteristics that this system must have in order to be used to study the reactions that we have in mind. The minimum needs are:

- Variable timing: 1Mhz maximum rate
- A 2000 data point buffer
- Greater than 8 bits per data word

- Capability to upload data to the VAX
- Graphics with hardcopy
- Must be inexpensive

The core of our system is a Rainbow microcomputer with a graphics board and an LA50 printer. This gives us our graphics and hardcopy capabilities as well as being a terminal to the VAX, so that data may be uploaded right after it is collected. The analog to digital (A/D) converter in the interface between the instrument and the Rainbow is a HAS1201 made by Analog Devices. This converter has a maximum rate of 1.05MHz at 12 bits per word. We wanted more than 8 bits to give us the sensitivity to make accurate readings in areas where the converter's full range was not being utilized. To get variable timing an 8253-5 programmable interval timer is used. It is configured in such a way that it gives us data sampling intervals ranging from 1μsec to about 8.9 years(more about how that is done later). The interface circuit has a 16k byte buffer which gives it 8k words of storage capacity. Finally an 8251A USART is used to communicate between the Rainbow and the buffer circuit. The cost for the interface circuit was under $1000 (this does not include the price of the Rainbow and printer). The most expensive item was the A/D converter at $512. The final characteristics we ended up with are:

- Variable sampling rate from 1μsec to 8.9 years per data point
- 8k word buffer for data
- 12 bits per data word
- Rainbow graphics and LA50 printer
- Also usable as a VAX terminal
- Inexpensive: less than $1000

Using the System

The system has an 8085 microprocessor which receives commands from the Rainbow and then acts on them. To use it one first sets up the internal registers of the circuit by sending them commands from the Rainbow. The data collection process is started either by sending a command from the Rainbow, or through the remote start input on the circuit itself. The circuit then sends the data it collects back to the Rainbow and the process can start over again.

The circuit operates in three possible modes :

- Buffered operation.
- Real time operation.
- Programmed operation.

Buffered Operation

This mode allows rapid collection rates of up to 1MHz. This is possible because all the data is first stored in the 8k buffer before it is sent to the Rainbow. The number of sets of data to be taken must first be specified. Each of these sets of data will be taken in succession after the start collecting signal is received. This is a very nice feature because the rate at which each set is taken can be different, so that in the beginning data may be taken quickly, but at the end data may be taken more slowly as the reaction slows. After the number of sets of data to be taken is loaded, the periods for each set are loaded. The restriction on the data collection intervals is that each period after the first must be an integer multiple of the period before it. In other words, the period for data set two will equal the period for data set one times the number entered for data set two's timer register; data set three's period will equal the period for data set two times the value entered for data set three's timer register. The number entered for period one is in halves of microseconds−1 and is between 0 and 65535. So if a period of 60μsec is desired, the register is loaded with 119, and if a rate of 1MHz is desired a 1 would be loaded. The next thing to be loaded would be the number of points to be taken for each set. The total number of points taken cannot exceed 8191 because of the buffer size. Finally, the remote start input would be enabled if the start collection signal is to come from outside, or the data collection could be started from the Rainbow. After data collection, the data would be transmitted to the Rainbow where it may be stored on floppy disk.

Real Time Operation

This mode is only usable at lower rates of data collection. Because the data words are 12 bits long, two bytes are required to transmit one word, so the maximum rate of data transfer into the Rainbow is about 600 words per second at the rate of 9600 baud. To use this mode the timer registers are loaded with the period of the sampling rate as before and then either the remote start is enabled or the timer is started from the Rainbow. Now the circuit will send each data word to the Rainbow as it is collected without buffering it. To stop the process, a command to stop is sent from the Rainbow.

Programmed Operation

Since there is an 8085 microprocessor in the interface circuit, small programs may be loaded into the circuit. This option is included for completeness and maximum flexibility.

Commands

The commands that the interface circuit can accept are:

- **Stop and reset**
- **Get 1,2 or 3 sets of data**

Set up to take buffered data at one, two or three different rates

- **Set up for real time collection**
- **Get one data point**
- **Load timer register 1,2 or 3**

  Loads the timer registers. Each register is 16 bits

- **Load count registers 1,2 or 3**

  Loads number of points to be taken with each rate for the buffered data. Total cannot exceed 8191.

- **Select timing from counter 1,2 or 3**

  When taking real time data, selects which timer the timing will be taken from.

- **Start/Stop Timer**

  Starts or stops timer without waiting for remote start.

- **Allow/Disallow remote start**
- **Load Temporary program**

  May load a temporary program. Up to 15 may be loaded as long as they fit into the memory restrictions.

- **Run Temporary program.**
- **Run diagnostic tests**
- **Load Status Register**

  This is an important register with hardware switches. More will be explained later.

Hardware Design

The interface circuit itself is a small microprocessor system with DMA (Direct Memory Access) capabilities for storing data from the A/D converter. The CPU is an 8085A and is used to control the states of the circuit and also to generate a 2MHz system clock from which all the timing is derived.

The A/D converter is a HAS1201 made by Analog Devices. This unit has a 1.05MHz maximum conversion rate, internal track and hold circuitry and 12 bit resolution. The one drawback to this unit is that there is no end of conversion signal. This is gotten around by tying the start conversion signal and the register strobe together so that the start conversion signal is also used as a pseudo end conversion signal. This does mean that the output will be delayed one period of the clock but this is only a minor problem that can be compensated for by programming.

The system's DMA control circuitry consists mainly of two sets of four 74LS191 presettable up/down counters. The first set is operated in count-up mode and is used to generate the addresses for DMA operations. When the CPU is held, the outputs of these counters are put on the address lines and they are incremented every time there is a start conversion signal. The other set is preloaded with the number of data points to be collected and then counts down wit each start conversion signal until it reaches zero and then it interrupts the CPU.

The system has four memory chips. The first one is an 8k EPROM with the operating system on it. The next is an 8k RAM for variables, the stack and any temporary programs that might be loaded. The last two are set up for DMA operations to be performed on them. One chip stores the most significant byte (MSB) and the other stores the least significant byte (LSB) of the A/D converter's data word. These two chips share the same address space when the CPU is held, but not the same data bus. When the CPU is not held, the data bus is rejoined into one peice and the two memory chips are moved into seperate address spaces.

The USART (Universal Synchronous/Asynchronous Receiver/Transmitter) is the circuit's link to the outside world. The data received pin of the chip has been tied to the RST5.5 interrupt on the 8085A, so that whenever a command is received, it will be able to get the CPU's attention. This allows the chip to be stopped in the middle of an operation.

The timing circuitry is the most complex part of the circuit. The heart of the circuitry is the 8253-5 programmable interval timer. This chip has three 16-bit gated, presettable repeating down counters. The outputs of these counters are tied to the clock input of the next counter, with the first tied to the 2MHz system clock. Timer one has a 2MHz input, timer two gets its input from timer 1 and timer 3 gets its input from timer 2. This gives the chip the range of a 48 bit counter, but more flexibility, because the outputs of these counters are also multiplexed so that the counter that the start conversion signal is derived from may be chosen from among the three. The fourth input to the multiplexer is taken from the device select logic for the A/D converter, so that a conversion may be started by the CPU directly. The gates for the counters are active high and the flip/flop that controls this may be set either from the remote start signal, or from a bit in the status register. The flip/flop is reset when a terminal count is reached in the DMA counter for the number of points to collect, or when it is reset by setting the status register bit to zero. The final piece of the timing circuitry is the status register. This important register is used to set the modes of operation for the circuit. Its contents are as follows:

Bit:	Use:
0,1:	Used as address for multiplexer to get start conversion source.
2:	1 allows remote start for timer,
	0 remote start will not start timer.

3: Start timer

4: DMA/$\overline{\text{CPU}}$.

5: Remote start mask: 0 blocks remote start completely.

6,7: Unused

Bits 2 and 5 work in conjunction with each other where bit 5 will block the remote start signal completely, but when it is set to allow the signal in, bit 2 will select what happens with the signal, a 1 starting the timer and a 0 interrupting the CPU. Bit 3 will start the timer regardless of what else is on. Bit 4 selects what happens when the timer starts. If it is a 1, then the CPU is held and a DMA operation will be performed to collect the data, otherwise the start conversion signals will also be sent as CPU interrupts so the CPU will be able to collect the data from the A/D converter directly. Finally bits 0 and 1 are used to encode the multiplexer, a 00 being the device select for the A/D converter and the other numbers being the respective timer registers on the 8253-5 timer.

Buffered Data Collection Process

The multiple data sampling rates of the buffered data are achieved through programming. First the timer registers are each loaded. The first timing rate is put into timer register one, the second into two, and the third into three. Then the number of points to be taken is loaded into the DMA register and the DMA address register is set to the beginning of the DMA memory. Next the status register is loaded to allow remote start on the timer and to select DMA operation and finally start conversions from timer 1. When the remote start comes, the data will be collected until the end count is reached and the CPU will be interrupted. The end count register will now be loaded with the number of points to be collected at the second rate and then the status register will be loaded. DMA will be selected with the start conversions from timer 2, but this time the *start timer* bit will be set so that as soon as the register is loaded the timer will start and the CPU will be held again. Now data collection will proceed and when the terminal count is reached, the CPU will be interrupted again. Now if a third set is to be taken the process will be repeated but with data for the third collection rate. The neat part of this system is that the address register is only loaded at the beginning of the process, so that it will continue to be incremented as the data is taken, but the first point taken in a set of data will be right after the last point taken in the previous set because the register still contains the old number. After the last set is taken the data will be sent out to the Rainbow.

Data Analysis

The program that was written to analyze the data is in **FORTRAN-77** and uses routines from **IMSL** to integrate the equations and **RGL** (**R**eGIS **G**raphics **L**ibrary) to generate the

graphics output on the Rainbow.

Some Basic Chemistry

In order to understand the program it is necessary to first understand how equations for the rates of chemical reactions are derived. First, all chemical reactions can be broken down into a series of steps which describe the reaction. These steps describe the interactions between each of the species in the reaction. For example, take the following reaction:

$$A + B + C \longrightarrow D + E$$

This equation means that reactants A, B and C combine to give products D and E. This might be broken down into the following mechanism:

$$A + B \xrightarrow{k_1} F$$

$$F + C \xrightarrow{k_2} G$$

$$G \xrightarrow{k_3} D + E$$

$$F \xrightarrow{k_4} A + B$$

$$G \xrightarrow{k_5} F + C$$

$$D + E \xrightarrow{k_6} G$$

$$A + C \xrightarrow{k_7} H + I$$

Reactions 1 to 3 are the basic mechanism which gives us the products, but there are other processes which also occur. Reactions 4 to 6 are the reverses of 1 to 3, and 7 is a reaction that uses the reactants up, but does not contribute to the reaction of interest; this is a competing or side reaction. All of these things must be taken into account when developing chemical mechanisms. The k's over the arrows are called the rate constants. This is a proportionality constant that helps tell how quickly each step of the mechanism proceeds with respect to the others.

These mechanism steps are important because they can be easily converted into differential equations showing the rate that the step proceeds at. There are four types of mechanism steps:

$$A \xrightarrow{h\nu} products$$

$$A \longrightarrow products$$

$$A + B \longrightarrow products$$

$$A + B + C \longrightarrow products$$

These may be converted to a rate equation as follows:

$$v = \Phi I_a$$

$$v = k[A]$$

$$v = k[A][B]$$

$$v = k[A][B][C]$$

In these equations the v is the rate of each step in the mechanism above. The first equation is a photochemical reaction so it is different from the rest. Here the rate is proportional to the amount of light absorbed (I_a) and the *quantum yield* (Φ) which is the ratio of the number of molecules that react to the number of photons of light absorbed. The other ones are easier. The rate of the step is proportional to the product of the concentrations (denoted by the square brackets) of the reactants. Since the rate for each step is proportional to the concentration of the reactants, the rate constant converts this proportionality to an equality. The rate constants are very important quantities because a mechanism step does not depend on what mechanism it is in. If the rate constant for a particular mechanism step can be found in one mechanism it will be exactly the same if that mechanism step is found in an entirely different chemical reaction.

Now that we have the rates for each step in the mechanism we can find the rate of change of each species in the reaction. This is done by adding together the rate of all steps where the species is formed and subtracting the rate of all steps where the species is used up. For example, take this sample mechanism:

$$A + B \longrightarrow C + D$$
$$C + E \longrightarrow F$$
$$F \longrightarrow C + E$$
$$C + D \longrightarrow A + B$$

With rate equations:

$$v_1 = k_1[A][B]$$
$$v_2 = k_2[C][E]$$
$$v_3 = k_3[F]$$
$$v_4 = k_4[C][D]$$

From this we can see the rates of change for the some of the species in the reaction would be:

$$\frac{\partial[A]}{\partial t} = -v_1 + v_4$$
$$\frac{\partial[F]}{\partial t} = v_2 - v_3$$
$$\frac{\partial[C]}{\partial t} = v_1 - v_2 + v_3 - v_4$$

These are a set of coupled non-linear differential equations which can be numerically integrated, using the Gear method because of the sizes of the terms involved, to give concentration versus time for each species in the reaction.

Now that concentration versus time data can be generated for a given mechanism and set of rate constants, we can compare this with the experimental data for the same reaction. When we find agreement between the two this shows that we have discovered a plausible mechanism

and the corresponding set of rate constants. This approach is used in the program to solve for the mechanism and rate constants. One puts in a proposed mechanism and rate constants, the computer integrates the differential equations corresponding to that mechanism and then plots both the experimental data and the calculated data on the screen. The mechanism and rate constants can then be modified until there is agreement between the two.

Running the Program

In order to use the program, the experimental data must first be loaded into VAX data files. For photochemical reactions, a VAX file must also be created giving the light intensity versus time profile for the incident light. After this is done the program may be run. It is menu driven to make it easy to use. A mechanism, a set of rate constants, and the initial concentrations are entered into the program. They may be saved for later use. The program will then automatically calculate the differential equations so that they may be integrated. This is an improvement over other kinetic simulation programs where the differential equations are coded into subroutines of the program and every time the mechanism is changed that subroutine has to be rewritten. When the program is run, it will plot both the experimental and simulated concentration versus time data for any chemical species in the reaction. Changes may be made to the mechanism and the data may be replotted, all interactively, until agreement is reached between the experimental and simulated data. The program also gives hardcopy of the graphs and the mechanism, as well as tabular results of the data.

How the mechanism is stored

The data structure that was developed for storing the mechanism is interesting. It consists of a series of arrays where *maxm* is the maximum number of steps that can be stored in the mechanism:

`rtype(maxm)`:	Type of reaction step.
`nlhs(maxm)`:	Number of reactants in step.
`lhs(4,maxm)`:	Internal code for each reactant in the step.
`nrhs(maxm)`:	Number of products in step.
`rhs(4,maxm)`:	Internal code for each product in the step.

The data structure is simple and it allows a general routine to evaluate the velocities (v) for each step as follows:

```
      function evalv(eqn, j, i,t ,c)
      implicit none
      integer eqn, j, lp
C eqn is the step being evaluated
      double precision t, c(j), i
```

```
C t is current time, c(j) is concentration of each species in the reaction
      external i
C evaluates light intensity at time t
      include'commondef.for/nolist'
C data structure for mechanism
      goto(10, 20, 30, 40), rtype(eqn)
      write(6,*)' Error, no mechanism step'
      write(6,*)'type ',rtype(eqn)
      stop
   10 evalv= k(eqn) * i(t)
      return
C photochemical
   20 evalv= k(eqn) * c(lhs(1,eqn))
      return
C single reactant
   30 evalv= k(eqn) * c(lhs(1,eqn) * c(lhs(2,eqn))
      return
C two reactants
   40 evalv= k(eqn) * c(lhs(1,eqn) * c(lhs(2,eqn) * c(lhs(3,eqn))
      return
C three reactants
      end
```

This function will return as its value the velocity of the mechanism step specified. It may be called by another routine (see below) that evaluates the first derivative of the function called by the DGEAR integration routine in IMSL. This subroutine is defined as follows:

```
      subroutine evmech(j, t, c, dc)
      integer j, q, r
      double precision t, c(j), dc(j), v, evalv
C c(j) is concentration of each species
C dc(j) is the first derivative returned by this routine.
C t is current time
      external evalv
C function for velocity of mechanism step
      include 'commondef.for/nolist'
```

```
C mechanism and other information
      do 5 q=1,j
    5   dc(q)=0.0d0
C zero array
      do 10 q=1,m
        v=evalv(q, j, t, c)
C get rate of current mechanism step
        do 20 r=1,nlhs(q)
   20     dc(lhs(r,q)) = dc(lhs(r,q)) - v
C subtract velocity from 1st der.  of each specied being reacted.
        do 30 r=1,nrhs(q)
   30     dc(rhs(r,q)) = dc(rhs(r,q)) + v
C add velocity to 1st der.  of each species produced by step.
   10 continue
C do this for each step in mechanism
      return
      end
```

This subroutine will systematically go through the entire mechanism and adjust the rate of change for each species in the reaction by the velocity of each mechanism step that it appears in.

Conclusion

This system is a complete system for collecting and analyzing kinetic data in a chemistry laboratory. The rate of data collection is variable up to 1MHz so a wide variety of systems may be explored. The data analysis gives chemical mechanisms and rate constants. All of this may be done from the same DEC Rainbow in the laboratory.

We are currently using this system at Union in a variety of research projects, most notably studies on the photochemical kinetics of alkyliodides and organometalic hydrides. The system is also being integrated into the physical chemistry laboratory to provide students with some experience with chemical reaction simulations.

Chapter 2: Documentation for the Interface Circuit.

The interface circuit is a small dedicated microprocessor system with the following characteristics:

- 8085A microprocessor.
- HAS1201 A/D converter.
  - 1.05MHz conversion rate.
  - 12 bit data word.
- 8251A USART (Universal Synchronous/Asynchronous Receiver—Transmitter) serial communication via RS232 line at 9600 baud.
- 8253-5 Programmable Interval Timer.
- DMA data collection system.
  - Initiated by an outside signal or by 8085A.
  - 16k byte buffer (holds 8k words) from A/D converter.
  - Variable collection rate, from 1μsec to about 8.9 years per data point.
- 8k EPROM with operating system.
- 8k RAM for system variables and stack and other data.

The circuit is used by sending commands over the RS232 line to the 8085 which will in turn respond. The commands set up different registers for different modes of operation in the system. There are two basic modes of operation and a third included for completeness. They are:

- Buffered data collection mode.
- Real time data collection mode.
- Programmed mode.

Buffered mode collects data using DMA operation, and then will send out the data via the RS232 port. Real time collection will have the timing elements interrupt the 8085A and it will then take a data point and send it over the RS232 line immediately. This mode is limited in speed to the rate at wich data may be transmitted over the RS232 line – about 600 samples per second. In programmed mode, a program in 8085 machine language is loaded into the circuits memory and is then executed. This is included for maximum flexibility and for completeness.

The circuit receives these commands,sent from the host computer via the RS232 port, in the format like so:

$$< 1\ byte\ command > [n\ byte\ argument]$$

The command is always one byte, and can be followed by any number of bytes for an argument, which depend on the particular command.

The commands that the system is programmed to respond to are:

Commands Sent to Interface

General Commands

Cmd Action

00 Stop and Reset.

01 Load Status Register.

→ + 1 byte value.

02 No Register Reset Mode.

03 Register Reset Mode.

04 Start Timer.

05 Stop Timer.

06 Use Timer 1

07 Use Timer 2

08 Use Timer 3

09 Get 1 Data Point.

0A Allow Remote Start.

0B Inhibit Remote Start.

0C Not Used

0D Dump DMA Memory.

0E Complete Memory Dump.

0F Run Diagnostic Program.

Program Mode Commands:

10-1F Load Temporary Program 0 to F.

→ + length of program (2 bytes) + starting location (2 bytes) + program.

20-2F Run Temporary Program 0 to F.

Buffered Operation

30 One set of data with timer 1.

→ + 2 byte value for timer 1.

→ + 2 byte value for number of points to be collected.

31 One set of data with timer 2.

→ + 2 byte value for timer 1 + 2 byte value for timer 2.

→ + 2 byte value for number of points to be collected.

32 One set of data with timer 3.

→ + 2 byte values for timers 1,2 and 3 (6 bytes).

→ + 2 byte value for number of points to be collected.

33 Two sets of data with timers 1 and 2.

→ + 2 byte values for timers 1 and 2 (4 bytes).

→ + 2 byte value for number of points to be collected for each set(4 bytes).

34 Two sets of data with timers 1 and 3.

→ + 2 byte values for timers 1,2 and 3 (6 bytes).

→ + 2 byte value for number of points to be collected for each set(4 bytes).

35 Two sets of data with timers 2 and 3.

→ + 2 byte values for timers 1,2 and 3 (6 bytes).

→ + 2 byte value for number of points to be collected for each set(4 bytes).

36 Three sets of data with timers 1, 2 and 3.

→ + 2 byte values for timers 1,2 and 3 (6 bytes).

→ + 2 byte value for number of points to be collected for each set(6 bytes).

Real Time Operation

40 Collect data using timer 2.

→ + 2 byte values for timers 1 and 2 (4 bytes).

41 Collect data using timer 3.

→ + 2 byte values for timers 1,2 and 3 (6 bytes).

Each command is sent in by the RS232 line, which interrupts the CPU so it can respond to the command. The command is then followed by the arguments, if any.

The memory map of the system is:

0000-1FFF:	EPROM - Operating System.
2000-3FFF:	RAM - Stack, variables and temporary programs.
4000-4FFF:	Status Register.

5000-5FFF:	8253-5 Timer.
5000:	Counter 1.
5001:	Counter 2.
5002:	Counter 3.
5003:	Control word.
6000-6FFF:	DMA end count register.
6000:	LSB
6001:	MSB
7000-7FFF:	DMA address register.
7000:	LSB
7001:	MSB
8000-9FFF:	HAS1201 A/D converter.
8000:	LSB
8001:	MSB
A000-BFFF:	8251A USART.
A000:	Data word.
A001:	Control word.
C000-DFFF:	LSB RAM buffer.
E000-FFFF:	MSB RAM buffer.

None of the components listed above requires its full address space other than the memory chips, and any addresses not specified will only contain multiple images of the component listed. This is because the addresses are not completely decoded to simplify the device select logic.

The Status Register is a software loadable register with hardware switches.

Bit:	Use:
0,1:	Used as address for multiplexer to get start conversion source.
2:	1 allows remote start for timer,
	0 remote start gives a CPU interrupt.
3:	Start timer
4:	DMA/$\overline{\text{CPU}}$.
5:	Remote start mask: 0 blocks remote start completely.
6,7:	Unused

Bits 2 and 5 work in conjunction with each other where bit 5 will block the remote start signal completely, but when it is set to allow the signal in, bit 2 will select what happens with the signal, a 1 starting the timer and a 0 interrupting the CPU. Bit 3 will start the timer regardless of what

else is on. Bit 4 selects what happens when the timer starts. If it is a 1, then the CPU is held and a DMA operation will be performed to collect the data, otherwise the start conversion signals will also be sent as CPU interrupts so the CPU will be able to collect the data from the A/D converter directly. Finally bits 0 and 1 are used to encode the multiplexer, a 00 being the device select for the A/D converter and the other numbers being the respective timer registers on the 8253-5 timer.

The interrupt structure of the system is:

TRAP Terminal Count in DMA cycle.
RST 7.5 Start Conversion from data collection timing.
RST 6.5 Remote Start signal.
RST 5.5 USART - received data.
INTR not used.

System Construction

The block diagram of the system is as follows:

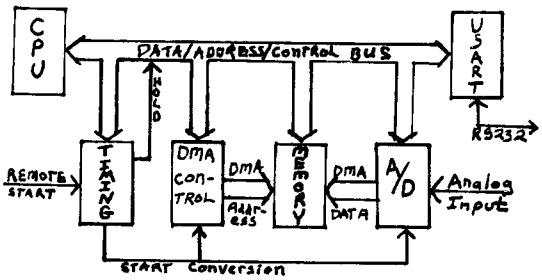

Figure 1.

After the timing elements have been properly loaded, start conversion signals may be generated from the timing circuitry. These start conversion signals are use by the system to start conversions on the A/D converter, to generate CPU interrupts and also to drive the DMA circuitry.

Timer system

The primary element of the timer system is the 8253-5 programmable interval timer. The timers are set up in mode 2 (rate generator). The clock input of timer 1 is tied to the 2MHz system clock, which is generated by the 8085 from a 4MHz crystal. The clock input of timer 2 is tied to the output of timer 1. The clock input of timer 3 is tied to the output of timer 2. The outputs of each of these timers is also put into a multiplexer and the start conversion signal is taken from this multiplexer. In effect, the multiplexer is used to expand the size of the timer. If it is set to the first timer the size is 16 bits, the second would be 32 bits and the third 48 bits. It is not quite the same as having a normal 48 bit counter because each timer counts the number of times the previous timer has rolled over, but the range is still there. The fourth input to the multiplexer is taken from the device select logic for the LSB of the D/A converter so that the CPU may generate start conversion signals (*e.g.* the get one data point command).

The status register is the final element of the timer system, it is also shared by the interrupt and DMA systems. The status register bits 0 and 1 are the addressing for the multiplexer. A 00 in the first two bits selects the device select logic and a 01 timer 1, a 10 timer 2 and 11 timer 3. The status register also controls how and when the gate to the timer is turned on. The gate input to the timers allows the timers to count when it is set high, and inhibits counting when it is set low. The gate is taken from a flip-flop that is set when bit 3 is set, or when bit 5 is set and a remote start signal comes in from the instrument. The output from the flip-flop is then ANDed with bit 2, and this is taken as the gate signal. The gate signal is then ANDed with bit 4 to give the HOLD request to the CPU. NOT bit 4 is also ANDed with bit 5 and the remote start signal to give the RST6.5 signal, so if the remote start signal is enabled, and the timing is on, then the CPU will either be interrupted or held. The flip-flop is reset by the Terminal Count signal from the DMA circuitry, or by setting bit 3 to zero. The terminal count signal is also used for the TRAP interrupt if the CPU is held.

This is the schematic of the timing system.

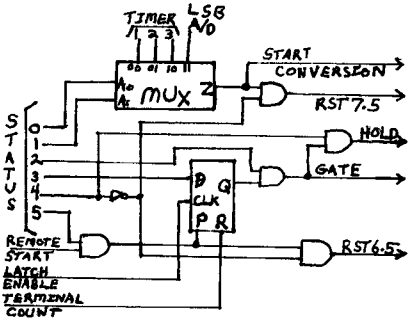

Figure 2.

The DMA system uses the previously described system for two things. First, the HOLD signal for the CPU is taken from the GATE signal, and second, the start conversion signals are taken from the multiplexer. When the CPU is held, the DMA system is turned on by the HLDA signal from the CPU. This allows the two sets of counters to begin counting. the first set of counters is the DMA address registers. These counters will increment for each start conversion signal sent on the line. The outputs of these counters are put on the address line as long as the HLDA signal is high. The second set is the DMA end count register. This register will decrement each time a start conversion signal is sent. When this register reaches zero, the terminal count signal will be sent which will interrupt the CPU and also reset the gate flip-flop. The other thing that the HLDA signal does is to separate the data bus into two sections. The first section is connected to just the most significant 4 bits of the A/D converter and one of the memory chips. The second section is connected to the rest of the system. The other thing that happens is that the output from the A/D converter is turned on, and the memory chips are write enabled. So every time a start conversion signal occurs, the last data word from the A/D converter is put on the line, and the address is incremented. The data on the bus will be stored in the address selected by address lines 0 to 12 of the address bus in both of the memory chips, but since there are two separate sections on the data bus, there is no data clash and the MSBs of the A/D word are stored in one memory chip, and the LSBs in the other. The data may then be extracted one byte at a time after the CPU is released.

As stated before, CPU interrupts are used for real time collection. The CPU is interrupted on the remote start signal, and then each start conversion signal will interrupt the CPU again so it

can get the data word that the A/D converter is producing. The data is then directly sent out via the USART to the controlling computer.

More on buffered operation

A major portion of the abilities of the buffered operation mode is achieved by programming. In buffered operation it is possible to change speeds in the middle of collecting data up to three times. This is accomplished by preloading these times into the 8253-5 timer and then changing the value of the multiplexer address after a certain amount of time. After the time values are pre-loaded into the timer, the number of points to be collected at each rate is then stored in the CPU registers and the first is loaded into the count register. The address register is then set to the beginning of the DMA buffer, and the status register is loaded to allow DMA operation, and either the timer is started or the remote start is enabled. When the first set of data is collected, the count register is loaded with the number of points to be taken at the second rate and the status register is loaded, selecting the next timer the rate is to be taken from, and also starting the timer. After the second set of data is taken, a third may be taken in a similar manner. After all the data is taken, it will then be sent out via the USART to the controlling computer.

Analog Amplification

The analog signal is assumed to be in the range of about 0 to 50mV. It is then put through an op-amp to amplify the signal to about a 0 to 10V signal. There is a variable gain on the op-amp feedback loop to allow for adjustment and there is also a baseline and gain adjustment between the op-amp and the A/D converter to allow a ±10% full scale adjustment and an offset adjustment of ±5%. The diagram of the input amplification circuit is:

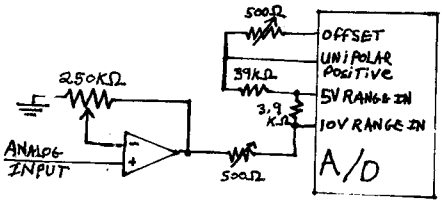

Figure 3.

Components of the system

The parts of the system is constructed from are as follows:

1 HAS1201 A/D converter.
1 1489 Quad receiver.
1 1488 Quad driver.
1 555 multivibrator.
1 AD-OP37 op-amp.
4 74LS373 octal latches.
3 74LS245 octal buffers.
1 8251 USART.
3 HM6264LP-12 8k RAM.
1 HN482764K-2 8k UV-EPROM.
4 74LS08 Quad AND gates.
3 74LS32 Quad OR gates.
1 74LS00 Quad NAND gate.
1 74LS08 Hex inverter.
8 74LS191 presettable up/down counters.
1 74LS153 Dual 4 input MUX.
1 8253-5 programmable interval timer.
1 74LS138 3 input decoder.
1 8085A microprocessor.
2 500Ω trim pots.
1 250kΩ trim pot.
1 10kΩ trim pot.
1 3.9kΩ resistor.
1 39kΩ resistor.
1 1kΩ resistor.
1 4.00MHz crystal.
2 20pF capacitors.
1 1000pF capacitor.
Assorted leveling capacitors.
1 35V ct transformer.
1 12.6V ct transformer.
1 LM232K +5V regulator.
1 L7815 +15V regulator.
1 L7805 −5V regulator.
1 L7815 −15V regulator.

8 Diodes.

8 1000μF capacitors.

References:

MCS-85^{TM} User's Manual. Intel Corporation ©1978.

Circuit Diagram

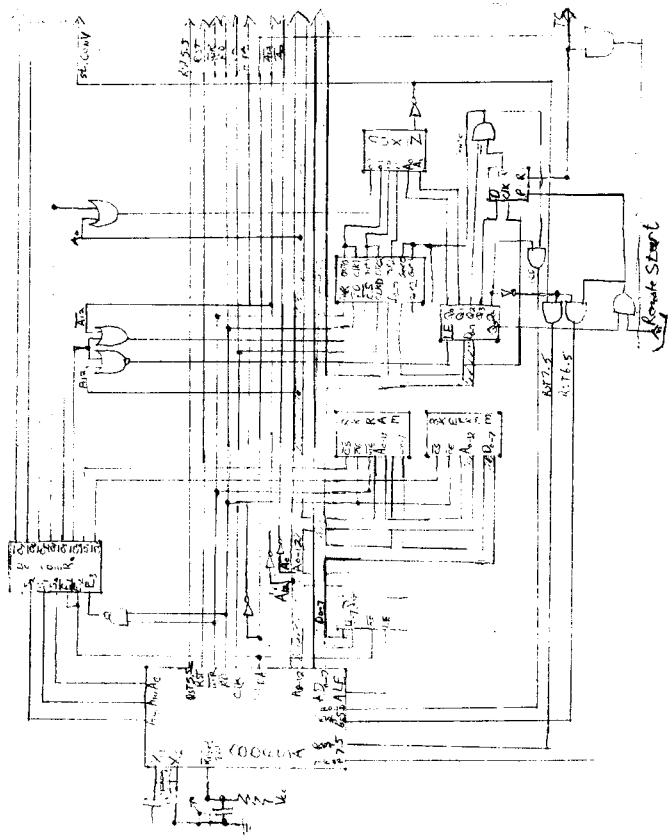

Figure 4

Circuit Diagram(cont)

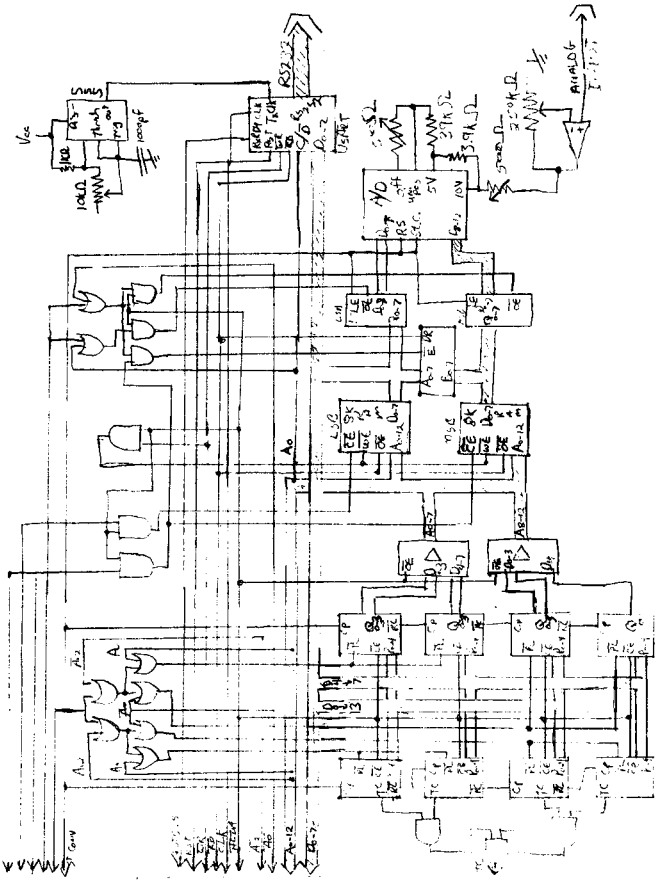

Figure 4 (cont)

Chapter 3: Documentation for the KinSim Program.

Introduction:

KinSim is a software package designed to allow you to simulate a chemical reaction. The input to the program consists of a trial mechanism written in symbolic form, (*e.g.* A + B $\longrightarrow$ C + D) and a set of trial rate constants (and branching ratios for photochemical steps). The program will calculate and plot as a function of time the concentration of any species in the system. Superimposed on this plot will be the corresponding experimental versus time data for this same species. Congruence of the simulation and experimental curves suggests the validity of the trial mechanism and rate constants.

One of the most attractive features of this package is that all the inputs are prompted for and, because the differential equations are constructed internally by the program, there is no need to change the program each time the mechanism is changed. The program is written in FORTRAN-77, so if there is a problem that cannot be dealt with by the existing code any good hack (or even a bad hack) should be able to patch in the required code. To facilitate this, we tried to keep to the ideal of structured programming (quite a task in FORTRAN) while developing the program.

Before using this package, the user will be required to create several data files. The first contains the data for photolyzing light intensity as a function of time. The other files contain experimental concentration versus time data.

We have not put any units on the output of the program to allow you freedom to choose whatever system of units suits you best; however, this also puts the burden of checking the units on *your* shoulders, because the computer does not care what units you use, it is just as happy to crunch the numbers the wrong way as the right way. Due to the limitations of the storage of floating point numbers in the FORTRAN language, it is best to choose units so that rate constants have values near one (*e.g.*, in a range of about $0.1E-04$ to $0.1E+05$) to avoid round off errors during integration of the differential equations.

This system was specifically designed for use on a DEC-Rainbow using the ReGIS (Remote Graphics Instruction Set) VT125 emulation package, but any VT125 emulation should work. The packages RGL (ReGIS Graphics Library) and IMSL are also used on the mainframe to do the integration and generate the graphics screens produced by the program.

We would like to thank Union College for providing the funds and resources for developing this package through the Undergraduate Summer Research Program.

Running KinSim

The first thing that needs to be done is that your experimental data files and your light intensity data file must be put onto the system. Both types of files have the same format, and are put into standard FORTRAN text files.

The first line of each file is the number of points in the file. It must be between 1 and 1000 (this can be changed by changing the parameter *maxdata* in the program code)

The file must then contain the values of *time* on the next line and either *concentration* for an experimental data file, or the adjusted value of *intensity* for the light intensity file. The maximum light intensity should be 1.0000 and the rest of the values should be expressed as a fraction of the maximum. The constant *Scale Factor* takes into account the amount of light that is actually absorbed by a particular molecule in the reaction. Then the time and data values alternate from line to line until all the data is put into the file like so:

number of points
time one
data one
time two
data two
.
.
.
time n
data n

The time and data points are double precision and the number of points must be an integer. You are allowed to have up to three experimental data files included in the calculation at one time.

When **KinSim** is interpreting the data for the light intensity, it will use the value of the first data point when the current time in the simulation is less than the time value of the first data point, and when the time value is greater than the last data point, it will use the last data point as the light intensity; otherwise it uses a linear interpolation between points as the value.

Now all that needs to be done is run the program. Type **RUN KinSim** or whatever is appropriate on your system and we are on our way! The first thing that you will get is a menu with ten options (it will help if you are actually running the program now so you can see what is going on better) that will look something like this:

```
1.  Load Mechanism File
2.  Load Flash profile file
3.  Load experimental data files
4.  Integrate and graph
5.  Integrate and make a table
6.  Change constants
7.  Change Mechanism
8.  Change initial concentrations
9.  Save mechanism and constants
0.  Exit program

Your Choice?
```

You can now type in any command you wish to use; however, there are some commands that require information from other commands before they can do what they are supposed to do. If you try to use one command before you use another that gives it the required information, the program will just go back to this menu without executing the command you asked it to do, until you have the information that the command needs, at which point you will be allowed to go on.

You will need to use command 7, or **change mechanism**, first to enter the mechanism into the program and then command 6, or **change constants**, to enter the scale factor and the infinity time value, and finally command 8, or **change initial concentrations** to set the number of initial concentrations, and what the initial concentrations are. You should then use command 9, or **save mechanism and constants**, to save what you have done so that you can use command 1, or **load mechanism file**, to do this all quickly, rather than go through the process of entering the mechanism every time.

Now you may finally get to what you wanted to do in the first place, simulations. You should now use commands 2 and 3 to load the experimental data and the light intensity from the files and then you may use either commands 4 or 5; these will do an integration and then plot out a graph, or put up a table showing experimental data contrasted with the calculated data. You will also be asked if you want a hardcopy of the data. The graph option will plot up to three sets of simulations and experimental data on the same set of axes for you. You may change the k's for the reactions and do the integrations again until the simulation plots are congruent with the experimental ones.

When you finish you should use command 9 again to save the current mechanism and constants on the disk, so that the next time you may pick up where you left off.

Detailed explanations of commands:

1. Load mechanism file:

This simple command will load a previously saved mechanism, scale factor, and sets of initial conditions.

To use the command type a **1** at the main menu and then the computer will say:

```
File with mechanism:
```

You should then type in the name of the file that you saved the mechanism under and then the computer will load in the file. If there is an error, the computer will tell you and ask for the file name again.

2. Load Flash Profile

This command will load in the light intensity versus time data from a file.

To use this command you should type a **2** at the main menu. The computer will then respond with:

```
File with flash profile data is:"NONE "
Name of file with flash profile data?
```

You should then tell the computer the name of the file with the flash profile data. If there is an error the computer will go back and ask for the name of the file again.

If you use this command a second time, the response will change slightly. The computer will tell you the name of the old file and then ask you if you want to change it. If you type yes, it will then ask for the new file name as before.

3. Load experimental data

This command is almost exactly like the previous one, except that it will load in the experimental concentration versus time data. Up to 3 sets of concentration versus time data may be read in corresponding to different starting concentrations of reactants. This feature is provided for use by those people doing multiple flash photolysis measurements on the same sample.

4. Integrate and Graph

This command will take all the information you have provided and then plot out the experimental data and the calculated data on a set of axes and also generate a hardcopy if you desire. To use this command you should type a **4** at the main menu, and then the computer will respond with:

```
Species to monitor:
```

You should type in the species that you wish to graph the concentration of (usually the same one that you collected data for). After that the computer will ask:

```
Do you want to graph concentration set 1?
```

You should type in **Y** or **N**. The computer will ask this for each set of initial conditions you have. The computer will now get the time window to integrate over by using the smallest and largest times in the experimental data. After that the computer will search the experimental data for the largest and the smallest concentration values, and when it gets them it will add 10% of the difference between them to either side to get the default range for the concentration on the graph, and will say:

```
Window for concentration data
Min:  -0.12345E-3 Max:  6.7890iE-2
Input new values (Negative to keep these)
Values:
```

You should now enter either two negative numbers separated by commas(if you like the range that is shown) or two new numbers (that are not negative) to set a new window. Now, finally, you will see a graph being constructed on the screen, with time on the horizontal axis, and concentration on the vertical axis. There will also be up to three sets of data, with boxes for experimental data set 1, circles for experimental data set 2, and triangles for experimental data set 3. Little dots will also start to appear. These are the calculated values, and there will be one line of dots for every set of experimental data that has been plotted. If the calculated values go out of the range of the graph, they will not be plotted. When an integration is completed, the infinity time value will be printed at the top of the screen. Finally, when it finishes, the computer will ask if you want to print the screen. If you type **Y** the graph will be printed.

When this is done, the computer will type out:

```
There are 20 mechanism steps
Which step's constants do you want to look at?
(?  for all, or numbers and subranges separated by commas)
```

Now if you want to change any constants or branching ratios you may do so by typing in the number of the steps you want to change, or return if you don't want to change any. For instance, if you wanted to change the first four constants, the seventh and the nineth you would type in "1-4,7,9" and the computer would display each one of those to you in turn and ask if you wish to change it like so:

```
Step # 1
   A9 = 0.670000000D+00
Change?
```

If you did want to change it, type **Y** and then the computer would ask for the new value. When the computer finished with your list, it will ask if you wish to change more constants, if you type in **Y** it will go back to the point where it asks you to type in a list of constants to change, otherwise it will ask:

```
Integrate again(I) or go back to menu(M)?
```

If you type **I** the computer will go back to the graph and remove the old calculated values, and then plot up new ones, otherwise it will take you back to the main menu.

5. Integrate and make a table

This command will print out a table with concentration versus time for three of the species in your reaction along with one set of experimental data. To use this command, type a **5** at the main menu, and then the computer will ask for the three species you wish to monitor. After that it will ask for which set of initial conditions you wish to use, and finally it will ask if you want to print the table.

After that, the computer will integrate the rate equations, and print out a table. The first screen of the table will print out on the screen, and then on the printer if you wanted a hardcopy, and then the computer will say to hit return to go on. It will then wait until you press return and then print the next screen of the table until the whole table has been printed.

After printing the table, the computer will go through the same procedure for changing the constants that it did in the graphing routine, and then will ask if you wish to integrate again, and if you do, it will go back to the point where it asked for which set of initial conditions you wanted to graph.

6. Change constants

This command will allow you to change some of the constants that the program needs. To use the command type in a **6** at the main menu, and then the computer will respond with:

```
In order to keep the current value:
   Enter negative values for numbers
   Enter spaces for character strings


Scale Factor:  0.000066700
New Value:
```

This means that anytime the computer asks for a number in this section of the program, if you type in a negative value, the old value of the number will remain unchanged, and if you type in a positive or zero value the value you type in will replace the old value. The same is true for character strings if you type a space; the old value will be unchanged, but if you type anything else, what you type will replace the old string.

The program then shows you the current value of *Scale Factor*, and prompts you to put in a new value for it. The scale factor is multiplied by the interpolated value of the intensity of light (which is a value between one and zero) to give the value of ϕI_a where $I_a = I_0(1 - 10^{\epsilon cl})$. This was done because there is not always enough data to find these things through calculations, so we offer a hit and miss method of finding the value; but the scale factor could be found through calculations if there is enough data available.

After getting the scale factor, the program will print out the new value and then go on to prompt for the time corresponding to the end of the integration.

After getting the infinite time value, the computer will type out:

```
There are 20 steps in the mechanism to look at
Which step's constant do you want to look at?
(?  for all, or numbers and subranges separated by commas)
```

This operates the same way as it does in commands 4 and 5, you can look there for details, but after you enter the numbers it is different. The computer will type back to you:

```
R + I ----> RI
k7 = 3.12


New Value:
```

This will be done for each step you want to look at, and you may enter a new value for the constant, or enter a negative number to leave it the same.

After the list of steps you have entered is finished, the computer will ask if you want to change any more constants. If you type **Y** the computer will take you back to the section where it asks for a list of steps to look at, otherwise it will take you to the main menu.

7. Change Mechanism

This command is the one that allows you to initially enter the mechanism into the program in symbolic form. The computer will respond to your command by splitting the screen so that the top part of the screen contains a submenu, and the bottom part will contain the mechanism, the rate constants, the branching ratios and the names of the species in the reaction. There are 8 commands in the submenu to allow you to add to or change a step in the mechanism, delete a step from the mechanism, move a step to a different place in the mechanism listing, switch the position of two steps in the mechanism listing, print out the mechanism listing, scroll the screen if there are more mechanism steps than fit on one screen, and to return to the main menu.

submenu commands 1 and 9: Add or change mechanism step

When you use this command, the computer will first prompt you with the four types of mechanism steps that there are, like so:

```
1) A -hv-> (products)          3) A + B ----> (products)
2) A ----> (products)          4) A + B + C ----> (products)

Your choice?
```

If you are changing a step the computer will also tell you what the step is now. You may now enter the number of the reaction type you want. If you change the type of the reaction, then you may get some messages telling you that some species have been removed from the reaction, this is normal, because when you change the type of a reaction it will delete the current step and then allow you to reenter a new step.

The next thing that will happen is that the computer will ask for reactant A. You should now type in the name of reactant A. If that name is not in the mechanism right now the computer will say so and ask if you have made an error; if you say **Y** then the computer will ask for reactant A again. Then if there is more than one reactant the computer will do the same thing for the rest of the reactants.

Now the computer will ask for product A, and you may then type it in as above. Then the computer will ask if there is another product, and if there is it will ask for that one too. you may

have up to *four* products in one mechanism step. You may change this by changing the paramater *mazr* in the program source code.

When there are no more products, the computer will ask for the k, or the branching ratio if it is a photochemical step.

Submenu command 2, Delete a mechanism step

This command will ask you for the step you want deleted and then remove it from the mechanism.

Submenu command 4, Move mechanism step

This command will ask you which step you want to move, and where you want to move it to, and then move that step to that place, and move others up or down one so that it fits in.

Submenu command 5, Switch two mechanism steps

This command will ask for two mechanism steps to switch, and then switch them

Submenu command 6, Print mechanism

This command will print the mechanism on your printer in the same format that it is on the screen.

Submenu command 7, Scroll screen

If the mechanism is too large to fit on one screen, then you may use this command to show different parts of it. It will ask you which step you want to be the first one on the screen, and then it will reprint the mechanism with that step being the first one. However, if the step you want to be at the top is less than 18 steps from the end of the mechanism, then the last 18 steps of the mechanism will be printed.

Submenu command 8, Unused

Submenue command 9, Return to main menu

This command will return you to the main menu screen

8. Change Initial Conditions

This command will allow you to set the number of sets of initial concentrations, the initial concentrations of each species in the reaction for each set, and also to change the names of the species in the reaction. To use this command type an **8** at the main menu and then the computer will respond by asking for a new number of sets of initial conditions, and then after getting that will type:

```
There are 10 species to look at:
1) I2              2) I              3) R 4) RO2
5) RO2I            6) O2             7) I*          8) R*
9) RI              10) Hydrocarb

Which species do you want to look at?
(? for all, or numbers and subranges separated by commas)
```

Now you may enter the numbers of the species you wish to change the initial concentrations or the names of, just like in command four (integrate and graph). The first thing the computer will respond with is:

```
Species number 4:  "RO2 "
New name:
```

Now you may enter a new name, or type spaces to keep the old name. The computer will then type back the name, to confirm that the name is right, then it will ask for initial concentrations for each set of initial concentrations that you are set up for. The computer will go through this for each species in the list and then it will ask if you want to change any more, and if you do, it will go back to the point where it types out what each species is, otherwise it will take you back to the main menu

9. Save mechanism

This very simple command will ask for the name of a file and then will save the mechanism, the initial concentrations, the number of sets of initial concentrations, and the scale factor in the file so that they can be loaded at a later time by command one (load mechanism file). If there is an error the program will ask for the file name again.

0. Exit program

This command will end the program execution, and put you back in the operating system.

Appendix A: Program Listing for KinSim `block data stats`

```
      implicit double precision (a-h,o-z)


      parameter (maxn = 50)
C        Constant, maximum number of rate laws

      integer wksize
      parameter (wksize = 4*maxn + maxn*13 + maxn*(maxn+1) )
C        Constant values for common block of INSL variables

      parameter (maxinit = 3)

      double precision x/0.0/, h/0.1d-3/, y(maxn, maxinit),
     * xend/0.0/, tol/0.1d-7/, wk(wksize)
      integer n/0/, meth/2/, miter/2/, ier/0/, iwk(maxn), index/1/
      common/imsldata/ n,x,h,y,xend,tol,meth,miter,index,iwk,wk,ier
C        Variables for calling INSL
C            n = number of diff eqns
C            x = place to start integration
C            h = stepsize
C            y = Initial values for function (concentrations)
C         xend = value to stop integration at
C          tol = tolerance value for function
C         meth = method of integration (2 is stiff gear method)
C        miter = method of iteration (2 is chord method with
C                jacobian matrix internally calculated)
C          iwk = integer work vector
C           wk = real work vector
C          ier = error flag variable
C        index = command variable (1 means this is the first call to routine)


      parameter (maxm = 50)
      parameter (maxr = 4)

      integer M/0/, ninit/0/, rtype(maxm), nlhs(maxm), nrhs(maxm),
     *  rhs(maxr, maxm), lhs(maxr, maxm)
      double precision k(maxm)/maxm*0/, sf/0/, tinf/0/
      character*14 spcs(maxn)
      character*20 mechnam/'NONE                '/
      common/mechanism/ mechnam, spcs, M, ninit, K, sf, tinf,
     *  rtype, nlhs, nrhs, lhs, rhs
C       Variables for defining the mechanism
C          mechnam = name of file with mechanism in it
```

```
C            spcs = labels with the names of species in the rxn
C               m = number if steps in the mechanism
C               K = K's or scale factors for each step in the mechanism
C

      parameter ( maxdata = 1000 )

      character*20 flashnam/'NONE                '/
      integer nfpoints/0/
      double precision ftime(maxdata),fdata(maxdata)
      common/flash/ flashnam, nfpoints, ftime, fdata
C         Variables for data on flash profile
C           flashnam = name of file with flash profile in it
C           nfpoints = number of data points
C              ftime = time of data point
C              fdata = data points

      character*20 exprnam(maxinit)/maxinit*'NONE                '/
      integer nepoints(maxinit)/maxinit*0/
      double precision etime(maxdata,maxinit),edata(maxdata,maxinit)
      common/experemental/ exprnam, nepoints, etime, edata
C         Variables for experemental data
C            exprnam = name of file with experimental data
C           nepoints = number of experimental data points
C              etime = time for data point
C              edata = data point

      logical graphrun(maxinit)/maxinit*.false./
      double precision gdata(maxinit,800,2)
      integer out(3),gpnts(maxinit)/maxinit*0/, monitor
      common /graphing/ graphrun,out,gdata,gpnts,monitor

C*********** end block data subprogram ************************************
      end


      integer function instr(start,str,substr)
C        This function finds a substring in a string.
      implicit none
      integer start,pl,ls,lss,tmp
      character*(*) str,substr

      ls=len(str)
      lss=len(substr)
C        get length of strings
      tmp=0
      pl=start
      do 10 while ((pl .le.  ls-lss+1) .and.  (tmp .eq.  0))
        if (substr .eq.  str(pl:pl+lss-1)) tmp=pl
        pl=pl+1
   10 continue
```

```
C       Search string until found, or until we run out of data
C       to search in the main string
      instr=tmp
      return
      end



C       file that contains the Get_Mech and Get_Exp_Data subroutines
      include 'getdata.for/list'
C       This subroutine loads data for the flash profile, and for the
C        experemental data.
C        The first line of teh file must contain the number of points,
C        and then lines must alternate, time/data until the end of the file

      subroutine load_data(nam, pts, t, d, eflag)
      implicit none
      character*20 nam
      integer pts,eflag,tmp
      double precision t(*),d(*)

      open(unit=2, file=nam, err=130, status='OLD', iostat=tmp,
     *   form='formatted')
C       Open file
  100 format(I15)
      read(2,100) pts
C       read number of points
  110 format(f30.14/f30.14)
      do 120 tmp=1,pts
        read(2,110) t(tmp),d(tmp)
  120 continue
C       read out all time and data.
      close(unit=2, err=132, iostat=tmp)
      eflag=0
      return
C       close file and return signaling everything is OK

  130 write(6,*)' Error in opening '//nam
      write(6,*)'0iostat equals ',tmp
      pts=0
      eflag=1
      return
  131 write(6,*)'0Error in reading '//nam
      pts=0
      eflag=1
      return
  132 write(6,*)' Error in closing '//nam
      write(6,*)'0iostat equals ',tmp
      pts=0
      eflag=1
      return
```

```
C       Error handling routines signal that something is wrong and then return
        end


C       This subroutine loads the mechanism from a data file where it
C       was saved in an earlier session
        subroutine file_input
        implicit none
        include 'commondef.for/list'
        integer maxm,maxn,maxdata,wksize,maxinit,maxr
        parameter (maxm = 50)
        parameter (maxr = 4)
        parameter (maxn = 50)
        parameter ( maxdata = 1000 )
        parameter (wksize = 4*maxn + maxn*13 + maxn*(maxn+1) )
        parameter (maxinit = 3)
        double precision x, h, y(maxn, maxinit),
     *  xend, tol, wk(wksize)
        integer n, meth, miter, ier, iwk(maxn), index
        common/imsldata/n,x,h,y,xend,tol,meth,miter,index,iwk,wk,ier

        integer M, ninit, rtype(maxm), nlhs(maxm), nrhs(maxm),
     *  lhs(maxr, maxm), rhs(maxr, maxm)
        double precision k(maxm), sf, tinf
        character*14 spcs(maxn)
        character*20 mechnam
        common/mechanism/ mechnam, spcs, M, ninit, K, sf, tinf,
     *      rtype, nlhs, nrhs, lhs, rhs

        character*20 flashnam
        integer nfpoints
        double precision ftime(maxdata),fdata(maxdata)
        common/flash/ flashnam, nfpoints, ftime, fdata

        character*20 exprnam(maxinit)
        integer nepoints(maxinit)
        double precision etime(maxdata, maxinit),edata(maxdata, maxinit)
        common/experemental/ exprnam, nepoints, etime, edata

        logical graphrun(maxinit)
        integer out(3),gpnts(maxinit),monitor
        double precision gdata(maxinit,800,2)
        common /graphing/ graphrun, out, gdata, gpnts, monitor
```

```
C       access block data area
      integer tmp, instr, tmp1
      external instr

   14 format('$New name of file with mechanism:  ')
    5 write(6,14)
   10 format(1a20)
      read(5,10) mechnam
      if (instr(1, mechnam, '.')  .eq.  0)
     *    mechnam=mechnam(1:instr(1,mechnam,' ')-1)//'.mech'
      open(unit=2, file=mechnam, err=100, status='old', iostat=tmp,
     *   form='formatted')
C      get file name and open it

   20 format(1I5)
      read(2,20) n
C       number of diff.  eqns.  also number of species in reaction
      read(2,20) ninit
C       number of initial conditions
   30 format(1a14)
   32 format(1d30.22)
      do 40 tmp=1,n
        read(2,30,err=110) spcs(tmp)
        do 39 tmp1=1,ninit
          read(2,32,err=110) y(tmp,tmp1)
   39   continue
   40 continue
C       read in the name of species in the mechanism, and its
C       its initial concentrations


      read(2, 50) sf
C       scale factor.  used to adjust the light intensity
      read(2,50) tinf
C       infinity time value
      read(2,20) m
C       number of mechanism steps
   50 format(d30.22)
      do 60 tmp=1,m
        read(2,20) rtype(tmp)
C         type of reaction
        read(2,20) nlhs(tmp)
C         number of reactants(left hand side)
        do 51 tmp1=1,nlhs(tmp)
          read(2,20) lhs(tmp1,tmp)
   51   continue
C         read in each reactant
        read(2,20) nrhs(tmp)
```

```
C         number of products(right hand side)
          do 52 tmp1=1,nrhs(tmp)
            read(2,20) rhs(tmp1,tmp)
   52     continue
C         read in each product
          read(2,50,err=110) k(tmp)
C         K, or branching ratio for reaction
   60   continue
C        read in the number of steps in the mechanism and read in
C        the K's or scale factors on each one

        close(unit=2, err=120, iostat=tmp)
        return
C         close the file and return

C         error handling for file i/o
  100   write(6,*)' Error opening '//mechnam
        write(6,*)'0iostat equals ',tmp
        goto 5
  110   write(6,*)'0Error reading '//mechnam
        close(unit=2, err=120, iostat=tmp)
        goto 5
  120   write(6,*)' Error closing '//mechnam
        write(6,*)'0iostat equals '.tmp
        goto 5
        end


C       gets data for light flash
        subroutine getflash
        implicit none
        include 'commondef.for/nolist'
C         get access to the block data area
        integer lerr, instr, tmp
        external instr
        character*1 tmpansr


   10   format(1a1)
        write(6,*)
        write(6,*)'File with flash profile data is:  "'//flashnam//'"'
        tmpansr='Y'
C         set default answer
        if (nfpoints .ne.  0) then
```

```
C       If there is no flash data, automatically change it
  283   format('$Change the flash profile?  ')
        write(6,283)
        read(5,10) tmpansr
        call caps(tmpansr, tmpansr)
      endif
      if (tmpansr .eq.  'Y') then
        lerr = 1
        do 30 while (lerr .eq.  1)
C          while error condition keep trying
  284     format('$Name of file with flash profile?  ')
          write(6,284)
   20     format(A20)
          read(5,20) flashnam
          if (instr(1,flashnam,'.')  .eq.  0)
     *       flashnam=flashnam(1:instr(1,flashnam,' ')-1)//'.flash'
C             Read in name, and add on extention
          call load_data(flashnam, nfpoints, ftime, fdata, lerr)
C               load data in, and set error flag
   30   continue
      endif
      return
      end


C       This subroutine loads in experemental data
      subroutine getexpr
      implicit none
      include 'commondef.for/nolist'
C       get access to the block data area
      integer lerr, instr, tmp
      external instr
      character*1 tmpansr

   10 format(1a1)
      write(6,*)
      do 783 tmp=1,ninit
C         Loop for each set of initial conditions
   82   format (' File with experemental data set ',i2,
     *     ' is:  "',1a20,'"')
        tmpansr='Y'
        write(6,82) tmp,exprnam(tmp)
        if (nepoints(tmp) .ne.  0) then
```

```
C          Only ask tc change data if there is already data
  285     format('$Change the experemental data file?  ')
          write(6,285)
          read(5,10) tmpansr
          call caps(tmpansr,tmpansr)
        endif
        if (tmpansr .eq.  'Y') then
          lerr = 1
          do 50 while (lerr .eq.  1)
C             Keep looping while there is an error condition
  286       format('$Name of file with experemental data?  ')
            write(6,286)
   20       format(1a20)
            read(5,20) exprnam(tmp)
            if (instr(1,exprnam(tmp),'.')  .eq.  0) exprnam(tmp)=
     *       exprnam(tmp)(1:instr(1,exprnam(tmp),' ')-1)//'.expr'
C                 Get name and tack an extention on it
            call load_data(exprnam(tmp), nepoints(tmp),
     *         etime(1,tmp), edata(1,tmp), lerr)
C                 Load in data
   50     continue
        endif
  783 continue
C       gets experemental data
C
      return
      end

C ***************************************************************************


C       file that contains the display() and update() subroutine
      include 'play.for/list'
C       This function gives the maximum value of a array of doubles
      double precision function arrmax(c,a)
      implicit none
      integer c,l
      double precision a(c),t

      t=-9.9d-38
      do 10 l=1,c
        if (a(l) .gt.  t) t=a(l)
   10 continue
      arrmax=t
      return
      end
```

```
C       this function gives the minimum value of an array of doubles
      double precision function arrmin(c,a)
      implicit none
      integer c,l
      double precision a(c),t

      t=9.9d37
      do 10 l=1,c
        if (a(l) .lt.  t) t=a(l)
   10 continue
      arrmin=t
      return
      end


C       This subroutine sets up the screen and also some values for
C       outputting a table or a graph
      subroutine setup(graphit)
      implicit none
      logical graphit
      include 'commondef.for/nolist'
      character*1 tmp
      double precision dmax,dmin,tmax,tmin,ftmp,ftmp1
      character*2 taxis, daxis
      character*14 monsp
      integer lp,lp1,findspcs
      double precision arrmax, arrmin
      external arrmax,arrmin,findspcs


    3 format(1a1)
   47 format(i5)
      if (graphit) then
C         If graph then....
       monitor = -1
       do 587 while (monitor .eq.  -1)
  710    format('$Species to monitor:  ')
         write(6,710)
   48    format(1a14)
         read(5,48) monsp
         monitor=findspcs(monsp)
         if ((monitor .lt.  1) .or.  (monitor .gt.  n)) monitor=-1
  587  continue
```

```
C       get the species to monitor on the graph
        write(6,*)
        dmax=-9.99e37
        dmin=9.99e37
        do 734 lp=1,ninit
          tmp=' '
          do 737 while((tmp .ne. 'Y') .and. (tmp .ne. 'N'))
   18       format('$Do you want to graph initial concentration set',
     *           i2,' ?')
            write(6,18) lp
            read(5,3) tmp
            call caps(tmp,tmp)
  737     continue
C         find out if this initial concentration set is to be graphed
          graphrun(lp)= tmp .eq. 'Y'
          if (graphrun(lp)) dmax=max(dmax, y(monitor,lp))
          if (graphrun(lp)) dmin=min(dmin, y(monitor,lp))
C           if so then find out current limits on graph from initial
C           concentrations.
  734   continue
        write(6,*)
        taxis='XB'
        daxis='YL'
C         set up for drawing graph paper
        tmin=0.0d0
        tmax=-9.99e37
        do 923 lp=1,ninit
          if (graphrun(lp))
     *      tmax=max(arrmax(nepoints(lp), etime(1,lp)), tmax)
          if (graphrun(lp))
     *      dmax=max(arrmax(nepoints(lp), edata(1,lp)), dmax)
          if (graphrun(lp))
     *      dmin=min(arrmin(nepoints(lp), edata(1,lp)), dmin)
  923   continue
C         get maximum and minimum values for data and time
        ftmp=(dmax-dmin)*0.10
        dmax=dmax+ftmp
        dmin=dmin-ftmp
C         expand window for data by 10% on each side
        write(6,*)' Window for concentration data is:  '
   13   format(' Min:  ',d18.10,'  Max:  ',d18.10)
        write(6,13) dmin, dmax
        write(6,*)'Input new values (negative to keep these)'
  713   format('$Values:  ')
        write(6,713)
   15   format(d30.22,d30.22)
        read(5,15)ftmp,ftmp1
        if (ftmp .ge. 0.0) dmin=ftmp
        if (ftmp1 .ge. 0.0) dmax=ftmp1
```

```
C          allow user to change window for concentration data.
        call clear_text
        call set_viewport(0.0, 0.0, 767.0, 459.0)
        call text_scroll(1,4)
        call draw_graphpaper('LIN', 10, 5, 'LIN', 5, 5, 'WHITE')
        call label_numaxis(taxis, 'Time', tmin, tmax, .true.)
        call label_numaxis(daxis, 'Concentration', dmin, dmax, .true.)
        call set_color('CYAN',1)
        call set_color(' ',3)
C          set up screen and draw graphpaper
        do 992 lp1=1,ninit
          if (graphrun(lp1)) call place(0.0, y(monitor,lp1), lp1,lp1)
          do 1925 lp=1,nepoints(lp1)
            if (graphrun(lp1))
     *          call place(etime(lp,lp1), edata(lp,lp1), lp1,lp1)
 1925     continue
  992   continue
C         put up experemental data on the screen
      do 802 lp1=1,ninit
        gpnts(lp1)=0
  802 continue
C        don't save the places of the experemental data
        call set_writemode('CO')
C          set COmplement write mode, so that data can be erased later

      else
C         and if we want a table.....

        write(6,*)' Put in the three numbers for species to monitor'
   65   format('$Species ',i3,':  ')
   66   format(1ai4)
        do 78 lp=1,3
          out(lp)=0
          do 100 while (out(lp) .eq.  0)
            write(6,65) lp
            read(5,66) monsp
            out(lp)=findspcs(monsp)
  100     continue
   78   continue
```

```
C          get the three species to monitor
           call clear_text
           call set_viewport(0.0, 0.0, 767.0, 399.0)
           call set_window(0.0, 0.0, 767.0, 399.0)
           call text_scroll(1,4)
           call set_textsize(1,)
           call set_writemode('RE')
           call box(0.0, 370.0, 767.0, 399.0)
           call move(0.0,0.0)
           call line(0.0, 399.0)
           do 110 lp=1,5
             call move(float(lp)*153.6-1.0, 0.0)
             call line(float(lp)*153.6-1.0, 399.0)
  110      continue
           call move(5.0, 395.0)
           call text('     Time     ')
           call move(158.6, 395.0)
           call text(' Experimental ')
           do 120 lp=1,3
             call move((158.6+lp*156.6), 395.0)
             call text(spcs(out(lp)))
  120      continue
C           set up screen and draw the table
         endif
         return
         end


C         This much used little subroutine will take a string and then
C         replace all the lower case letters with upper case letters, while
C         not touching the rest of the characters
         subroutine caps(tmp,tmp1)
         implicit none
         character*(*) tmp,tmp1
         integer i

         tmp1=tmp
         do 10 i=1,len(tmp)
           if ((ichar(tmp(i:i)) .ge.  ichar('a'))
     *      .and.  (ichar(tmp(i:i)) .le.  ichar('z')))
     *      tmp1(i:i)=char(ichar(tmp(i:i))-ichar('a')+ichar('A'))
  10     continue
         return
         end


C          This usefull little function will remove a positive integer
C          from a string and update pointers so that it can be called
```

```
C         again to get the next integer in the string
      integer function pullnum(s,p)
      implicit none
      character*(*) s
      integer p,tmp

      tmp = 0
      do 10 while ((len(s) .ge.  p) .and.  ((s(p:p) .lt.  '0')
     *          .or.  (s(p:p) .gt.  '9')))
        p=p+1
   10 continue
C       find next number
      do 20 while ((len(s) .ge.  p) .and.
     *       ((s(p:p) .ge.  '0') .and.  (s(p:p) .le.  '9')))
        tmp=10*tmp + ichar(s(p:p)) - ichar('0')
        p=p+1
   20 continue
C       get its value
      do 30 while ((len(s) .ge.  p) .and.  (s(p:p) .eq.  ' '))
        p=p+1
   30 continue
C     Move pointer to next non-space character
      pullnum=tmp
      return
      end

C       This procedure updates the K's after graphing or printing a table
C       without disturbing the screen.
      subroutine update(smlp)
      implicit none
      logical smlp, change
      include 'commondef.for/nolist'
      integer pos,pos1,lp,spos
      character*1 tmp
      character*60 longtmp
      integer instr, pullnum
      external instr, pullnum
      double precision dbltmp
      character*16 kstr

      tmp='Y'
      do 32767 while (tmp .eq.  'Y')
    3   format(1a60)
    4   format(a1)
    5   format(' There are ',I<int(alog10(floatj(m))+1.0)>,
     *                                  ' mechanism steps')
        write(6,5) m
        write(6,*) ' Which step's constants do you want to look at?'
        write(6,*) ' (?  for all,or numbers and subranges seperated',
     *   ' by commas)'
        read(5,3) longtmp
```

```
C          get K's to be updated
          pos=0
          spos=1
          do 100 while (len(longtmp) .ge.  spos)
            if (instr(1,longtmp,'?')  .ne.  0) then
              pos=1
              pos1=m
              spos=len(longtmp)+1
            else
              pos=pullnum(longtmp,spos)
              pos1=pos
              if (longtmp(spos:spos) .eq.  '-') pos1=pullnum(longtmp,spos)
            endif
C           find range of k's to be updated
            if (pos .gt.  pos1) then
              lp=pos
              pos=pos1
              pos1=lp
            endif
C             switch range if top is smaller than bottom
            do 154 lp=max(pos,0),min(max(pos1,0),m+1)
              if ((lp .gt.  0) .and.  (lp .le.  m)) then
C                   check to make sure k exists
    9           format(' Step number ',i<int(log10(float(lp)))+1>,':')
                write(6,9) lp
   11           format('     ',1a16)
                call makek(lp, kstr)
                write(6,11) kstr
  701           format('$Change?  ')
                write(6,701)
                read(5,4) tmp
                call caps(tmp,tmp)
C                   print current value and ask for change
  704           format('$New value:  ')
                if (tmp .eq.  'Y') write(6,704)
   13           format(d30.14)
                if (tmp .eq.  'Y') read(5,13) k(lp)
C                  if change wanted, then get new value
              endif
  154       continue
  100     continue
  707     format('$Change more constants ?  ')
          write(6,707)
          read(5,4) tmp
          call caps(tmp,tmp)
C           Ask if more changes are desired
```

```
32767 continue
      do 4445 while ((tmp .ne.  'I') .and.  (tmp .ne.  'M'))
  714   format('$Integrate again(I) or go back to Menu(M) ?  ')
        write(6,714)
        read(5,4) tmp
        call caps(tmp,tmp)
C         find out if whe user wants to integrate the equations again
 4445 continue
      h=0.1d-3
      tol=0.1d-7
      index=1
      meth=2
      miter=2
C      set up constants for integration
      smlp = tmp .eq.  'I'
C       set flag to loop back again and re-integrate
      return
      end


C        This subroutine saves the mechanism and other values so that
C        we can come back later after we get sick of playing with
C        this silly computer.
      subroutine savemech
      implicit none
      include 'commondef.for/nolist'
      character*20 tmpnam
      integer instr, tmp, tmp1
      external instr

   10 format('0Current name of mechanism file:  "',1a20,'"')
      write(6,10) mechnam
   20 format('$New name(Space to keep the same name):  ')
      write(6,20)
   30 format(1a20)
      read(5,30) tmpnam
      if (tmpnam.ne.'                                    ')  mechnam=tmpnam
      if (instr(1,mechnam,'.')  .eq.  0)
     *   mechnam=mechnam(1:instr(1,mechnam,' '))//'.mech'
C       get name and tack an extention on the end
      open(unit=2, file=mechnam, err=100, status='new', iostat=tmp,
     *   form='formatted')
```

```
C       open file
   40 format(1i5)
      write(2,40) n
      write(2,40) ninit
   50 format(1a14)
   60 format(1d30.22)
      do 200 tmp=1,n
        write(2,50) spcs(tmp)
        do 210 tmp1=1,ninit
          write(2,60) y(tmp, tmp1)
  210   continue
  200 continue
C       save chemical names and initial concentrations
      write(2,60) sf
      write(2,60) tinf
C        save scale factor and infinity time value
      write(2,40) m
      do 220 tmp=1,m
        write(2,40) rtype(tmp)
        write(2,40) nlhs(tmp)
        do 201 tmp1=1,nlhs(tmp)
          write(2,40) lhs(tmp1,tmp)
  201   continue
        write(2,40) nrhs(tmp)
        do 202 tmp1=1,nrhs(tmp)
          write(2,40) rhs(tmp1,tmp)
  202   continue
        write(2,60) k(tmp)
  220 continue
C        save mechanism
      close(unit=2, err=120, iostat=tmp)
C        close file and return
      write(6,*)'Saved'
      return

C       error handling
  100 write(6,*)' Error opening '//mechnam
      return
  110 write(6,*)' Error reading '//mechnam
      return
  120 write(6,*)' Error closing '//mechnam
      return
      end

C       This subroutine lets you change the number of initial
C       conditions and the names of species in the reaction
```

```
C       and also initilal concentrations
      subroutine playspcs
      implicit none
      include 'commondef.for/nolist'
      integer tmp,lp1,lp2, pos, pos1, spos, pullnum, instr
      external pullnum, instr
      double precision dtmp
      character*14 ntmp
      character*1 ctmp
      character*60 longtmp

   10 format(1i5)
  320 format('$New value:  ')
      write(6,*)' In order to keep the current value:'
      write(6,*)'    Enter negative values for numbers,'
      write(6,*)'    and spaces for character strings.'
      write(6,*) ' '
  330 format(' Number of sets of initial concentrations:  ',1i5)
      tmp=0
      do 810 while ((ninit .eq.  0) .or.  (tmp .eq.  0))
        write(6,330) ninit
        write(6,320)
        read(5,10) tmp
        if (tmp .gt.  0) ninit=tmp
  810 continue
C       change number of initila conditions
      write(6,330) ninit
      write(6,*)
      ctmp='Y'
      do 32767 while (ctmp .eq.  'Y')
    3   format(1a60)
    4   format(a1)
  720   format('  There are ',i3,' species to look at.')
        write(6,720) n
        write(6,*)
  725   format(x,4(i3,'.  ',1a14,x))
        do 811 lp1=1,n,4
          write(6,725) (lp2,spcs(lp2), lp2=lp1,min(lp1+3,n))
  811   continue
C         write out the species in the reaction
        write(6,*)
        write(6,*)
        write(6,*) ' Which species do you want to look at?'
        write(6,*) ' (?  for all,or numbers and subranges seperated'.
     *    ' by commas)'
        read(5,3) longtmp
        pos=0
        spos=1
C         prompt user for the changes to be made
        do 786 while (len(longtmp) .ge.  spos)
```

```
C          loop through the input string
          if (instr(1,longtmp,'?')  .ne.  0) then
            pos=1
            pos1=n
            spos=len(longtmp)+1
          else
            pos=pullnum(longtmp,spos)
            pos1=pos
            if (longtmp(spos:spos) .eq.  '-') pos1=pullnum(longtmp,spos)
          endif
C             get the range to e changed
          if (pos .gt.  pos1) then
            lp1=pos
            pos=pos1
            pos1=lp1
          endif
C           If top of range is smaller than beginning, switch them
          do 154 lp1=max(pos,0),min(max(pos1,0),n+1)
            if ((lp1 .gt.  0) .and.  (lp1 .le.  n)) then
C               Loop and make sure that the species exists
   20         format(1a14)
  340         format(' Species number',1i3,':  "',1a14,'"')
              write(6,340) lp1, spcs(lp1)
  580         format('$New name:  ')
              write(6,580)
              read(6,20) ntmp
              if (ntmp .ne.  '                              ')  spcs(lp1)=ntmp
C               get new name
              write(6,*)
              write(6,*)
              write(6,340) lp1, spcs(lp1)
   30         format(f30.20)
  350         format(' Initial concentration',1i2,':  ',d30.22)
              do 1010 lp2=1,ninit
                write(6,350) lp2,y(lp1,lp2)
                write(6,320)
                read(5,30) dtmp
                if (dtmp .gt.  0.0d0) y(lp1,lp2)=dtmp
                write(6,350) lp2,y(lp1,lp2)
                write(6,*)
 1010           continue
C               get initial concentrations
              write(6,*)
            endif
  154     continue
  786   continue
  723   format('$ Change more constants ?  ')
        write(6,723)
        read(5,4) ctmp
        call caps(ctmp,ctmp)
```

```
C          keep looping until done
32767 continue
      return
      end

C        this subroutine changes the constants, scale factor, the
C        infinity time value, and the reaction k's
      subroutine playk
      implicit none
      include 'commondef.for/nolist'
      integer tmp,lp1,lp2, pos, pos1, spos, pullnum, instr
      external pullnum, instr
      double precision dtmp
      character*14 ntmp
      character*1 ctmp
      character*60 longtmp, eqnstr
      character*16 kstr

  320 format('$New value:  ')
    3 format(1a60)
    4 format(a1)
   30 format(f30.20)
   10 format(1i5)
      write(6,*)' In order to keep the current value:'
      write(6,*)'    Enter negative values for numbers,'
      write(6,*)'    and spaces for character strings.'
      write(6,*) ' '
      write(6,*)
  360 format(' Scale factor:  ',f30.20)
      write(6,360)sf
      write(6,320)
      read(5,30) dtmp
      if (dtmp .ge.  0.0d0) sf=dtmp
      write(6,360)sf
C        get new scale factor
      write(6,*)
  410 format(' Time for infinity:  ',f30.20)
      write(6,410)tinf
      write(6,320)
      read(5,30) dtmp
      if (dtmp .ge.  0.0d0) tinf=dtmp
```

```
C       get new infinity time
        write(6,410)tinf
        write(6,*)
        ctmp='Y'
        do 2767 while (ctmp .eq. 'Y')
  729   format(' There are ',i4,' mechanism steps to look at.')
        write(6,729) m
        write(6,*) ' Which step's constant do you want to look at?'
        write(6,*) ' (? for all,or numbers and subranges seperated',
     *   ' by commas)'
        read(5,3) longtmp
C         get k's to be changed from the user
        pos=0
        spos=1
        do 111 while (len(longtmp) .ge. spos)
          if (instr(1,longtmp,'?') .ne. 0) then
            pos=1
            pos1=m
            spos=len(longtmp)+1
          else
            pos=pullnum(longtmp,spos)
            pos1=pos
            if (longtmp(spos:spos) .eq. '-') pos1=pullnum(longtmp,spos)
          endif
C           get range of k's to be changed
          if (pos .gt. pos1) then
            lp1=pos
            pos=pos1
            pos1=lp1
          endif
C           switch the range if the first is larger than the second
          do 156 lp1=max(pos,0),min(max(pos1,0),m+1)
            if ((lp1 .gt. 0) .and. (lp1 .le. m)) then
C                 make sure that the K exists
  430         format(' K(',i3,') = ',f30.20)
              call makeeqn(lp1,60,eqnstr)
              call makek(lp1,kstr)
              write(6,*) eqnstr
              write(6,*) '         '//kstr
              write(6,320)
              read(5,30) dtmp
              if (dtmp .ge. 0) k(lp1)=dtmp
              write(6,430) lp1,k(lp1)
C               get a new value
              write(6,*)
            endif
  156     continue
  111   continue
  723   format('$ Change more constants ? ')
        write(6,723)
        read(5,4) ctmp
```

```
C           ask if we are done
            call caps(ctmp,ctmp)
 2767 continue
      return
      end


C       this usefull littl function will find the actual length of the
C       string, by searching from hte end tword the front until a
C       non-space character is found.       It is unfortunate that FORTRAN
C       doss not supply this function and I had to make it myself
      integer function backlen(str)
      implicit none
      integer i
      character*(*) str

      i=len(str)
      do 10 while ((str(i:i) .eq.  ' ')  .and.  (i .gt.  1))
        i=i-1
   10 continue
      backlen=i
      return
      end

C       This subroutine assembles the mechansim step in a printable format
      subroutine makeeqn(eqn, maxlen, str)
      implicit none
      include'commondef.for/nolist'
      integer eqn,maxlen, rlen(8), s, place, tmp, backlen, instr, r
      external backlen, instr
      character*(*) str
      character arrow*7
      character*14 react(8)

      do 5 s=1,len(str)
        str(s:s)=' '
    5 continue
C       remove everyting from the print string
      arrow=' ----> '
C       set the value of the arrow
      if (rtype(eqn) .eq.  1) arrow=' -hv-> '
C       reset it if it is a photochemical reaction
      place=7
C       set length of string
      place=place+3*(nlhs(eqn)-1)
      place=place+3*(nrhs(eqn)-1)
```

```
C        add in length of ' + ' for each one that will be required
       do 10 s=1,nlhs(eqn)
         react(s)=spcs(lhs(s,eqn))
         rlen(s)=backlen(spcs(lhs(s,eqn)))
         place=place+rlen(s)
   10  continue
C      get lhs off equation ready to be added into the string
       do 20 s=1,nrhs(eqn)
         react(nlhs(eqn)+s)=spcs(rhs(s,eqn))
         rlen(nlhs(eqn)+s)=backlen(spcs(rhs(s,eqn)))
         place=place+rlen(nlhs(eqn)+s)
   20  continue
C      get rhs of equatin ready to be added into the string
       do 30 while(place .gt.  maxlen)
C        loop while the string is too long
         r=1
         do 40 s=2,nlhs(eqn)+nrhs(eqn)
           if (rlen(r) .lt.  rlen(s)) r=s
   40    continue
C          find longest string
         rlen(r)=rlen(r)-1
         place=place-1
C          chop a character off of it
   30  continue
C        start constructing string
       place=rlen(1)
       str(1:place)=react(1)(1:rlen(1))
C          add first reactant
       if (nlhs(eqn) .gt.  1) then
         do 45 r=2,nlhs(eqn)
           str(place+1:place+3)=' + '
           place=place+3
           str(place+1:place+rlen(r))=react(r)(1:rlen(r))
           place=place+rlen(r)
   45    continue
       endif
C          add an operator and each other ractant
       str(place+1:place+7)=arrow
       place=place+7
C          add arrow
       r=nlhs(eqn)+1
       str(place+1:place+rlen(r))=react(r)(1:rlen(r))
       place=place+rlen(r)
C          add first reactant
       if (nrhs(eqn) .gt.  1) then
         do 50 r=nlhs(eqn)+2,nlhs(eqn)+nrhs(eqn)
           str(place+1:place+3)=' + '
           place=place+3
           str(place+1:place+rlen(r))=react(r)(1:rlen(r))
           place=place+rlen(r)
   50    continue
```

```
C         add an operator and each other reactant
        endif
        do 60 while ((place .lt.  maxlen) .and.
     *     (instr(1,str,arrow) .lt.  maxlen/2-4))
C         while the string is lessthan the maximum length and teh arrow
C         is less than halfway accross, add a space to the begining.
C         this makes the output pretty
          str=' '//str
          place=place+1
   60   continue
        return
        end


C       this subroutine puts the k in a printable format, and also
C       makes ajustments for it being a branching ratio
        subroutine makek(eqn,str)
        implicit none
        include'commondef.for/nolist'
        integer tmp,i,eqn
        character*(*) str

        do 5 i=1,len(str)
          str(i:i)=' '
    5   continue
C       clean out string
        if (rtype(eqn) .eq.  1) then
C         if branching ratio needed then....
          tmp=0
          do 10 i=1,eqn
            if ((rtype(i) .eq.  1) .and.  (lhs(1,i) .eq.  lhs(1,eqn)))
     *         tmp=tmp+1
   10     continue
C           figure out which branching ratio is being used
          write(str(1:3),20) lhs(1,eqn)
C           put value of reactant into ratio
          str(1:1)=char(ichar('A')+tmp-1)
C           put the actual ratio into the string
          str(4:5)=' ='
C           put equaltiy operator into string
        else
C         But if it is a k...
   20     format(i<int(alog10(floatj(eqn))+2.0)> )
          write(str(1:3),20) eqn
C           put equation number into k
          str(1:1)='k'
          str(4:5)=' ='
```

```
C          put k and equality operator into string
        endif
   30 format(e11.4)
        write(str(6:16),30) k(eqn)
C          put value of constant into string
        return
        end

C         this function returns a label
        character*4 function numlab(i)
        implicit none
        integer i
        character*4 tmp

   10 format(i3,')')
        write(tmp,10) i
        numlab=tmp
        return
        end


C         This function makes, and then prints out one line of the mechanism
C         at the location of the screen directed
        subroutine putline(ln, st)
        implicit none
        include'commondef.for/nolist'
        integer ln,st,tmp,ln1
        character*80 str
        double precision xmech/164.0/, xks/574.0/, vert
        character*4 numlab
        external numlab

        ln1=max(min(ln,18),1)
        vert=379.0-20.0*ln1
C         figure out where to putit
        do 10 tmp=1,80
          str(tmp:tmp)=' '
   10 continue
C          clean the string
        if (st .le.  n) then
          str(1:4)=numlab(st)
          str(5:18)=spcs(st)
        endif
C         put in the species, if there is one
        str(19:22)=numlab(st)
        if (st .le.  m) then
          call makeeqn(st,42,str(23:64))
          call makek(st, str(65:80))
        endif
```

```
C       put in the mechanism and teh k if there is one
      xmech=164.0
      xks=574.0
      call linetext(6+ln1, 2, str)
C       print out the step
      call move(xmech,vert)
      call line(xmech,vert-19.0)
      call move(xks,vert)
      call line(xks,vert-19.0)
C       replace the lines that were erased
      return
      end


C       this function searches the equations for a particular species,
C       or moves each species number above the one specified down one
      logical function clean(num, remove)
      implicit none
      logical remove, ltmp
      integer num, lp, lp1, tmp
      include 'commondef.for/nolist'


      ltmp=.false.
C       set flag
      do 10 lp=1,m
        do 20 lp1=1,nlhs(lp)
          if (.not. remove) ltmp=ltmp .or. (lhs(lp1,lp) .eq. num)
C             if search, check for equaltiy
          if (remove .and. (lhs(lp1,lp) .gt. num))
     *        lhs(lp1,lp)=lhs(lp1,lp)-1
C             if remove, decrement if above value
   20   continue
        do 30 lp1=1,nrhs(lp)
          if (.not. remove) ltmp=ltmp .or. (rhs(lp1,lp) .eq. num)
          if (remove .and. (rhs(lp1,lp) .gt. num))
     *        rhs(lp1,lp)=rhs(lp1,lp)-1
   30   continue
C         do the same for the rhs
   10 continue
      if (remove) then
        write(6,*) ' ',spcs(num),' removed from mechanism'
        n=n-1
        do 50 lp=num,max(n,num)
          spcs(lp)=spcs(lp+1)
          do 60 lp1=1,maxinit
            y(lp1,lp)=y(lp1,lp+1)
   60     continue
   50   continue
```

```
C          clean up and teel the user if one was removed
      endif
      clean= .not.  ltmp
C          set flag
      end


C        this function returns the place of a species, when passed its name
C        it will return a zero if the species is not there.
C        The function is case sensitive
      integer function findspcs(str)
      implicit none
      include'commondef.for/nolist'
      character*(*) str
      integer backlen,lp,tmp,l1,l2
      external backlen

      tmp=0
      l1=backlen(str)
      do 10 lp=1,n
        l2=backlen(spcs(lp))
        if ((l1 .eq.  l2) .and.  (str(1:l1) .eq.  spcs(lp)(1:l2))) tmp=lp
   10 continue
      findspcs=tmp
      return
      end
```

```
C       This subroutine either changes or adds a new mechanism step
      subroutine cstep(ch, eqn)
      implicit none
      logical ch, chtmp
      integer eqn, tmp, lp, lp1, findspcs, clean
      external findspcs, clean
      include 'commondef.for/nolist'
      character ans*1, sptmp*14, kstr*16
      double precision ktmp

      ans=' '
      chtmp = ch
      do 10 while ((ans .lt. '0') .or. (ans .gt. '4'))
        write(6,*)
        write(6,*)
        write(6,*) '   1) A -hv-> (products)      '//
     *    '3)  A  +  B  ----> (products)'
        write(6,*) '   2) A ----> (products)      '//
     *    '4) A + B + C ----> (products)'
   11   format(' Equation is currently type',i2)
        if (chtmp) write(6,11) rtype(eqn)
   12   format('$Your choice?  ')
        write(6,12)
   13   format(1a1)
        read(5,13) ans
   10 continue
C       get new reaction type
      if (ans .eq. '0') return
C       escape hatch
      tmp=rtype(eqn)
      rtype(eqn)=ichar(ans)-ichar('0')
      if (chtmp .and. (tmp .ne. rtype(eqn))) then
C         if reaction type is different, then...
        chtmp=.false.
C         flag that we are no longer just changing chemicals
        do 20 lp=1,nlhs(eqn)
          tmp=lhs(lp,eqn)
          lhs(lp,eqn)=0
          if (clean(tmp, clean(tmp, .false.)))  continue
C           check for each chemical on the left hand side, and
C           if it is no longer in the mechanism, remove it
   20   continue
        do 30 lp=1,nrhs(eqn)
          tmp=rhs(lp,eqn)
          rhs(lp,eqn)=0
          if (clean(tmp, clean(tmp, .false.)))  continue
   30   continue
```

```
C         do the same for the right hand side
        endif
        do 40 lp=1,max(rtype(eqn)-1,1)
   21     format('$Reactant ',i1,':  ')
   22     format(i1)
   23     format(' Current value of reactant ',i1,':  ',1a14)
          if (chtmp) then
            write(6,23) char(ichar('A')+lp-1), spcs(lhs(lp,eqn))
            write(6,*) 'Enter new value, or a '//
     *        'space to keep the same value'
C           if changing print out current value
          else
            write(6,*)'Enter value for'
C           otherwise ask for value
          endif
          sptmp='                    '
          do 60 while (sptmp .eq. '                              ')
C           loop untill we get a good value
            write(6,21) char(ichar('A')+lp-1)
            read(5,22) sptmp
            if (chtmp .and.  (sptmp .eq. '                              ')) then
              tmp=lhs(lp,eqn)
              sptmp=spcs(lhs(lp,eqn))
C               if a space was entered, and we are only changing, keep old value
            else
              tmp=findspcs(sptmp)
C               otherwise get new value
            endif
            if (chtmp) lpi=lhs(lp,eqn)
C               save current value, if changeing.
            if (tmp .eq.  0) then
              ans=' '
              do 70 while ((ans .ne.  'Y') .and.  (ans .ne.  'N'))
                write(6,*) sptmp,' is not in the mechanism now.'
   26           format('$Did you type it correctly?  ')
                write(6,26)
                read(5,13) ans
                call caps(ans,ans)
C                 verify that a new name is not a typeo
   70         continue
              if (ans.eq.'Y') n=n+1
              if (ans.eq.'Y') tmp=n
              if (ans.eq.'Y') spcs(n)=sptmp
              if (ans.eq.'Y') y(1,n)=0.0d0
              if (ans.eq.'Y') y(2,n)=0.0d0
              if (ans.eq.'Y') y(3,n)=0.0d0
C               if it is good, create a new species in the mechanism
              if (ans.eq.'N') sptmp='                    '
```

```
C             if not, set flag to re-enter
            endif
            if (tmp .ne.  0) lhs(lp,eqn)=tmp
C               set new value for valid name
            if (chtmp .and.  (lp1 .ne.  tmp) .and.  (tmp .ne.  0)) then
              if (clean(lp1, clean(lp1, .false.)))  lp1=lp1
            endif
C               Check that an old value removed is not deleted from mechanism
C               and if so, remove it
   60   continue
   40 continue
      nlhs(eqn)=max(rtype(eqn)-1,1)
C        set number of reactants
      ans='Y'
      lp=0
      do 300 while (ans .eq.  'Y')
C          loop while there are products to be entered
        lp=lp+1
        sptmp='                '
        do 100 while(sptmp .eq.  '                                  ')
C             loop untill we get valid data
          write(6,*)
          write(6,*)
          if (chtmp .and.  ( lp .le.  nrhs(eqn))) then
            write(6,*) 'Current product ',char(ichar('A')+lp-1),' is ',
     *          spcs(rhs(lp,eqn))
            write(6,*) 'Enter a space to keep the old name'
C              inform of old data
          else
            write(6,*) 'Enter name for product ',char(ichar('A')+lp-1)
C               or ask for new data
          endif
  101     format('$Name?  ')
  102     format(1a14)
          write(6,101)
          read(5,102) sptmp
          if (chtmp .and.  (sptmp .eq.  '                                  ')  .and.
     *      (lp .le.  nrhs(eqn)))  then
            tmp=rhs(lp,eqn)
            sptmp=spcs(rhs(lp,eqn))
C              if changing, the keep old data
          else
            tmp=findspcs(sptmp)
C               otherwise search for value
          endif
          if (chtmp .and.  (lp .le.  nrhs(eqn))) lp1=rhs(lp, eqn)
```

```
C            save old value for deletion
           if (tmp .eq.  0) then
             ans=' '
             do 270 while ((ans .ne.  'Y') .and.  (ans .ne.  'N'))
               write(6,*) sptmp,' is not in the mechanism now.'
  226          format('$Did you type it correctly?  ')
               write(6,226)
               read(5,13) ans
               call caps(ans,ans)
C                check for typeo
  270        continue
             if (ans.eq.'Y') n=n+1
             if (ans.eq.'Y') tmp=n
             if (ans.eq.'Y') spcs(n)=sptmp
             if (ans.eq.'Y') y(1,n)=0.0d0
             if (ans.eq.'Y') y(2,n)=0.0d0
             if (ans.eq.'Y') y(3,n)=0.0d0
C              if not a typeo, create a new species in mechanism
             if (ans.eq.'N') sptmp='                '
C              otherwise set flag for re-entry
           endif
           if (tmp .ne.  0) rhs(lp,eqn)=tmp
C                set new value in mecehanism
           if (chtmp .and.  (lp1 .ne.  tmp) .and.  (tmp .ne.  0)
     *       .and.  (lp .le.  nrhs(eqn))) then
             if (clean(lp1, clean(lp1, .false.)))  lp1 = lp1
C              clean out any old values
           endif
  100    continue
         ans=' '
         do 90 while((ans .ne.  'Y') .and.  (ans .ne.  'N'))
   72      format('$Is there another product?  ')
           write(6,72)
           read(5,13) ans
           call caps(ans,ans)
   90    continue
C          ask if there are any more products
  300 continue
      nrhs(eqn)=lp
C          set number of products
      write(6,*)
      if (chtmp) then
        call makek(eqn,kstr)
        write(6,*)'Current value of '//kstr
        write(6,*)'Enter a negative value to keep the old one'
      endif
   76 format('$New value for k?  ')
      write(6,76)
```

```
C       prompt for new k
 78 format(d30.15)
      read(5,78) ktmp
      if (ktmp .lt.  0.0d0) then
        if (.not.  chtmp) k(eqn)=0.0d0
      else
        k(eqn)=ktmp
      endif
C       get new k
      end


C       This subroutine will change the mechanism
      subroutine playmech
      implicit none
      include'commondef.for/nolist'
      integer topm,tmp,tmp1,lp,lp1,mrtype,mnlhs,mnrhs,
     *   mlhs(maxr),mrhs(maxr),pullnum,adddir
      double precision xmech/165.0/, xks/575.0/
      character ans*1, lans*5, dt*9, tm*8
      logical clean
      external clean, pullnum
      double precision mk

      call clear_text
      call text_scroll(1,4)
      call set_writemode('OV')
      call set_viewport(0.0, 0.0, 767.0, 399.0)
      call set_window(0.0, 0.0, 767.0, 399.0)
      call move(0.0, 399.0)
      call line(767.0, 399.0)
      call line(767.0, 360.0)
      call line(0.0, 360.0)
      call line(0.0, 399.0)
      call move(1.0, 389.0)
      call text('    Diff Eqns:  ')
      call move(xmech, 389.0)
      call text('      Mechanism:  ')
      call move(xks, 389.0)
      call text('    k's ')
      call move(xmech-1.0, 399.0)
      call line(xmech-1.0, 0.0)
      call move(xks-1.0, 399.0)
      call line(xks-1.0, 0.0)
      call set_viewport(0.0, 0.0, 767 ), 359.0)
      call set_window(0.0, 0.0, 767.0, 359.0)
```

```
C         set up table
        if (m .ne.  0) then
          topm=1
          do 10 lp=topm,min(topm+17,m)
            call putline(lp,lp)
     10   continue
        endif
C         print out table
        ans=' '
        do 900 while (ans .ne.  '9')
          ans=' '
          do 30 while ((ans .lt.  '1') .or.  (ans .gt.  '9'))
            write(6,*)' 1) Add mechanism step     4) Move mechanism ',
     *        'step          7) Scroll screen'
            write(6,*)' 2) Delete mechanism step  5) Switch two ',
     *        'mechanism steps  8) '
            write(6,*)' 3) Change mechanism step  6) Print out mechanism',
     *        '          9) Return to menu'
     11     format('$Your choice?  ')
C           print out menu and prompt for command
            write(6,11)
     12     format(1a1)
            read(5,12) ans
            call caps(ans,ans)
C           get legitimate command
     30   continue
          goto (100,200,300,400,500,600,700,800,900),
     *     (ichar(ans)-ichar('0'))
          goto 900
C           and yet another poor imitation of a CASE statement


C         Add step to mechanism command
    100   m=m+1
          if (m .eq.  1) topm=1
C         increment number of mechanism steps, and if it is the first one
C         set the first step on the screen to it
          call cstep(.false., m)
C           get step
          if ((m .le.  topm+17)  and.  (m .ge.  topm)) then
            call putline(m-topm+1, m)
C             if the step added can be displayed without scrolling the
C             screen, then do so
          else
```

```
C           otherwise reprint the screen
            topm=max(m-17,1)
            do 40 lp=topm,m
              call putline(lp-topm+1, lp)
   40       continue
          endif
          goto 900

C         delete mechanism step command
  200     tmp=-1
          do 70 while (((tmp .lt.  0) .or.  (tmp .gt.  m)) .and.  (m.gt.0))
            write(6,13)
            read(5,14) lans
            lp=1
            tmp=pullnum(lans,lp)
   70     continue
C           get step to be deleted
          if (tmp .ne.  0) then
            do 80 lp=1,nlhs(tmp)
              lp1=lhs(lp,tmp)
              lhs(lp,tmp)=0
              if (clean(lp1, clean(lp1, .false.)))  continue
   80       continue
C             remove left hand side from mechanism
            do 90 lp=1,nrhs(tmp)
              lp1=rhs(lp,tmp)
              rhs(lp,tmp)=0
              if (clean(lp1, clean(lp1, .false.)))  continue
   90       continue
C             remove right hand side from mechanism
            m=m-1
C           decrement number of steps
            do 110 lp=tmp,m
              k(lp)=k(lp+1)
              rtype(lp)=rtype(lp+1)
              nlhs(lp)=nlhs(lp+1)
              nrhs(lp)=nrhs(lp+1)
              do 120 lp1=1,maxr
                lhs(lp1,lp)=lhs(lp1,lp+1)
                rhs(lp1,lp)=rhs(lp1,lp+1)
  120         continue
              k(lp)=k(lp+1)
  110       continue
C             move everything down
          if (topm+17 .gt.  m) topm=max(topm-1, 1)
          do 130 lp=topm,min(topm+17,m)
            call putline(lp-topm+1, lp)
  130     continue
```

```
C         reprint screen
        endif
        goto 900

C         change mechanism step
  300   tmp = 0
        do 50 while (((tmp .lt. 1) .or. (tmp .gt. m)) .and. (m.gt.0))
          write(6,13)
   13     format('$which mechanism step? ')
   14     format(1a5)
          read(5,14) lans
          lp=1
          tmp=pullnum(lans,lp)
   50   continue
C         get step to be changed
        if (m .ne. 0) call cstep(.true., tmp)
C         change step
        if ((tmp.ne.0) .and. (tmp .ge. topm)
     *       .and. (tmp .le. topm+17))then
          topm = max(1,min(tmp-9,m-17))
          do 60 lp=topm,min(m,topm+17)
            call putline(lp-topm+1, lp)
   60     continue
        endif
C         adjust screen pointer, and reprint screen
        if (tmp .ne. 0) call putline(tmp-topm+1, tmp)
        goto 900

C           move/switch mechanism steps
  400   continue
  500   if (m .lt. 2) goto 900
C           If there aren't enough steps, then skip this
        tmp=0
        do 140 while ((tmp .lt. 1) .or. (tmp .gt. m))
   21     format('$Mechanism step number: ')
          write(6,21)
   22     format(1a5)
          read(5,22) lans
          lp=1
          tmp=pullnum(lans, lp)
  140   continue
```

```
C          get step to move
           lp1=0
           do 150 while((lp1 .lt.  1) .or.  (lp1 .gt.  m))
             if (ans .eq.  '5') then
               write(6,23)
             else
               write(6,24)
             endif
    23       format('$Switch with?  ')
    24       format('$Move to?  ')
             read(5,22) lans
             lp=1
             lp1=pullnum(lans, lp)
   150     continue
C            get place to move it to, or swicth with
           mk=k(tmp)
           mrtype=rtype(tmp)
           mnlhs=nlhs(tmp)
           mnrhs=nrhs(tmp)
           do 160 lp=1,maxr
             mlhs(lp)=lhs(lp,tmp)
             mrhs(lp)=rhs(lp,tmp)
   160     continue
C            save in temporary variables
           if (ans .eq.  '4') then
             adddir=-1
             if (tmp .lt.  lp1) adddir=1
             do 170 lp=tmp,lp1,adddir
               rtype(lp)=rtype(lp+adddir)
               nlhs(lp)=nlhs(lp+adddir)
               nrhs(lp)=nrhs(lp+adddir)
               do 180 tmp1=1,maxr
                 lhs(tmp1,lp)=lhs(tmp1,lp+adddir)
                 rhs(tmp1,lp)=rhs(tmp1,lp+adddir)
   180         continue
               k(lp)=k(lp+adddir)
   170       continue
C              if move, then move everything inbetween to a new spot
           else
             rtype(tmp)=rtype(lp1)
             nlhs(tmp)=nlhs(lp1)
             nrhs(tmp)=nrhs(lp1)
             do 190 tmp1=1,maxr
               lhs(tmp1,tmp)=lhs(tmp1,lp1)
               rhs(tmp1,tmp)=rhs(tmp1,lp1)
   190       continue
             k(tmp)=k(lp1)
```

```
C          if switch, move the other value to its new spot
        endif
        rtype(lp1)=mrtype
        nlhs(lp1)=mnlhs
        nrhs(lp1)=mnrhs
        k(lp1)=mk
        do 210 tmp1=1,maxr
          lhs(tmp1,lp1)=mlhs(tmp1)
          rhs(tmp1,lp1)=mrhs(tmp1)
  210   continue
C          put it back, from temporary variables
        if (lp1 .gt.  tmp) then
          lp=lp1
          lp1=tmp
          tmp=lp
        endif
        if (ans .eq.  '4') then
          do 220
     *    lp=max(topm,min(topm+17,lp1)),max(topm,min(topm+17,tmp))
            call putline(lp-topm+1, lp)
  220     continue
C            print out range that was shifted
        else
          if ((lp1 .ge.  topm) .and.  (lp1 .le.  topm+17))
     *       call putline(lp1-topm+1, lp1)
          if ((tmp .ge.  topm) .and.  (tmp .le.  topm+17))
     *       call putline(tmp-topm+1, tmp)
C           print out the ones that were switched
        endif
        goto 900


C          get printout
  600   if (m .eq.  0) goto 900
        call date(dt)
        call time(tm)
        write(6,*)
        write(6,*)
        write(6,*)
        write(6,*) dt//'   '//tm
C          put date and time on
        if (topm .ne.  1) then
          topm=1
          do 230 lp=topm,min(topm+17, m)
            call putline(lp-topm+1, lp)
  230     continue
        endif
C          make sure the first ones are on the screen
        call copy_screen
```

```
C          print them out
         do 240 while (topm .lt.  m-17)
           lp1=topm
           topm=min(topm+18, m-17)
           do 250 lp=topm,topm+17
             call putline(lp-topm+1, lp)
  250      continue
C            put next set on the screen
           call copy_area(0.0, 0.0, 767.0, float(topm-lp1)*20.0-1.0)
C            print out the new ones
  240    continue
C            keep going until all are printed
         goto 900

C          scroll screen
  700    if (m .lt.  19) goto 900
C          do only if there are more thtn one screenfull
    32   format('$Which step do you want to be the first',
     *       ' one on the screen?  ')
    33   format(1a5)
         tmp=0
         do 270 while ((tmp .lt.  1) .or.  (tmp .gt.  m))
           write(6,32)
           read(5,33) lans
           lp=1
           tmp=pullnum(lans,lp)
  270    continue
C          get thestep that is desired to be on the screen
         lp1=topm
         topm=min(tmp, max(m-17,1))
C          adjust the first thing on the screen
         if (topm .ne.  lp1) then
           do 260 lp=topm, topm+17
             call putline(lp-topm+1, lp)
  260      continue
         endif
C          if the screen is not exactly the same, then reprint it
         goto 900

C            no command
  800    continue

C        exit and/or loop back to command menu
  900 continue
      return
      end

C ***************************************************************************
```

```
C       file that contains the Graph subroutine
      include 'graph.for/list'
C       this subroutine puts data up on the graph.  It also stores
C       the points that it uses, for erasure later on.
      subroutine place(t,c,marker,iset)
      implicit none
      include 'commondef.for/nolist'
      double precision t,c
      integer marker,iset

      gpnts(iset)=gpnts(iset)+1
      gdata(iset,gpnts(iset),1)=t
      gdata(iset,gpnts(iset),2)=c
C       save data point
      call plot_point(gdata(iset,gpnts(iset),1),
     *      gdata(iset,gpnts(iset),2), , marker)
C       plot point
      return
      end

C       This subroutine plots out data on a graph and prints it
      subroutine graph
      implicit none
      include 'commondef.for/nolist'
      double precision step,i,lastt,arrmax
      external evmech, evjacob,i,arrmax
      integer pl,pli
      double precision conc(maxn), cinf(maxinit)
      character*1 tmp
      character dt*9,tm*8

      do 23 pli=1,3
        if (gpnts(pli) .ne.  0) then
          do 142 pl=1,gpnts(pli)
            call plot_point(gdata(pli, pl, 1),gdata(pli, pl, 2),,0)
  142     continue
          gpnts(pli)=0
        endif
   23 continue
C       Erase any existing plots on the graph
      do 891 pli=1,ninit
C       Loop for each set of initial conditions
        if (graphrun(pli)) then
C           If it is supposed to be graphed, graph it
          lastt= arrmax(nepoints(pli), etime(i,pli))
C           find the last time value in the experemental data
          step= lastt/128.0d0
```

```
C           get step size for 128 points
          x = 0.0d0
          index=1
          tol=0.1d-7
          h=0.1d-4
          meth=2
          miter=2
C           set up values for integrating routine
          do 32 pl=1,n
            conc(pl)=y(pl,pl1)
   32     continue
C           set initial concentrations
          call place(x,y(monitor,pl1),0,pl1)
C           put first value on graph
          do 156 while (x .lt.  lastt)
C           Integration loop
            xend = x + step
C            next time value to stop at
            call dgear(n, evmech, evjacob, x, h, conc, xend,
     *          tol, meth, miter, index, iwk, wk, ier)
C                Integrate
            if (ier .gt.  128) write(6,*)' Integration error -- ',ier
            if(index .ne.  0) write(6,*)' Index=',index
            index=0
C                Error conditions
            call place(x,conc(monitor),0,pl1)
C                Put point on graph
  156     continue
          call dgear(n, evmech, evjacob, x, h, conc, tinf, tol, meth,
     *         miter, index, iwk, wk, ier)
C             Integrate to infinity time value
          if (ier .gt.  128) write(6,*)' Integration error -- ',ier
  725     format(' Data set ',i2,' at time ',f10.2,' :  ',f13.9)
          write(6,725) pl1,tinf,conc(monitor)
          cinf(pl1)=conc(monitor)
C               Write out and save infinity time value
        endif
  891 continue
      tmp=' '
      do 476 while ((tmp .ne.  'Y') .and.  (tmp .ne.  'N'))
  888   format('$Print Screen?  ')
        write(6,888)
  475   format(1a1)
        read(5,475) tmp
        call caps(tmp,tmp)
  476 continue
C        See if a hardcopy is desired
      if (tmp .eq.  'Y') then
C         If so, then set up top of screen with date and
```

```
C         infinity time values.
          write(6,*)
          write(6,*)
          write(6,*)
          write(6,*)
          call date(dt)
          call time(tm)
          write(6,*) dt//'   '//tm
          do 499 pl1=1,ninit
            if (graphrun(pl1)) write(6,725) pl1, tinf, cinf(pl1)
  499     continue
          call copy_screen
C           print out screen
        endif
        return
        end

C ***************************************************************************


C       file that contains the Table subroutine
        include 'table.for/list'
C          This subroutine prints a table with the concentrations
C          of three reactants vs.  time with the experemental data also
C          and gives a hard copy if desired
        subroutine table
        implicit none
        include 'commondef.for/nolist'
        integer linesprinted, place,lp,pl1
        character*1 tmp,prt
        double precision conc(maxdata), ftmp, ftmp1(3)
        external evmech, evjacob
        character tm*8,dt*9

        tmp='M'
        prt=' '
    3   format(a1)
        do 88 while((ichar(tmp)-ichar('0') .le.  0) .or.
     *          (ichar(tmp)-ichar('0') .gt.  ninit))
          write(6,*)' Which set of initial conditions?'
          read(5,3) tmp
   88   continue
C         get initial condition set from user
        pl1=ichar(tmp)-ichar('0')
        do 10 while ((prt .ne.  'Y') .and.  (prt .ne.  'N'))
   17     format('$Do you want to print the table?  ')
          write(6,17)
          read(5,3) prt
          call caps(prt,prt)
   10   continue
```

```
C         Ask if a hard copy is desired
      if (prt .eq.  'Y') then
        write(6,*)
        write(6,*)
        call date(dt)
        call time(tm)
        write(6,*)' Initial concentration set #'//ftmp
        write(6,*) dt//'    '//tm
        write(6,*)
        call copy_area(0.0, 362.0, 767.0, 479.0)
      endif
C         print time and date and table heading if a hard cpoy is requested
      call text_scroll(7,24)
C         set scroll area so that header is not scrolled off the screen
      linesprinted=0
      index=1
      h=0.1d-3
      tol=0.1d-7
      meth=2
      miter=2
C       set up for integration
      do 20 place=1,n
        conc(place)=y(place,pl1)
   20 continue
C       set initial concentration values
      x=0.0d0
   18 format(2x,f14.7,1x,f14.7,1x,f14.7,1x,f14.7,1x,f14.7,1x,f14.7)
   19 format(2x,f14.7,1x,14x 1x,f14.7,1x,f14.7,1x,f14.7)
C       formats for printing table
      do 25 lp=1,3
        ftmp1(lp)=conc(out(lp))
   25 continue
      write(6,19) x,ftmp1(1),ftmp1(2),ftmp1(3)
      linesprinted=linesprinted+1
C       write out the first line with initial concentrations
      do 123 place=1,nepoints(pl1)
C         loop with one stop for each point in the experemental data
        call dgear(n, evmech, evjacob, x, h, conc, etime(place,pl1),
     *    tol, meth, miter, index, iwk, wk, ier)
C         integrate
        if (ier .gt.  128) write(6,*) ' Integration error -- ',ier
        index=0
        do 30 lp=1,3
          ftmp1(lp)=conc(out(lp))
   30   continue
        write(6,18) x, edata(place,pl1),ftmp1(1),ftmp1(2),ftmp1(3)
C         write out data
        linesprinted=linesprinted+1
        if (linesprinted .gt.  17) then
```

```
C            if screen is full then....
             call move(0.0,0.0)
             call line(0.0, 399.0)
             do 32 lp = 1,5
               call move(float(lp)*153.6-1.0, 0.0)
               call line(float(lp)*153.6-1.0, 399.0)
    32       continue
C                 Patch up lines on table
             if (prt .eq.  'Y')
     *          call copy_area(0.0, 0.0, 767.0, 369.0)
C                print it out
             call text_scroll(1,4)
             write(6,*)' Hit return to go on'
             read(5,3)tmp
C                 wait for user to respond
             call text_scroll(7,24)
             linesprinted=0
           endif
   123 continue
C          when done integrate to infinite time value
         if (etime(nepoints(pl1),pl1) .lt.  tinf)then
           call dgear(n, evmech, evjacob, x, h, conc, tinf,
     *      tol, meth, miter, index, iwk, wk, ier)
           if (ier .gt.  128) write(6,*) ' integration error -- ',ier
           do 143 lp=1,3
             ftmp1(lp)=conc(out(lp))
   143     continue
           write(6,19) x,ftmp1(1),ftmp1(2),ftmp1(3)
C             and write it out
           linesprinted=linesprinted+1
         endif
         tmp=' '
         call move(0.0, 0.0)
         call line(0.0, 399.0)
         do 132 lp = 1,5
           call move(float(lp)*153.6-1.0, 0.0)
           call line(float(lp)*153.6-1.0, 399.0)
   132 continue
         call move (0.0, 0.0)
         call line(767.0, 0.0)
C          patch up the table
         if (prt .eq.  'Y')
        *  call copy_area(0.0, 0.0, 767.0, float(linesprinted)*20.0)
C            and print it out if desired
         call text_scroll(1,4)
         write(6,*)' Hit return to go on'
         read(5,3)tmp
C          wait for the user to respond
         return
         end
```

```
C ***************************************************************************

C       file that contains the mechanism subroutines
C            EVMECH(N,X,Y(N),YPRIME(N))
C       and  EVJACOB()  - which is not used in this implementation
C
      include 'mech.for/list'
C         This function generates the intensity of light as a function
C         of time using a linear interpolation of the data
      double precision function i(t)
      implicit none
      include 'commondef.for/nolist'
      double precision t,tmp
      integer itmp

      itmp=1
      if (t .le.  ftime(1)) then
        tmp=sf*fdata(1)
C         If time is less than any of the data points, use the first value
C         alone for data
      elseif (t .ge.  ftime(nfpoints)) then
        tmp=sf*fdata(nfpoints)
C         If time is greater than any of the data points, use the
C         last data point alone for data
      else
        do 10 while ((t .gt.  ftime(itmp)) .and.  (itmp .lt.  nfpoints))
          itmp=itmp+1
   10   continue
C           search for the time value that is larger than the current time
        tmp=sf*(fdata(itmp-1)+(t-ftime(itmp-1))
     *          *((fdata(itmp)-fdata(itmp-1))
     *          /(ftime(itmp)-ftime(itmp-1))))
C         Otherwise do a linear interpolation between the point before and
C         the point found
      endif
      i=tmp
      return
      end

C       subroutine used to generate a jacobian matrix..    Unused in this
C       implementation
      subroutine evjacob(n,x,y,pd)
      double precision x,y(n),pd(n,n)
      integer n
      return
      end


C         This function evaluates the velocety of a reaction mechanism
```

```
C         step.
      double precision function evalv(eqn, j, t, c)
      implicit none
      integer eqn, j, lp
      double precision t, c(j), i
      external i
      include'commondef.for/nolist'

C         Another crude implementation of a CASE statement
      goto (10, 20, 30, 40), rtype(eqn)
      write(6,*)' Unrecoverable error in mechanism --'
      write(6,*)'   No mechanism step type ',rtype(eqn)
      stop
   10 evalv= k(eqn)*i(t)
      return
C       Photo-chemical step
   20 evalv= k(eqn)*c(lhs(1,eqn))
      return
C       Single reactant.
   30 evalv= k(eqn)*c(lhs(1,eqn))*c(lhs(2,eqn))
      return
C       Two reactants
   40 evalv= k(eqn)*c(lhs(1,eqn))*c(lhs(2,eqn))*c(lhs(3,eqn))
      return
C       Three reactants
C
C       You may add your own wierd and unusual steps here if so desired
C
C

      end

C       Subroutine call by IMSL routine DGEAR
C       Evaluates differential equations
      subroutine evmech(j, t, c, dc)
      implicit none
      integer j,q,r
      double precision t, c(j), dc(j), v, evalv
      external evalv
      include 'commondef.for/nolist'

      do 5 q=1,j
        dc(q)=0.0d0
    5 continue
C     Set initial values of 0.0 into the differential equations
      do 10 q=1,m
C          Loop for each mechanism step
        v=evalv(q, j, t, c)
```

```
C          get velocity of step
         do 20 r=1,nlhs(q)
           dc(lhs(r,q))=dc(lhs(r,q)) - v
   20    continue
C          subtract velocity of mechanism step from each reactant's diff.  eqn.
         do 30 r=1,nrhs(q)
           dc(rhs(r,q))=dc(rhs(r,q)) + v
   30    continue
C          add velocity of mechanism step to each product's diff.  eqn.
   10 continue
      return
      end



C******************** Beginning of the real program!!!!  ********************
      program kinsim

      implicit none


      logical bigloop, smallloop
C       Boolean variables for terminating the loops in the program

      logical didit(9)/9*.false./
C       Boolean variables to make sure there is enough data to do
C       each part of the program

      character*1 command
C       Character for the command phase of things


      call init_graphics
      call set_color('RED',1)
      call set_color(' ',3)
C       initialize for the graphics and integration packages

      bigloop = .true.

      do 1734 while ( bigloop )

      call text_scroll(1,24)
      call clear_text
      write(6,*) char(27)//'[1;1H'
```

```
C       Clear screen ANSI code
      write(6,*)'          1.    Load Mechanism file.'
      write(6,*)
      write(6,*)'          2.    Load Flash profile file.'
      write(6,*)
      write(6,*)'          3.    Load experimental data files.'
      write(6,*)
      write(6,*)'          4.    Integrate and graph.'
      write(6,*)
      write(6,*)'          5.    Integrate and make a table'
      write(6,*)
      write(6,*)'          6.    Change constants'
      write(6,*)
      write(6,*)'          7.    Change mechanism'
      write(6,*)
      write(6,*)'          8.    Change initial concentrations'
      write(6,*)
      write(6,*)'          9.    Save mechanism and constants'
      write(6,*)
      write(6,*)'          0.    Exit program'
      write(6,*)
      write(6,*)
      write(6,*)
    1 format('$Your choice?  ')
    2 format(1a1)
      command=' '
      do 3 while ((command .lt.  '0') .or.  (command .gt.  '9'))
        write(6,1)
        read(6,2) command
        call caps(command,command)
    3 continue
C       get a legal command
      goto (1000,100,200,300,400,500,600,700,800,900)
     *    ,(ichar(command)-ichar('0')+1)
      write(6,*)'Illegal command -- ',ichar(command)
      goto 1734
C         A very crude, but effective implementation of a case statement
C         This will cause one section of code corresponding to the
C         command number to be executed


  100 didit(1)=.true.
      call file_input
      goto 1734
C       command 1:      load mechanism file


  200 didit(2)=.true.
      call getflash
      goto 1734
```

```
C       command 2:      load the flash profile (intensity of light
C                     as a function of time)


  300 if (didit(1) .or.  didit(8)) then
        didit(3)=.true.
        call getexpr
      endif
      goto 1734
C        command 3:    load experemental data(Conc vs.  time)
C        can only be done after there is a mechanism loaded, or
C        the number of sets of initial conditions has been set


  400 continue
C        Graph data:      done together with table.
  500 smallloop = didit(2) .and.  didit(3) .and.
     *  (didit(1) .or.  (didit(8) .and.  didit(6)))
C         Table.    can only be done when a mechanism file has been
C         loaded, or there is initial conc.  data and a mechanism
      didit(4)= command .eq.  '4'
      if (smallloop) call setup(didit(4))
      do 1652 while ( smallloop )
        if (didit(4)) then
          call graph
C           If we want graphical output, then integrate and graph it
        else
          call table
C           Otherwise, put out a table from the integration
        endif
        call update(smallloop)
C         update info like K's etc.  without erasing graph.
 1652 continue
C       end of smallloop
      goto 1734

  600 if (didit(1) .or.  didit(7)) then
        didit(6)=.true.
        call playk
      endif
      goto 1734
C        Change constants:  can only be done when there is a mechanism


  700 didit(7)=.true.
      call playmech
      goto 1734
```

```
C       Load, or change a mechanism from the keyboard


  800 if (didit(1) .or.  didit(7)) then
        didit(8)=.true.
        call playspcs
      endif
      goto 1734
C       Get initial conc.  data:  can only be done when there is a mechanism


  900 if (didit(1) .or.  (didit(8) .and.  didit(6))) call savemech
      goto 1734
C         Save mechanism:    can only be done when there is a mechanism


 1000 bigloop = .false
C         Set flag to end program


 1734 continue
C        end of bigloop
        call clear_screen
        call text_scroll(1,24)
        end
```